sql commands :

Sql Installation:
Linux: https://dev.mysql.com/downloads/mysql/
Windows : https://dev.mysql.com/downloads/windows/installer/8.0.html

system cls ----> clear screen (WINDOWS)
Ctrl+L / Ctrl+Shift+K —---> clear screen (Linux)

mysql>
1] show databases; (to show current databases present in mysql.)
2] CREATE DATABASE trendytech; (to create new database)
3] drop database <dbname>;   (to remove database)
4] use trendytech; (get into inside database) or (switch database)
5] select database(); (to check u r in which database)
Example:
===========

```
CREATE TABLE employee(
name varchar(20),
age int,
salary int
);
```

(int for numeric --holds whole numbers
varchar for string)holds strings upto 255 char).

show tables; (to check for which table u r connected)
describe employee; or desc employee; (to see the structure of the table)
drop table <tablename>; (to remove the table)

CREATE TABLE trendytech.employee ----->(here we created table employee and attached to trendytech db)

```
(
name varchar(20),
age int,
salary int
);
```

CRUD OPERATIONS:

CREATE = insert stmts
READ = select stmts
UPDATE = update stmts
DELETE = delete stmts

creation of table and insert statements:

EXAMPLE:

```
CREATE TABLE employee(
firstname VARCHAR(20),
middlename VARCHAR (20),
lastname VARCHAR(20),
age INT,
salary INT,
location VARCHAR(20)
);
```

Values insert into database
==============================

```
INSERT INTO employee (firstname,middlename,lastname,age,salary,location)
VALUES ('shan', 'kumar','acharya',26,10000,'Banglore');

INSERT INTO employee (firstname,middlename,lastname,age,salary,location)
VALUES ('anil', 'kumar','acharya',26,10000,'Banglore');
```

(or)   (or)   (or)

```
INSERT INTO employee VALUES ('varun', 'kumar','acharya',26,10000,'Banglore');

INSERT INTO employee (firstname,lastname,age,salary,location)
VALUES ('Rajesh','acharya',30,20000,'Banglore');
```

How to insert multiple values:
================================

```
INSERT INTO employee (firstname,middlename,lastname,age,salary,location)
VALUES ('john','kumar','acharya',30,20000,'Banglore'),('NTR','jr','acharya',35,25000,'Banglore');
```

NULL
======
For example no one should enter null values for the firstname so what to do?
how can we do that:

```
CREATE TABLE employee(
firstname VARCHAR(20) NOT NULL,
middlename VARCHAR (20),
lastname VARCHAR(20) NOT NULL,
age INT NOT NULL,
salary INT NOT NULL,
location VARCHAR(20) NOT NULL
);
```

DEFAULT VALUES:
=================

```
CREATE TABLE employee(
firstname VARCHAR(20) NOT NULL,
middlename VARCHAR (20),
lastname VARCHAR(20) NOT NULL,
age INT NOT NULL,
salary INT NOT NULL,
location VARCHAR(20) DEFAULT 'Banglore'
);
```

Here the requirement is 95% of candidates hire from banglore location only; so we provided default location is banglore in the above table.

```
INSERT INTO employee (firstname,lastname,age,salary)
VALUES ('john','acharya',30,20000);
```

Here the requirement is 95% of candidates hire from banglore location only; so we provide default location is banglore but suppose we hire candidate from diff location how can we explicitly mention in the table.

```
INSERT INTO employee (firstname,lastname,age,salary,location)
VALUES ('Rajesh','acharya',30,20000,'Hyderabad');
```

Combination of Not null and Default:
====================================

```
CREATE TABLE employee(
firstname VARCHAR(20) NOT NULL,
middlename VARCHAR (20),
lastname VARCHAR(20) NOT NULL,
age INT NOT NULL,
salary INT NOT NULL,
location VARCHAR(20) NOT NULL DEFAULT 'Banglore'
);
```

by creating this not null for location u wont give null value manually so by default it will take as 'banglore' location:
--

Primary key
Auto Increment key
Unique key

So suppose in the office we have same name person, same age, same salary then how can we differentiate From one another, the concept of "primary key" comes into picture.

Primarykey:
============

Uniquely identify each record in the table

```
CREATE TABLE employee(
Id INT,
firstname VARCHAR(20) NOT NULL,
middlename VARCHAR (20),
lastname VARCHAR(20) NOT NULL,
age INT NOT NULL,
salary INT NOT NULL,
location VARCHAR(20) NOT NULL DEFAULT 'Banglore'
);
```

```
INSERT INTO employee (id,firstname,lastname,age,salary,location)
VALUES (1,'Rajesh','acharya',30,20000,'Hyderabad');
```

So while insert above table u will get o/p with id 1 but when u r creating with same values it will create the Same id and other values.
So i need the query for the diff id for diff employees so that we have to mention id as “PRIMARY KEY”.

```
CREATE TABLE employee(
Id INT PRIMARY KEY,
firstname VARCHAR(20) NOT NULL,
middlename VARCHAR (20),
lastname VARCHAR(20) NOT NULL,
age INT NOT NULL,
salary INT NOT NULL,
location VARCHAR(20) NOT NULL DEFAULT 'Banglore'
);
```

[or]   [or]   [or]   [or]

```
CREATE TABLE employee(
Id INT,
firstname VARCHAR(20) NOT NULL,
middlename VARCHAR (20),
lastname VARCHAR(20) NOT NULL,
age INT NOT NULL,
salary INT NOT NULL,
location VARCHAR(20) NOT NULL DEFAULT 'Banglore'
PRIMARY KEY (id,name) —------> (so we can add 2 columns as PMKEY)
);
```

Now we are not unable to create same values, and for a primary key NULL is not allowed and also repeated Values are not allowed.

Simply we cannot duplicate id’s or NULL values in the tables.

AUTO INCREMENT:
==================

```
CREATE TABLE employee(
id INT AUTO_INCREMENT,
firstname VARCHAR(20) NOT NULL,
middlename VARCHAR (20),
lastname VARCHAR(20) NOT NULL,
age INT NOT NULL,
salary INT NOT NULL,
location VARCHAR(20) NOT NULL DEFAULT 'Banglore',
PRIMARY KEY (id,name) —------> (so we can add 2 columns as PMKEY)
);

INSERT INTO employee (firstname,lastname,age,salary)
VALUES ('Rajesh','acharya',30,20000);

INSERT INTO employee (firstname,lastname,age,salary)
VALUES ('kapil','acharya',30,20000);

Select * from employee; —------>(extract data from table <employee>)
```

UNIQUE KEY:
===============

U can have only one primary key and the primary key cannot hold any NULL then we should
Use UNIQUE KEY when we have to uniquely identify each record.

PMKEY cannot hold null values
UNKEY can hold NULL values
For ex in mysql a unique can hold any number of NULL values

But in some of the other famous DB’s UNKEY holds only one NULL value.

So the purpose of UNKEY is to make sure the values do not duplicate
We can have only one PMKEY but multiple unique keys in a table

```
CREATE TABLE employee(
Id int UNIQUE KEY,
firstname VARCHAR(20),
lastname VARCHAR(20),
age INT NOT NULL
);
```

Ex:
```
Insert into employee values (1, ‘kapil’, ‘sharma’, 28);
```

—---

CRUD – 2
==========

CREATE TABLE employee(
id INT AUTO_INCREMENT,
firstname VARCHAR(20) NOT NULL,
middlename VARCHAR (20),
lastname VARCHAR(20) NOT NULL,
age INT NOT NULL,
salary INT NOT NULL,
location VARCHAR(20) NOT NULL DEFAULT 'Banglore',
PRIMARY KEY (id)
);

INSERT INTO employee (firstname,lastname,age,salary)
VALUES ('kapil','acharya',30,20000);

INSERT INTO employee (firstname,lastname,age,salary)
VALUES ('shan','acharya',20,20000);

INSERT INTO employee (firstname,lastname,age,salary)
VALUES ('karna','acharya',33,20000);

Select
========
— Selecting all columns
Select * from employee;

— selecting specific column name
Select firstname, lastname from employee;

— Select by applying a where clause (filter condition)
select * from employee where age > 25;
select * from employee where firstname="shan";

By default it is case insensitive, so u want to search for specific name with capital letter name use binary
select * from employee where binary firstname="shan";

The above stmt will match the exact case and is case sensitive.

ALIAS
=======
select firstname as name, lastname as surname from employee;

UPDATE
===========
So here we updated lastname in the employee table using ‘update’ keyword.
update employee set lastname='kumar' where firstname='shan';

So here we updated location for the specific employee in the employee table using ‘update’ keyword
update employee set location='Hyderabad' where firstname='kapil';

update employee set location='Hyderabad'; (this cmd changes the entire location column to Hyderabad if u dont use where clause or it will also useful to change location column to complete hyderabad)

Suppose u want to increase salary for each person by 5000 in the table
update employee set salary= salary+5000;

Suppose we have multiple records with the same name and we want to change employee location so for that we have to use 'AND' condition.

update employee set location='Banglore' where firstname='shan' and lastname='kumar';

WARNING: be careful while updating the records and use where clause for filter records.

DELETE
=========
If u want to delete employee we can write below cmd
delete from employee where id=3;

WARNING:
delete from employee; (it will delete entire records in employee)

ALTER
=========

Alter is to alter or changes the structure of the table (ex: u can add field or u can change structure of varchar if u given 20 and change it to 30 char)
Update is to update the changes or manipulate the date in the table.

Here we added 'jobtitle' column to the table using alter cmd
alter table employee add column jobtitle varchar(30);

So if u want delete column in the table u can use alter with drop cmd
alter table employee drop column jobtitle;

So here i want increase the length of the chars of firstname using alter with 'modify'cmd
alter table employee modify column firstname varchar(30);

Here we can drop primary key with alter cmd
mysql> alter table employee drop primary key;
ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be defined as a key
If u get above error, drop the table
drop table employee and follow the steps.

1.add new INT column without AUTO_INCREMENT
2.copy column value
3.drop AUTO_INCREMENT column

CREATE TABLE employee(
id INT primary key NOT NULL,
firstname VARCHAR(20) NOT NULL,

```
middlename VARCHAR (20),
lastname VARCHAR(20) NOT NULL,
age INT NOT NULL,
salary INT NOT NULL,
location VARCHAR(20) NOT NULL DEFAULT 'Banglore'
);
```

Now there is no primary key in the table suppose again u want to add PMKEY using ‘add’ cmd
alter table employee add primary key(id);

DDL vs DML
============
Data Definition Language —----- deals with table structure.
Ex: create, alter, drop etc.,

Data Manipulation Language —----- here we deal with the data directly.
Ex: insert, update, delete etc.,

TRUNCATE:
==========
Truncate also removes all records but it is a DDL command.
Truncate internally drops the table and recreates it.
truncate table employee;

FOREIGN KEY CONSTRAINT:
==========================

```
create table  students(
student_id int AUTO_INCREMENT,
student_fname VARCHAR(20) NOT NULL,
student_lname VARCHAR (20) NOT NULL,
student_mname VARCHAR(20),
student_email VARCHAR(30) NOT NULL,
student_phone VARCHAR(30) NOT NULL,
student_alternate_phone VARCHAR(15),
enrollment_date  TIMESTAMP NOT NULL default CURRENT_TIMESTAMP,
years_of_exp int NOT NULL,
student_company VARCHAR(20),
batch_date VARCHAR(30) NOT NULL,
source_of_joining VARCHAR(20) NOT NULL,
location varchar(30) NOT NULL ,
primary key (student_id),
unique key (student_email)
);
```

Whenever we do auto_increment compulsory we have to mention the primary key.
2nd if u dont want duplicate emails then u can mention unique key.
TIMESTAMP NOT NULL default CURRENT_TIMESTAMP, or
TIMESTAMP ON UPDATE CURRENT_TIMESTAMP and we can mention current timestamp alias as NOW()

Student seed data set-1
========================

```
Insert into students (student_fname, student_lname, student_email, student_phone, years_of_exp, student_company,
batch_date, source_of_joining, location)
values ('rohit', 'sharma', 'rohit@gamil.com', '9191919191', 6, 'walmart', '5-10-2022', 'linkedin', 'Bangalore');

Insert into students (student_fname, student_lname, student_email, student_phone, years_of_exp, student_company,
batch_date, source_of_joining, location)
values ('shan', 'kumar',  'shan@gamil.com', '9191949391', 3, 'amazon', '5-11-2022', 'linkedin', 'Bangalore');

Insert into students (student_fname, student_lname, student_email, student_phone, years_of_exp, student_company,
batch_date, source_of_joining, location)
values ('virat', 'kohli', 'virat@gamil.com', '9293929591', 5, 'walmart', '5-11-2022', 'linkedin', 'Hyderabad');

Insert into students (student_fname, student_lname, student_email, student_phone, years_of_exp, student_company,
batch_date, source_of_joining, location)
values ('rahul', 'kumar', 'rahul@gamil.com', '9092949191', 7, 'walmart', '11-12-2022', 'linkedin', 'Chennai');
('dhoni', 'singh', 'dhoni@gmail.com', '9092949191', 7, 'walmart', '11-12-2022', 'linkedin', 'pune'),
('brain', 'lara','brain@gamil.com', '9490919191', 5, 'TCS', '5-11-2022', 'youtube', 'Bangalore'),
('carl', 'hooper', 'carl@gamil.com', '9590979191', 8, 'TCS', '19-02-2022', 'google', 'Bangalore'),
('sachin', 'kumar', 'sachin@gamil.com', '9590899191', 20, 'Shell', '19-02-2022', 'google', 'pune');
```

Select statement
===============
Here the table is big and I am unable to see whole columns in a single screen so I am selecting only some columns.

```
select student_id, enrollment_date, student_fname, student_email, years_of_exp, student_company, batch_date,
source_of_joining, location from students;
```

Suppose if i start another course like devops, data science or any….
I have to create another course table like below

Course table
=====================

```
create table courses(
course_id int NOT NULL,
course_name varchar(20) NOT NULL,
course_duration_months int NOT NULL,
course_fee int NOT NULL,
PRIMARY KEY (course_id)
);
```

Seed data
============

```
insert into courses values (1, 'Big Data', 6, 50000);

insert into courses values (2, 'web Dev', 3, 20000);

insert into courses values (3, 'Data science', 6, 40000);

insert into courses values (4, 'DevOps', 3, 10000);

drop table students;
```

Bcoz we have to change slightly in table which course they are offering
Recreate table with changes.

```
create table  students(
student_id int AUTO_INCREMENT,
student_fname VARCHAR(20) NOT NULL,
student_lname VARCHAR (20) NOT NULL,
student_mname VARCHAR(20),
student_email VARCHAR(30) NOT NULL,
student_phone VARCHAR(30) NOT NULL,
student_alternate_phone VARCHAR(15),
enrollment_date  TIMESTAMP NOT NULL default CURRENT_TIMESTAMP,
selected_course int NOT NULL DEFAULT 1,
years_of_exp int NOT NULL,
student_company VARCHAR(20),
batch_date VARCHAR(30) NOT NULL,
source_of_joining VARCHAR(20) NOT NULL,
location varchar(30) NOT NULL ,
primary key (student_id),
unique key (student_email)
);
```

Seed data set-2
=================

```
Insert into students (student_fname, student_lname, student_email, student_phone, selected_course, years_of_exp,
student_company, batch_date, source_of_joining, location)
values ('rohit', 'sharma', 'rohit@gmail.com', '9191919191', 2, 6, 'walmart', '5-10-2022', 'linkedin', 'Bangalore');

Insert into students (student_fname, student_lname, student_email, student_phone, years_of_exp, student_company,
batch_date, source_of_joining, location)
values ('virat', 'kohli', 'virat@gmail.com', '9293929591', 5, 'walmart', '5-11-2022', 'linkedin', 'Hyderabad');

Insert into students (student_fname, student_lname, student_email, student_phone, years_of_exp, student_company,
batch_date, source_of_joining, location)
values ('shan', 'kumar',  'shan@gmail.com', '9191949391', 3, 'amazon', '5-11-2022', 'linkedin', 'Bangalore');

Insert into students (student_fname, student_lname, student_email, student_phone, selected_course, years_of_exp,
student_company, batch_date, source_of_joining, location)
values ('shami', 'sharma', 'shami@gmail.com', '9191919191', 3, 6, 'walmart', '5-10-2022', 'linkedin', 'Bangalore');

Insert into students (student_fname, student_lname, student_email, student_phone, years_of_exp, student_company,
batch_date, source_of_joining, location)
values ('rahul', 'kumar', 'rahul@gmail.com', '9092949191', 7, 'walmart', '11-12-2022', 'linkedin', 'Chennai'),
('dhoni', 'singh', 'dhoni@gmail.com', '9092949191', 7, 'walmart', '11-12-2022', 'linkedin', 'pune'),
('brain', 'lara','brain@gmail.com', '9490919191', 5, 'TCS', '5-11-2022', 'youtube', 'Bangalore'),
('carl', 'hooper', 'carl@gmail.com', '9590979191', 8, 'TCS', '19-02-2022', 'google', 'Bangalore'),
('sachin', 'kumar', 'sachin@gmail.com', '9590899191', 20, 'Shell', '19-02-2022', 'google', 'pune');
```

SELECT:

```
select student_id, enrollment_date, selected_course, student_fname, student_email, years_of_exp, student_company,
batch_date, source_of_joining, location from students;
```

NOTE:
example :
```
Insert into students (student_fname, student_lname, student_email, student_phone, selected_course, years_of_exp,
student_company, batch_date, source_of_joining, location)
values ('shami', 'sharma', 'shami@gmail.com', '9191919191', 5, 6, 'walmart', '5-10-2022', 'linkedin', 'Bangalore');
```

In the above record we enter course id 5 but there is no course id for 5 but when we execute the query, the System will record the id 5. It is inserted but it is not right. So how can we resolve the issue, so then the“Foreignkey” Comes into picture.

```
select * from courses;  (we have only 4 course available)

drop table students;
```

Now investigate what we have to do

```
create table  students(
student_id int AUTO_INCREMENT,
student_fname VARCHAR(20) NOT NULL,
student_lname VARCHAR (20) NOT NULL,
student_mname VARCHAR(20),
student_email VARCHAR(30) NOT NULL,
student_phone VARCHAR(30) NOT NULL,
student_alternate_phone VARCHAR(15),
enrollment_date  TIMESTAMP NOT NULL default CURRENT_TIMESTAMP,
selected_course int NOT NULL DEFAULT 1,
years_of_exp int NOT NULL,
student_company VARCHAR(20),
batch_date VARCHAR(30) NOT NULL,
source_of_joining VARCHAR(20) NOT NULL,
location varchar(30) NOT NULL ,
primary key (student_id),
unique key (student_email),
FOREIGN KEY (selected_course) REFERENCES courses(course_id)
);
```

Parent table —- courses (dependent)
Child table —- student

The foreign key constraint is used to prevent actions that would destroy links between two tables..
A foreign key is a field in one table that refers to the primary key in another table.

Selected_course is a foreign key in the students table which refers to course_id (primary key) in the course table.

The table with the foreign key is called the child table.
The table with the primary key is called the parent or referenced table.

NOT NULL —PRIMARY KEY —- UNIQUE KEY —--- FOREIGN KEY

What is constraint = constraints are used to limit the type of data that can go into a table.
This ensures the accuracy and reliability of the data is maintained.

If there is any violation then the action is aborted.
—---

DISTINCT
=========

select location from students; (this is ideal)

select DISTINCT location from students; (it doesn't duplicate values)

select DISTINCT student_company from students;

select distinct source_of_joining from students;

ORDER BY
=============

Order by is used to sort the changes from asc to desc order.

select student_fname from students order by student_fname;

select student_id, enrollment_date, selected_course, student_fname, student_email, years_of_exp, student_company, batch_date, source_of_joining, location from students order by years_of_exp;

select student_fname from students order by years_of_exp;
select student_fname,years_of_exp from students order by years_of_exp;

select student_fname,years_of_exp from students order by years_of_exp desc;

We can sort it out by column names using order by 1 or 2

select student_fname,years_of_exp from students order by 2 desc;
select student_fname,years_of_exp from students order by 1 desc;

select student_fname,years_of_exp from students order by years_of_exp,student_fname;

LIMIT
=======

select * from students limit 3;

Get 3 candidates with least exp;

select student_id, enrollment_date, selected_course, student_fname, student_email, years_of_exp, student_company, batch_date, source_of_joining, location from students order by years_of_exp limit 3;

Get 3 candidates with highest exp;

select student_id, enrollment_date, selected_course, student_fname, student_email, years_of_exp, student_company, batch_date, source_of_joining, location from students order by years_of_exp desc limit 3;

Want to know from which sources the last 5 candidates have enrolled.

select source_of_joining from students order by enrollment_date desc limit 5;

select student_fname from students order by enrollment_date desc limit 5;

And here I am getting o/p linkedin for 5 times and i don't want to know the same source of join so then i will use a distinct keyword.

–THE BELOW QUERY WON'T WORK
select DISTINCT source_of_joining from students order by enrollment_date desc limit 5;
WE WILL DISCUSS IN BELOW PARTS.

I want to know who is enrolled last.

select student_id, enrollment_date, selected_course, student_fname, student_email, years_of_exp, student_company, batch_date, source_of_joining, location from students order by enrollment_date desc limit 1;

select * from students order by enrollment_date limit 0,3;
select * from students order by enrollment_date limit 3,2;

LIKE
======
select student_id, enrollment_date, selected_course, student_fname, student_email, years_of_exp, student_company, batch_date, source_of_joining, location from students where student_fname LIKE '%ra%';

% is a wildcard character.
_ (underscore gives u exactly how many characters in the table that u mention in the query). ex : '__' or' ___'or'_____'.
select student_id, enrollment_date, selected_course, student_fname, student_email, years_of_exp, student_company, batch_date, source_of_joining, location from students where student_fname LIKE 'ra%';

select student_id, enrollment_date, selected_course, student_fname, student_email, years_of_exp, student_company, batch_date, source_of_joining, location from students where student_fname LIKE '%at';

select student_id, enrollment_date, selected_course, student_fname, student_email, years_of_exp, student_company, batch_date, source_of_joining, location from students where student_fname LIKE '_____';

select student_id, enrollment_date, selected_course, student_fname, student_email, years_of_exp, student_company, batch_date, source_of_joining, location from students where student_fname LIKE '____';

—---
–THE BELOW QUERY WON'T WORK
select DISTINCT source_of_joining from students order by enrollment_date desc limit 5;
WE WILL DISCUSS IN BELOW PARTS.
WHY
=====
ORDER OF EXECUTION
====================
FROM (LOADING THE TABLE)
select * from students;

SELECT (PROJECTING source_of_joining,enrollment_date)
select source_of_joining,enrollment_date from students;

Order by (based on enrollment_date it will order by)
select source_of_joining,enrollment_date from students order by enrollment_date;

Even though we don't have an enrollment date in select stmt we can do order by.
select source_of_joining from students order by enrollment_date;

select DISTINCT source_of_joining from students order by enrollment_date;

ORDER OF EXECUTION
=====================
FROM (LOADING THE TABLE)
select * from students;

SELECT (PROJECTING source_of_joining,enrollment_date)
select source_of_joining,enrollment_date from students;

DISTINCT
select DISTINCT source_of_joining from students;
select DISTINCT source_of_joining,enrollment_date from students;

Order by (based on enrollment_date it will do order by)
select source_of_joining,enrollment_date from students order by enrollment_date;

So here the order of execution is imp so that the distinct and order by wont work here in this case.
—---

AGGREGATE FUNCTIONS
==========================
COUNT()
----
select count(*) from students; (it tells how many rows are there)
select count(student_company) from students; (it returns how many company peoples are joining)
select count(distinct student_company) from students;
select count(distinct location) from students;   (from which location)
select count(distinct source_of_joining) from students;
select count(*) from students where batch_date like '%-02-%';
select count(*) from students where batch_date like 19-%';

GROUP BY

What i want to know that how many people have joined my course got to know about me through

select source_of_joining ,count(*) from students GROUP BY source_of_joining;

How many people are joined location wise may b same location or diff

select location, count(*) from students group by location;

–the below query doesn't work becoz u have to mention item in group by as well as in select stmt:
select location, count(*) from students group by source_of_joining;

select location, source_of_joining, count(*) from students group by location, source_of_joining;

select selected_course,count(*) from students group by selected_course;

select selected_course,batch_date,count(*) from students group by selected_course,batch-date;

MIN() & MAX()

----

select min(years_of_exp) from students;
select max(years_of_exp) from students;

select min(years_of_exp,student_name) from students; (this won't work)

select student_fname from students order by years_of_exp limit 2;
select student_fname from students order by years_of_exp limit 1;

–each source of joining i want to get max exp
Select source_of_joining, max(years_of_exp) from students group by source_of_joining;

SUM()

Here we combined sum of exp who are joined through source of joining using sum aggregate fun.

Select source_of_joining, SUM(years_of_exp) from students group by source_of_joining;

AVG()

Select source_of_joining, AVG(years_of_exp) from students group by source_of_joining;
Select location, AVG(years_of_exp) from students group by location;
Select student_company, AVG(years_of_exp) from students group by student_company;
—--

DECIMAL

===========

```
create table courses_new(
course_id int NOT NULL,
course_name varchar(20) NOT NULL,
course_duration_months decimal(3,1) NOT NULL,
Course_fee int NOT NULL,
PRIMARY KEY(course_id)
);
```

insert into courses_new values (1, 'Big Data', 6.5, 50000);

insert into courses_new values (2, 'web Dev', 3.5, 20000);

insert into courses_new values (3, 'Data science', 6, 40000);

Want to update the course fee?
update courses_new set course_fee = 40000 where course_id = 2;

History of when we make changes to courses name or fees or id so for that we need timestamp to know at
What time we make changes in the table.

TIMESTAMP

============

create table courses_new(
course_id int NOT NULL,
course_name varchar(20) NOT NULL,
course_duration_months decimal(3,1) NOT NULL,
course_fee int NOT NULL,
changed_at TIMESTAMP DEFAULT NOW(),
PRIMARY KEY(course_id)
);

insert into courses_new(course_id, course_name, course_duration_months, course_fee) values (1, 'Big Data', 6.5, 50000);

insert into courses_new (course_id, course_name, course_duration_months, course_fee) values (2, 'web Dev', 3.5, 20000);

insert into courses_new (course_id, course_name, course_duration_months, course_fee) values (3, 'Data science', 6, 40000);

insert into courses_new (course_id, course_name, course_duration_months, course_fee) values (4, 'DevOps', 10.5, 30000);

While changing the course fee u r getting the same time while the course is created but not time so here we r making a small change in definition of table

create table courses_new(
course_id int NOT NULL,
course_name varchar(20) NOT NULL,
course_duration_months decimal(3,1) NOT NULL,
course_fee int NOT NULL,
changed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP() ON UPDATE CURRENT_TIMESTAMP(),
PRIMARY KEY(course_id)
);

HERE WE CAN USE EITHER
NOW () OR CURRENT TIMESTAMP()

select * from courses_new;
update courses_new set course_fee = 10000 where course_id = 1;
—---

LOGICAL OPERATORS

=====================

Select * from students where location = ‘Bangalore’;

//to get people who are not from bangalore//
 select * from students where location != 'Bangalore';

//get all courses which has the word ‘data’
select * from courses where course_name like '%data%';

//get all courses which has the word not ‘data’
select * from courses where course_name not like '%data%';

//all students from bangalore who join through linkedin less than 8 years of exp.
Select * from students where years_of_exp < 8 and source_of_joining = 'linkedin' and location = 'Bangalore';

// I want all people who do not fall between 8 to 12 years of exp.
select * from students where years_of_exp < 8 or years_of_exp > 12;
select * from students where years_of_exp between 8 and 12;
select * from students where years_of_exp not between 8 or 12;

//get list of students who are working from flipkart,walmart,TCS
select * from students where student_company = 'flipkart' or student_company = 'walmart' or student_company = 'TCS';

select * from students where student_company IN (flipkart, walmart. tcs);
select * from students where student_company not IN ('flipkart', 'walmart', 'tcs');

//if a course is more than 4 months we categorize it as a master program else it is a pg diploma.
```
select course_id,course_name,course_fee,
Case
when course_duration_months > 4 then 'Masters'
else 'PG Diploma'
end as course_type
from courses;
```

// people working for walmart,google we want to say product based and all others service based.
```
Select student_id, student_fname, student_lname, student_company,
Case
When student_company in ('walmart','google','flipkart') then 'product based'
else 'service based'
End as company_type
from students;
```

Here we can define any number of conditions in a table using logical operators.
In, not in, like, not like, between, not between, >, < , !=, or.

JOINS
========

We have 2 tables
1] courses
2] students

//I want to know in which course Rahul has enrolled.
select student_fname, selected_course from students;

From the above query we only get student name and selected course id but we don't get any course name along with it bcoz it is in another table. So we can we do that

Students
=========
Student_fname, selected_course
Rahul, 1

Courses
==========
Course_id, course_name
1, Big data

Select course_name from courses where course_id = (select selected_course from students where student_fname = 'rahul');

The above approach is one way but it is not the righteous way if we have multiple columns. So here we are Using joins

So in that join we have to join one similar column from both tables.

In students table selected_course
In courses table course_id
select students.student_fname, students.student_lname, courses.course_name from students join courses on students.selected_course = courses.course_id;

//by default it is above the inner join.
Only matching records are considered. Non matching records are discarded.

Left outer join
=============
//all the matching records from left and right table are considered
+
all the non matching records in the left table which does not have the match in the right padded with NULL.

select students_latest.student_fname, students_latest.student_lname, courses_latest.course_name from students_latest join courses_latest on students_latest.selected_course = courses_latest.course_id;

select students_latest.student_fname, students_latest.student_lname, courses_latest.course_name from students_latest left join courses_latest on students_latest.selected_course = courses_latest.course_id;

Right outer join
=============
//all the matching records from left and right table are considered
+
all the non matching records in the right table which does not have the match in the left padded with NULL.

select students_latest.student_fname, students_latest.student_lname, courses_latest.course_name from students_latest right join courses_latest on students_latest.selected_course = courses_latest.course_id;

Full outer join
=============

all matching records
 +
non matching records from left
 +
non matching records from right

select students_latest.student_fname, students_latest.student_lname, courses_latest.course_name from students_latest left join courses_latest on students_latest.selected_course = courses_latest.course_id
union
select students_latest.student_fname, students_latest.student_lname, courses_latest.course_name from students_latest right join courses_latest on students_latest.selected_course = courses_latest.course_id;

Inner
Left outer
Right outer
Full outer join there is no keyword for the full outer so any database directly we can use
With the UNION keyword to combine both the right and left table.
Cross join

Diff b/w WHERE and HAVING clause in sql
=====================================
Select source_of_joining, count(*) as total from students group by source_of_joining;
// I want to know the lead sources through which more than 1 person has registered.

Select source_of_joining, count(*) as total from students group by source_of_joining where total > 1;
The above stmt doesn't work due to the reason below.
Where clause is used to filter the individual records before aggregation…(before group by).

Select source_of_joining, count(*) as total from students group by source_of_joining HAVING total > 1;

// I want to know the count of people who registered through linkedin.

Select source_of_joining, count(*) as total from students group by source_of_joining HAVING source_of_joining = 'linkedin';
or
Select source_of_joining, count(*) as total from students where source_of_joining = 'linkedin' group by source_of_joining;

Can we use where and Having in same query
==
// from which locations more than 1 student has joined & the students exp is more than 5 years of exp.

select location, count(*) as total from students where years_of_exp > 5 group by location having total > 1;

Where is used before group by and do filtering on individual records.
Having is used after group by and do filtering on aggregated records.

We can use ‘where’ and ‘having’ in the same query also.

Where is more performant than having…
—--

Over clause and Partition BY
==========================

```
CREATE TABLE EMPLOYEE (
firstname varchar(20),
lastname varchar(20),
Age int,
Salary int,
Location varchar(20)
);

INSERT INTO EMPLOYEE VALUES ('sachin', 'kumar', 28, 10000, 'bangalore');
INSERT INTO EMPLOYEE VALUES ('shane', 'warne', 30, 20000, 'bangalore');
INSERT INTO EMPLOYEE VALUES ('rohit', 'sharma', 32, 30000, 'hyderabad');
INSERT INTO EMPLOYEE VALUES ('shikar', 'dawan', 32, 25000, 'hyderabad');
INSERT INTO EMPLOYEE VALUES ('rahul', 'kumar', 31, 20000, 'bangalore');
INSERT INTO EMPLOYEE VALUES ('saurav', 'kumar', 32, 10000, 'bangalore');
INSERT INTO EMPLOYEE VALUES ('kapil', 'sharma', 31, 10000, 'pune');
// I want to know how many people from each location and avg salary at each location

select location, count(location)as total,avg(salary) as average from employee group by location;
```

```
location        count(location)                 avg(salary)
Bangalore            4                            15000.0000
hyderabad            2                            27500.0000
pune                 1                            10000.0000
```

```
select firstname, lastname, location, count(location),avg(salary) from employee group by location;
```

When we run the above query we can get an error so we can achieve it using a join…

```
select firstname, lastname, EMPLOYEE.location, total_count, avg_salary from EMPLOYEE join (select
location, count(location) as total_count ,avg(salary) as avg_salary from EMPLOYEE group by location)
temptable on EMPLOYEE.location = temptable.location;
```

// we can use OVER PARTITION BY to achieve this easily.

```
select firstname, lastname, location,
count(location) over(partition by location) as total,
avg(salary) over(partition by location) as average from EMPLOYEE;
```

—---

ROW NUMBER
================

We are working on the same table as above.

```
select firstname, lastname, salary,
row_number() over(order by salary desc) as highest from EMPLOYEE;
```

Output:

```
firstname  lastname       salary  highest
rohit      sharma         30000     1
shikar     dawan          25000     2
shane      warne          20000     3
rahul      kumar          20000     4
sachin     kumar          10000     5
saurav     kumar          10000     6
kapil      sharma         10000     7
```

//From the above o/p we get the highest salary in descending order so we can decide who is getting the highest salary as numbered 1 and 2nd highest salary as 2 and same as all etc.,........//

–find the 5th highest salary
There are many approaches to do but row number is the simplest way.

```
select * from (select firstname, lastname, salary,
row_number() over(order by salary desc) as rownum from EMPLOYEE) temptable where rownum =5;
```

Output:

```
firstname  lastname       salary    rownum
sachin     kumar          10000        5
```

// the problem statement is to assign row numbers for partitions based on each location..

```
select firstname, lastname, location, salary,
row_number() over(partition by location order by salary desc) as clubbed from EMPLOYEE;
```

//I want to find the highest salary getters at each location….

```
select * from (select firstname, lastname, location, salary,
row_number() over(partition by location order by salary desc) as rownum from EMPLOYEE) temptable
where rownum = 1;
```

When we use row_number
We should be using the order by clause if not it won't work
We can also use the partition by but it is –optional–
The row number starts from 1 for every partition.

RANK & DENSE RANK
=====================

From the same table above

```
select firstname, lastname, salary,
row_number() over(order by salary desc) as highest from EMPLOYEE;
```

```
select firstname, lastname, salary,
RANK() over(order by salary desc) as highest from EMPLOYEE;

select firstname, lastname, salary,
DENSE_RANK() over(order by salary desc) as highest from EMPLOYEE;
```

Using row number()

Output:

```
firstname  lastname     salary  highest
rohit      sharma       30000     1
shikar     dawan        25000     2
shane      warne        20000     3     [here the salary for shane & rahul same but the given highest shows as
rahul      kumar        20000     4      shane but rahul as same so that rownum doesn't handle duplicate values
sachin     kumar        10000     5      properly].
saurav     kumar        10000     6
kapil      sharma       10000     7
```

Using RANK()

```
firstname  lastname     salary  highest
rohit      sharma       30000     1
shikar     dawan        25000     2
shane      warne        20000     3
rahul      kumar        20000     3
sachin     kumar        10000     5
saurav     kumar        10000     5
kapil      sharma       10000     5
```

USING DENSE_RANK()

```
firstname  lastname     salary  highest
rohit      sharma       30000      1
shikar     dawan        25000      2
shane      warne        20000      3
rahul      kumar        20000      3
sachin     kumar        10000      4
saurav     kumar        10000      4
kapil      sharma       10000      4
```

If there are no duplicates the row number, rank and dense_rank lead to similar results…

Only the difference comes when there are duplicates…

Rank - for duplicates the same rank is assigned and for the next entry it skips the ranks.

Dense-rank – it doesn't not skip any ranks in between..

```
select * from (select firstname, lastname, salary,
row_number() over(order by salary desc) as rownum from EMPLOYEE) temptable where rownum =5;
```

```
select * from (select firstname, lastname, salary,
rank() over(order by salary desc) as rownum from EMPLOYEE) temptable where rownum =5;
```

```
firstname  lastname       salary  rownum
sachin     kumar          10000     5
kapil      sharma         10000     5
saurav     kumar          10000     5
```

```
select * from (select firstname, lastname, salary,
dense_rank() over(order by salary desc) as rownum from EMPLOYEE) temptable where rownum =4;
```

```
firstname  lastname       salary  rownum
sachin     kumar          10000     4
saurav     kumar          10000     4
kapil      sharma         10000     4
```

======================================End===