SQL NOTES

ORDER OF QUERY EXECUTION
FROM → JOIN → WHERE → GROUP BY → HAVING → SELECT → DISTINCT → ORDER BY

DDL Commands for Databases

1 CREATE -- CREATE DATABASE IF NOT EXISTS singh;
2. DROP-- DROP DATABASE IF EXISTS singh;

DDL Commands for Tabels

1 Create -- CREATE TABLE singh.users(
user_id INTEGER,
name VARCHAR(255),
email VARCHAR (255),
password VARCHAR(255));

Example :- INSERT INTO singh.users (user_id,name,email,password)
VALUES ('1','Gourab','gaurav@gmail.com','1234');

2 Truncate -- TRUNCATE TABLE singh.users;
3 Drop -- DROP TABLE IF EXISTS singh.users;

CONSTRAINTS IN MY SQL

1. **NOT NULL**

CREATE TABLE singh.users(
user_id INTEGER NOT NULL,
name VARCHAR(255) NOT NULL,

```
email VARCHAR (255) ,
password VARCHAR(255));
```

2. **<u>UNIQUE</u>**

```
CREATE TABLE singh.users(
user_id INTEGER NOT NULL,
name VARCHAR(255) NOT NULL,
email VARCHAR (255) NOT NULL UNIQUE,
password VARCHAR(255) NOT NULL);
```

ANOTHER WAY TO CREATING CONSTRAINTS

```
CREATE TABLE singh.users (
   user_id INTEGER NOT NULL,
   name VARCHAR(255) NOT NULL,
   email VARCHAR(255) NOT NULL,
   password VARCHAR(255) NOT NULL,
   CONSTRAINT users_email_unique UNIQUE (name,
email, password)
);
```

CONSTRAINT users_email_unique UNIQUE (name, email, password)

- constraint niche likhne ka faayda ye hai ki hum kisi do ya teen ke combination par unique constrain laga sakte hai jabki ye upar wale syntax se possible nii hota aur dusra faayda ye hai hai baad me hum constraint change kar sakte hain aur hamara table v bhi affected nahi hoga upar wale syntax me change karne ke liye hame usko delete karna parta jisme ki data loss ki possibility jyada hoti hai.

3. **<u>PRIMARY KEY</u>**

```
CREATE TABLE singh.users (
   user_id INTEGER NOT NULL,
   name VARCHAR(255) NOT NULL,
   email VARCHAR(255) NOT NULL,
   password VARCHAR(255) NOT NULL,
   CONSTRAINT users_email_unique UNIQUE (name,
email, password),
```

```
    CONSTRAINT users_pk PRIMARY KEY (user_id,name)
);
```

4. AUTO_INCREMENT

```
CREATE TABLE singh.users (
    user_id INTEGER PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL
);
```

5. CHECK

```
CREATE TABLE singh.students(
        student_id INTEGER PRIMARY KEY
AUTO_INCREMENT,
        name VARCHAR(50) NOT NULL,
        age INTEGER,
        CONSTRAINT students_age_check CHECK (age > 18
AND age< 60)
);
```

6. DEFAULT

```
CREATE TABLE singh.ticket(
ticket_id INTEGER PRIMARY KEY,
name VARCHAR(255) NOT NULL,
travel_date DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

7. FOREIGN KEY

```
CREATE TABLE singh.customers(
        cid INTEGER PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR (255) NOT NULL,
    email VARCHAR (255) NOT NULL UNIQUE
);
```

```
CREATE TABLE singh.orders(
   order_id INTEGER PRIMARY KEY AUTO_INCREMENT,
   cid INTEGER NOT NULL,
   order_date DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP,

   CONSTRAINT orders_fk FOREIGN KEY (cid)
REFERENCES customers(cid)
   );
```

REFRENCIAL ACTIONS

1 RESTRICT

2 CASCADE

3 SET NULL

4 SET DEFAULT

ALTER TABLE COMMAND

1. **ADD COLUMNS**

```
ALTER TABLE singh.customers ADD COLUMN address
VARCHAR (255) NOT NULL;
```

For on the position :-

```
ALTER TABLE singh.customers ADD COLUMN
pin_code VARCHAR (255) NOT NULL after address;
```

ADD MULTIPLE COLUMNS AT A TIME

```
ALTER TABLE singh.customers
ADD COLUMN pan_number VARCHAR (255) after name,
ADD COLUMN joining_date DATETIME NOT NULL
DEFAULT CURRENT_TIMESTAMP
;
```

2. **<u>DELETE COLUMNS</u>**

```
ALTER TABLE singh.customers
DROP COLUMN pan_number;
```

```
ALTER TABLE singh.customers
DROP COLUMN password,
DROP COLUMN joining_date;
```

3. **MODIFY COLUMNS**

```
ALTER TABLE singh.customers
  MODIFY COLUMN name INTEGER;
```

EDITING AND DELETING CONSTRAINTS

1. **ADD**

```
ALTER TABLE singh.customers ADD CONSTRAINT
customer_age_check CHECK (age>13);
```

2. **DELETE**

```
ALTER TABLE singh.customers DROP CONSTRAINT
customer_age_check ;
```

3. **EDIT** –

edit nahi hota hai aapko pahle drop karna parega uske baad phir se create karna hoga.

INSERT

```
CREATE DATABASE IF NOT EXISTS gourav;

CREATE TABLE IF NOT EXISTS gourav.users(
        user_id INTEGER PRIMARY KEY
AUTO_INCREMENT,
   name VARCHAR(255) NOT NULL,
   email VARCHAR(255) NOT NULL UNIQUE,
   password VARCHAR(255) NOT NULL
);
```

```
INSERT INTO gourav.users (user_id,name,email,password)
VALUES (NULL, 'Gourav', 'gourav@gmail.com','1234')

INSERT INTO gourav.users
VALUES (NULL, 'radhe', 'radhe@gmail.com','1234')

INSERT INTO gourav.users (name,password,email)
VALUES ('amit', '12345','amit@gmail.com')
```

Insert Multiple

```
INSERT INTO gourav.users VALUES
(NULL, 'Shubham', 'Shubham@gmail.com', '123'),
(NULL, 'Surbhi', 'Surbhi@gmail.com', '12345');
```

Select All

```
SELECT * FROM gourav.smartphones WHERE 1;
SELECT * FROM gourav.smartphones;
```

Filter columns

```
SELECT model,price, rating FROM
gourav.smartphones;
SELECT model, battery_capacity, os FROM
gourav.smartphones;
```

Aliasing → Renaming Columns

```
SELECT model, battery_capacity as 'mAh', os as 'operating
system' FROM gourav.smartphones;
```

Creating Expression using cols

```
SELECT model,
sqrt(resolution_width*resolution_width +
resolution_height*resolution_height)/screen_size as 'ppi'
FROM gourav.smartphones;

SELECT model,rating/10 FROM gourav.smartphones;
```

CONSTANTS

```
SELECT model, 'smartphones' AS 'type' FROM
gourav.smartphones;
```

Distinct (Unique) Values from a col

```
SELECT DISTINCT(brand_name) AS 'ALL Brands'
FROM gourav.smartphones;
```

Distinct Combos

```
SELECT DISTINCT brand_name, processor_brand
FROM gourav.smartphones;
```

Filter rows Where clause

1. **Find all Samsung phones**

```
select * from gourav.smartphones
where brand_name= 'samsung'
```

2. **Find all phones with Price > 50000**

```
select * from gourav.smartphones
where price > 50000
```

BETWEEN

Find all phones in the price range of 10000 and 20000

```
select * from gourav.smartphones
where price > 10000 and price < 20000

select * from gourav.smartphones
where price BETWEEN 10000 and  20000
```

3. **Find phones with rating > 80 and price < 25000**

```
select * from gourav.smartphones
where rating > 80 and price < 25000

select * from gourav.smartphones
where rating > 80 and price < 25000 and processor_brand =
'snapdragon'
```

4. **find all Samsung phones with ram > 8GB**

```
select * from gourav.smartphones
where brand_name = 'samsung' and ram_capacity > 8
```

5. **find all Samsung phones with snapdragon processor**

```
select * from gourav.smartphones
where brand_name = 'samsung' and processor_brand =
'snapdragon'
```

6. **Find brands who sell phones with price > 50000**

```
SELECT DISTINCT brand_name FROM
gourav.smartphones
WHERE price > 50000;
```

IN AND NOT IN

```
SELECT * FROM gourav.smartphones
where processor_brand IN ('snapdragon', 'exynos','bionic')

SELECT * FROM gourav.smartphones
where processor_brand NOT IN ('snapdragon',
'exynos','bionic')
```

UPDATE

```
UPDATE gourav.smartphones
SET processor_brand = 'dimensity'
WHERE processor_brand = 'mediatek'
```

DELETE

Delete all phones price > 200000
DELETE FROM gourav.smartphones
where price > 200000

DELETE FROM gourav.smartphones
WHERE primary_camera_reared > 150 AND brand_name = ‘samsung’

MAX/MIN

1. **Find the minimum and maximum price**

SELECT MAX(price) from gourav.smartphones;
SELECT MIN(price) from gourav.smartphones;
SELECT MIN(ram_capacity) from gourav.smartphones;
SELECT MAX(ram_capacity) from gourav.smartphones;

2. **Find the price of the costliest Samsung phone**

SELECT MAX(price) from gourav.smartphones
where brand_name = 'samsung';

AVG

- **find avg rating of apple phone**

SELECT AVG(rating) from gourav.smartphones
where brand_name = 'apple';

SUM

SELECT SUM(price) from gourav.smartphones;

COUNT

- **Find the number of oneplus phones**

SELECT COUNT(*) from gourav.smartphones

```
WHERE brand_name = 'oneplus';
```

COUNT(DISTINCT)

- **Find the number of brands available**

```
SELECT COUNT(DISTINCT(brand_name)) from
gourav.smartphones;
```

STD

- **find the std of screen size**

```
SELECT STD(screen_size) from gourav.smartphones;
```

VARIANCE

- **Find the variance of Xiomi phone price**

```
SELECT VARIANCE(price) from gourav.smartphones
where brand_name = 'xiaomi';

SELECT VARIANCE(screen_size) from
gourav.smartphones;
```

SCALAR FUNCTION

ABS

```
SELECT ABS(price-100000) AS 'temp' from
gourav.smartphones;
```

ROUND

```
SELECT model,
ROUND(sqrt(resolution_width*resolution_width +
resolution_height*resolution_height)/screen_size) as 'ppi'
FROM gourav.smartphones;
```

- **ROUND THE PPI FOR TWO DECIMAL PLACES**

```
SELECT model,
ROUND(sqrt(resolution_width*resolution_width +
resolution_height*resolution_height)/screen_size,2) as 'ppi'
FROM gourav.smartphones;
```

FOR CEIL/FLOOR

```
SELECT CEIL (Screen_size) from gourav.smartphones;
SELECT FLOOR (Screen_size) from gourav.smartphones;
```

TASK

```
USE sql_tasks;
SELECT * FROM insurance;
```

1. Show records of 'male' patient from 'southwest' region.

```
SELECT * FROM insurance
WHERE gender = 'male' AND region = 'southwest';
```

2. Show all records having bmi in range 30 to 45 both inclusive.

```
SELECT * FROM insurance
WHERE bmi BETWEEN 30 AND 45;
```

3. Show minimum and maximum bloodpressure of diabetic patient who smokes. Make column names as MinBP and MaxBP respectively.

```
SELECT MIN(bloodpressure) AS 'MinBP',
MAX(bloodpressure) AS 'MaxBP'
FROM insurance
WHERE diabetic = 'Yes' AND smoker = 'Yes';
```

4. Find no of unique patients who are not from southwest region.

```
SELECT COUNT(DISTINCT(PatientID)) FROM insurance
WHERE region <> 'southwest';
```

5. Total claim amount from male smoker.

```
SELECT SUM(claim)
FROM insurance
WHERE gender = 'male';
```

6. Select all records of south region.

```
SELECT * FROM insurance
WHERE region LIKE 'south%';
```

7. No of patient having normal blood pressure. Normal range[90-120]

```
SELECT COUNT(*) FROM insurance
WHERE bloodpressure BETWEEN 90 AND 120;
```

8. No of patient below 17 years of age having normal blood pressure as per below formula -BP normal range = 80+(age in years × 2) to 100 + (age in years × 2)
Note: Formula taken just for practice, don't take in real sense.

```
SELECT COUNT(*) FROM insurance
WHERE age < 17
AND (bloodpressure BETWEEN 80+(age * 2) AND 100 +
(age * 2));
```

9. What is the average claim amount for non-smoking female patients who are diabetic?

```
SELECT AVG(claim) FROM insurance
WHERE gender = 'female'
AND smoker = 'No';
```

10. Write a SQL query to update the claim amount for the patient with PatientID = 1234 to 5000.

```
UPDATE insurance SET claim = 5000
WHERE PatientID = 1234;

SELECT * FROM insurance WHERE PatientID = 1234;
```

11. Write a SQL query to delete all records for patients who are smokers and have no children.

```
DELETE FROM insurance
WHERE smoker = 'Yes' AND children = 0
```

Task Completed

Sorting Data

1) **find top 5 samsung phones with biggest screen size**

```
SELECT model,screen_size FROM gourav.smartphones
WHERE brand_name = 'samsung' ORDER BY screen_size
DESC LIMIT 5;
```

2) **Sort all the phone in descending order of number of total cameras**

```
SELECT model, num_front_cameras + num_rear_cameras
AS 'total_cameras'
FROM gourav.smartphones
ORDER BY total_cameras DESC;
```

3) **Sort data on the basis of ppi in decreasing order.**

```
SELECT model,
ROUND(sqrt(resolution_width*resolution_width +
resolution_height*resolution_height)/screen_size,2) as 'ppi'
FROM gourav.smartphones
ORDER BY ppi DESC;
```

4) **Find the phone with 2nd largest battery**

```
SELECT model, battery_capacity
FROM gourav.smartphones
ORDER BY battery_capacity DESC LIMIT 1,1;
```

5) **Find the name and rating of the worst rated apple phone**

```
SELECT model, rating
FROM gourav.smartphones
WHERE brand_name = 'apple'
ORDER BY rating ASC limit 1 ;
```

6) **Sort phones alphabetically and then on the basis of rating in desc order**

```
SELECT * FROM gourav.smartphones
ORDER BY brand_name ASC, rating DESC;
```

7) **Sort phones alphabetically and then on the basis of price in asc order;**

```
SELECT * FROM gourav.smartphones
ORDER BY brand_name ASC, price ASC;
```

8) **Find the phone name ,price of the costliest phone**

```
SELECT model,price
FROM gourav.smartphones
ORDER BY price DESC LIMIT 1;
```

GROUPING DATA

1. **Group smartphones by brand and get the count, average price, max rating, avg screen size and avg battery capacity**

```
SELECT brand_name, COUNT(*) AS 'num_phones',
ROUND (AVG(price)) as 'avg_price',
MAX(rating) AS 'Max_Rating',
ROUND(AVG(screen_size),2) AS 'avg_screen_size',
ROUND(AVG(battery_capacity)) AS 'avg_battery_capacity'
FROM gourav.smartphones
GROUP BY brand_name
ORDER BY num_phones DESC LIMIT 15;
```

2. **Group smartphones by whether they have an NFC and get the average price and rating**

```
SELECT has_nfc,
AVG(price) AS 'avg_price',
AVG(rating) AS 'avg_rating'
FROM gourav.smartphones
GROUP BY has_nfc
```

3. **Group smartphones by the extended memory available and get the average price**

```
SELECT extended_memory_available,
AVG(price) AS 'avg_price'
FROM gourav.smartphones
GROUP BY extended_memory_available;
```

4. **Group smartphones by the brand and processor brand and get the count of models and average primary camera (rear) resolution**

```
select brand_name,
processor_brand,
COUNT(*) AS 'num_phones',
ROUND(AVG(primary_camera_rear)) AS
'avg_camera_resolution'
from gourav.smartphones
GROUP BY brand_name,processor_brand;
```

5. **Find top 5 most costly phone brands**

```
select brand_name, ROUND(AVG(price)) as 'avg_price'
from gourav.smartphones
GROUP BY brand_name
order by avg_price DESC LIMIT 5;
```

6. **Which brands make the smallest screen smartphones**

```
select brand_name, ROUND(AVG(screen_size)) as
'avg_screen_size'
from gourav.smartphones
GROUP BY brand_name
order by avg_screen_size ASC LIMIT 1;
```

7. **Avg price of 5g phones vs avg price of non 5g phones**

```
SELECT has_5g,
AVG(price) AS 'avg_price',
AVG(rating) AS 'avg_rating'
FROM gourav.smartphones
GROUP BY has_5g
```

8. **Group smartphones by the brand , and find the brand with the highest number of models that have both NFC and IR blaster**

```
SELECT brand_name, COUNT(*) AS 'count'
FROM gourav.smartphones
WHERE has_nfc = 'True' AND has_ir_blaster = 'True'
GROUP BY brand_name
ORDER BY count DESC LIMIT 1;
```

9. **Find all Samsung 5g enabled smartphones and find out the average price for nfc and non-nfc phones**

```
SELECT has_nfc, AVG(price) AS 'avg_price'
FROM gourav.smartphones
WHERE brand_name = 'samsung'
GROUP BY has_nfc;
```

HAVING CLAUSE

I. **Find the average rating of smartphone brands which have more then 20 phones.**

```
SELECT brand_name,
COUNT(*) AS 'counts',
AVG(rating) AS 'avg_rating'
FROM gourav.smartphones
GROUP BY brand_name
HAVING counts >= 20
ORDER BY avg_rating DESC;
```

II. **Find the top 3 brands with the highest avg ram that have a refresh rate of at least 90 hz and fast charging available and don't consider brands which have less than 10 phones**

```
SELECT brand_name,
AVG(ram_capacity) AS 'avg_ram'
FROM gourav.smartphones
WHERE refresh_rate > 90 AND fast_charging_available = 1
GROUP BY brand_name
HAVING COUNT(*) > 10
ORDER BY 'avg_ram' DESC LIMIT 3;
```

III. **Find the average price of all the phone brands with avg rating > 70 and num_phones more than 10 among all 5g enabled phones.**

```
SELECT brand_name, AVG(price) AS 'avg_price'
FROM gourav.smartphones
WHERE has_5g = 'True'
GROUP BY brand_name
HAVING AVG(rating) > 70 AND COUNT(*) > 10;
```

<u>Practice</u>

i. **Find the top 5 batsman in IPL**

```
SELECT batter, SUM(batsman_run) AS 'runs'
FROM gourav.ipl
GROUP BY batter
ORDER BY runs DESC LIMIT 5;
```

ii. **Find the 2nd highest 6 hitter in IPL**

```
SELECT batter, COUNT(*) AS 'num_sixes'
FROM gourav.ipl
WHERE batsman_run = 6
GROUP BY batter
ORDER BY num_sixes DESC LIMIT 1,1;
```

iii. **Find Batsman With Centuries In IPL**

```
SELECT batter,ID,SUM(batsman_run) AS 'score' FROM
gourav.ipl
GROUP BY batter,ID
HAVING score >= 100
ORDER BY batter DESC
```

iv. **Find the top 5 batsman with highest strike rate who have played a min of 1000 balls.**

```
SELECT batter, SUM(batsman_run), COUNT(batsman_run),
ROUND ((SUM(batsman_run)/COUNT(batsman_run)) *
100,2) AS 'Sr'
FROM gourav.ipl
GROUP BY batter
HAVING COUNT(batsman_run) > 1000
ORDER BY sr DESC LIMIT 5;
```

TASK

USE sql_tasks;

1) **Q1: Find out the average sleep duration of top 15 male candidates who's sleep duration are equal to 7.5 or greater than 7.5**

```
SELECT AVG(`Sleep duration`) FROM (
SELECT * FROM task33.sleepefficiency WHERE `Sleep
duration` >= 7.5 AND Gender= 'male' ORDER BY `Sleep
duration` DESC LIMIT 15
) AS sleeps
```

2) **Show avg deep sleep time for both gender.Round result at 2 decimal places.**
Note - sleep time and deep sleep percentage will give you, deep sleep time

```
SELECT Gender,AVG(`Sleep duration`*(`Deep sleep
percentage`/100)) AS 'avg_deep_sleep'
FROM sleep
GROUP BY Gender;
```

3) **Find out the lowest 10th to 30th light sleep percentage records where deep sleep percentage values are between 25 to 45.**

Display age, light sleep percentage and deep sleep percentage columns only.

```
SELECT Age,`Light sleep percentage`,`Deep sleep
percentage` FROM sleep
WHERE `Deep sleep percentage` BETWEEN 25 AND 45
ORDER BY `Light sleep percentage` LIMIT 10,20;
```

4) **Group by on exercise frequency and smoking status and show average deep sleep time, average light sleep time and avg rem sleep time.**
Note - Note the differences in deep sleep time for smoking and non smoking status

```
SELECT `Exercise frequency`,`Smoking status`,
AVG(`Sleep duration`*(`Deep sleep percentage`/100)),
AVG(`Sleep duration`*(`REM sleep percentage`/100)),
AVG(`Sleep duration`*(`Light sleep percentage`/100))
FROM sleep
GROUP BY `Exercise frequency`,`Smoking status`
ORDER BY AVG(`Sleep duration`*(`Deep sleep
percentage`/100));
```

5) **Group By on Awakening and show AVG Caffeine consumption, AVG Deep sleep time and AVG Alcohol consumption only for people who do exercise atleast 3 days a week. Show result in descending order awekenings**

```
SELECT Awakenings,
AVG(`Caffeine consumption`),
AVG(`Sleep duration`*(`Deep sleep percentage`/100)),
AVG(`Alcohol consumption`)
FROM sleep
WHERE `Exercise frequency` >= 3
GROUP BY Awakenings
ORDER BY Awakenings DESC;
```

6) **Display those power stations which have average 'Monitored Cap.(MW)' (display the values) between 1000 and 2000 and the number of occurance of the power stations (also display these values) is greater than 200.Also sort the result in ascending order.**

```
SELECT `Power Station`,
AVG(`Monitored Cap.(MW)`) AS 'Avg_Capacity',
COUNT(*) AS 'Occurence'
FROM power
GROUP BY `Power Station`
HAVING (Avg_Capacity BETWEEN 1000 AND 2000) AND
Occurence > 200
ORDER BY Avg_Capacity DESC;
```

7) **Display top 10 lowest "value" State names of which the Year either belong to 2013 or 2017 or 2021 and type is 'Public In-State'. Also the number of occurance should be between 6 to 10. Display the average value upto 2 decimal places, state names and the occurance of the states.**

```
SELECT State,
ROUND(AVG(Value),2) AS 'Avg_Value',
COUNT(*) AS 'frequency' FROM undergrad
WHERE Year IN (2013,2017,2021) AND Type = 'Public In-
State'
GROUP BY State
HAVING frequency BETWEEN 6 AND 10
ORDER BY Avg_Value ASC LIMIT 10;
```

8) **Best state in terms of low education cost (Tution Fees) in 'Public' type university.**

```
SELECT State,AVG(Value) FROM undergrad
```

```
WHERE Type LIKE '%Public%' AND Expense LIKE
'%Tuition%'
GROUP BY State
ORDER BY AVG(Value) ASC LIMIT 1;
```

9) 2nd Costliest state for Private education in year 2021. Consider, Tution and Room fee both.

```
SELECT State,AVG(Value) FROM undergrad
WHERE Year = 2021 AND Type LIKE '%Private%'
GROUP BY State
ORDER BY AVG(Value) DESC LIMIT 1,1;
```

10) Display total and average values of Discount_offered for all the combinations of 'Mode_of_Shipment' (display this feature) and 'Warehouse_block' (display this feature also) for all male ('M') and 'High' Product_importance. Also sort the values in descending order of Mode_of_Shipment and ascending order of Warehouse_block

```
SELECT Mode_of_Shipment,Warehouse_block,
SUM(Discount_offered),AVG(Discount_offered)
FROM shipment
WHERE Gender = 'M' AND Product_importance = 'high'
GROUP BY Mode_of_Shipment,Warehouse_block
ORDER BY Mode_of_Shipment DESC,Warehouse_block
ASC
```

Task Completed

SQL JOIN

Cross Join →

```
SELECT * FROM gourav.users t1
```

```
CROSS JOIN gourav.groups t2
```

Inner Join →

```
SELECT * FROM gourav.membership t1
INNER JOIN gourav.users1 t2
ON t1.user_id = t2.user_id;
```

Left Join →

```
SELECT * FROM gourav.membership t1
LEFT JOIN gourav.users t2
ON t1.user_id = t2.user_id;
```

Right Join →

```
SELECT * FROM gourav.membership t1
RIGHT JOIN gourav.users1 t2
ON t1.user_id = t2.user_id;
```

FULL OUTER JOIN

```
SELECT * FROM gourav.membership t1
LEFT JOIN gourav.users1 t2
ON t1.user_id = t2.user_id
UNION
SELECT * FROM gourav.membership t1
RIGHT JOIN gourav.users1 t2
ON t1.user_id = t2.user_id;
```

SET OPERATORS

UNION

```
SELECT * FROM gourav.person1
UNION
SELECT * FROM gourav.person2;
```

UNION ALL

```
SELECT * FROM gourav.person1
UNION ALL
SELECT * FROM gourav.person2;
```

INTERSECT

```
SELECT * FROM gourav.person1
INTERSECT
SELECT * FROM gourav.person2;
```

EXCEPT

```
SELECT * FROM gourav.person1
EXCEPT
SELECT * FROM gourav.person2;
```

SELF JOINS

```
SELECT * FROM gourav.users1 t1
JOIN gourav.users1 t2
ON t1.emergency_contact = t2.user_id;
```

JOIN On More than one column

```
SELECT * FROM gourav.students t1
JOIN gourav.class t2
ON t1.class_id = t2.class_id
AND t1.enrollment_year = t2.class_year;
```

Join on more than two column

```
SELECT * FROM flipkart.order_details t1
```

```
JOIN flipkart.orders t2
ON t1.order_id = t2.order_id
JOIN flipkart.users t3
ON t2.user_id = t3.user_id;
```

Filtering Columns

```
SELECT t1.order_id, t1.amount, t1.profit, t3.name FROM
flipkart.order_details t1
JOIN flipkart.orders t2
ON t1.order_id = t2.order_id
JOIN flipkart.users t3
ON t2.user_id = t3.user_id;
```

Q1. Find order_id, Name and city by joining users and orders.

```
SELECT t1.order_id, t2.name ,t2.city
FROM flipkart.orders t1
JOIN flipkart.users t2
ON t1.user_id = t2.user_id
```

Q2. Find order_id, Product Category by joining order_details and category

```
SELECT t1.order_id, t2.vertical
FROM flipkart.order_details t1
JOIN flipkart.category t2
ON t1.category_id = t2.category_id;
```

FILTERING ROWS

i. **Find all the orders placed in pune.**

```
SELECT * FROM flipkart.orders t1
JOIN flipkart.users t2
ON t1.user_id = t2.user_id
```

```
WHERE t2.city = 'Pune';


SELECT * FROM flipkart.orders t1
JOIN flipkart.users t2
ON t1.user_id = t2.user_id
WHERE t2.city = 'Pune' AND t2.name = 'Sarita';
```

ii. **Find all orders under Chairs Category.**

<u>Practice Questions</u>

I. **Find all Profitable orders?**

```
SELECT t1.order_id,SUM(t2.profit) FROM flipkart.orders t1
JOIN flipkart.order_details t2
ON t1.order_id = t2.order_id
GROUP BY t1.order_id
HAVING SUM(t2.profit)> 0;
```

II. **Find the customer who has placed max number of orders?**

```
SELECT name, COUNT(*) AS 'num_orders' FROM
flipkart.orders t1
JOIN flipkart.users t2
ON t1.user_id = t2.user_id
GROUP BY t2.name
ORDER BY num_orders DESC LIMIT 1;
```

III. **Which is the most Profitable Category?**

```
SELECT t2.vertical, SUM(profit) FROM flipkart.order_details
t1
JOIN flipkart.category t2
ON t1.category_id = t2.category_id
GROUP BY t2.vertical
ORDER BY SUM(profit) DESC LIMIT 1;
```

IV. Which is the most profitable state?

```
SELECT state, SUM(profit) FROM flipkart.orders t1
JOIN flipkart.order_details t2
ON t1.order_id = t2.order_id
JOIN flipkart.users t3
ON t1.user_id = t3.user_id
GROUP BY state
ORDER BY SUM(profit) DESC LIMIT 1;
```

V. Find all categories with profit higher than 5000?

```
SELECT t2.vertical, SUM(profit) FROM flipkart.order_details
t1
JOIN flipkart.category t2
ON t1.category_id = t2.category_id
GROUP BY t2.vertical
HAVING SUM(profit) > 5000;
```

TASK

11) Find out top 10 countries which have maximum A and D values.

```
SELECT A.country,A,D FROM (SELECT country,A FROM
country_ab
ORDER BY A DESC LIMIT 10) A
LEFT JOIN
```

```
(SELECT country,D FROM country_cd
ORDER BY D DESC LIMIT 10) B
ON A.country = B.country
UNION
SELECT B.country,A,D FROM (SELECT country,A FROM
country_ab
ORDER BY A DESC LIMIT 10) A
RIGHT JOIN
(SELECT country,D FROM country_cd
ORDER BY D DESC LIMIT 10) B
ON A.country = B.country
ORDER BY country;
```

12) Find out highest CL value for 2020 for every region. Also sort the result in descending order.

```
SELECT Region,MAX(CL) FROM country_cl t1
JOIN country_ab t2
ON t1.country = t2.country
WHERE t1.Edition = 2020
GROUP BY Region
ORDER BY MAX(CL) DESC;
```

13) Find top-5 most sold products.

```
SELECT Name,SUM(Quantity) AS 'total_quantity' FROM
sales t1
JOIN product t2
ON t1.ProductID = t2.ProductID
GROUP BY t1.ProductID
ORDER BY total_quantity DESC LIMIT 5;
```

14) Find sales man who sold most no of products.

```
SELECT
t1.SalesPersonID,FirstName,LastName,SUM(Quantity) AS
'num_sold' FROM sales t1
JOIN employee t2
ON t1.SalesPersonID = t2.EmployeeID
GROUP BY t1.SalesPersonID
ORDER BY num_sold DESC LIMIT 5;
```

15) Sales man name who has most no of unique customer.

```
SELECT
t1.SalesPersonID,FirstName,LastName,COUNT(DISTINCT
CustomerID) AS 'unique_customers' FROM sales t1
JOIN employee t2
ON t1.SalesPersonID = t2.EmployeeID
GROUP BY t1.SalesPersonID
ORDER BY unique_customers DESC LIMIT 5;
```

16) Sales man who has generated most revenue. Show top 5.

```
SELECT t1.SalesPersonID,t3.FirstName,t3.LastName,
ROUND(SUM(t1.Quantity * t2.Price)) AS 'total_revenue'
FROM sales t1
JOIN product t2
ON t1.ProductID = t2.ProductID
JOIN employee t3
ON t1.SalesPersonID = t3.EmployeeID
GROUP BY t1.SalesPersonID
ORDER BY total_revenue DESC LIMIT 5;
```

17) List all customers who have made more than 10 purchases.

```
SELECT
t1.CustomerID,t2.FirstName,t2.LastName,COUNT(*) FROM
sales t1
JOIN customer t2
ON t1.CustomerID = t2.CustomerID
GROUP BY t1.CustomerID
HAVING COUNT(*) > 10;
```

18) List all salespeople who have made sales to more than 5 customers.

```
SELECT
t1.SalesPersonID,FirstName,LastName,COUNT(DISTINCT
CustomerID) AS 'unique_customers' FROM sales t1
JOIN employee t2
ON t1.SalesPersonID = t2.EmployeeID
GROUP BY t1.SalesPersonID
HAVING unique_customers > 5;
```

19) List all pairs of customers who have made purchases with the same salesperson.

```
SELECT *
FROM (SELECT DISTINCT t1.CustomerID AS
'first_customer',
t2.CustomerID AS 'second_customer',
t1.SalesPersonID
FROM sales t1
JOIN sales t2
ON t1.SalesPersonID = t2.SalesPersonID
AND t1.CustomerID != t2.CustomerID) A
JOIN customer B
ON A.first_customer = B.customerID
LEFT JOIN customer C
ON A.second_customer = C.CustomerID
LEFT JOIN employee D
ON A.SalesPersonID = D.EmployeeID
```

Zomato Case Study

1) select a particular database

```
USE zomato
SELECT COUNT(*) FROM order_details
```

2) return n random records

replicated sample function from pandas

```
SELECT * FROM users ORDER BY rand() LIMIT 5
```

3) To find the NULL values

```
SELECT * FROM orders WHERE restaurant_rating IS NULL
```

To replace NULL values with 0

```
UPDATE orders SET restaurant_rating = 0
WHERE restaurant_rating IS NULL
```

4) find orders placed by each customer

```
SELECT t2.name,COUNT(*) AS '#orders' FROM orders t1
JOIN users t2
ON t1.user_id = t2.user_id
GROUP BY t2.user_id
```

5) find restaurant with most number of menu items

```
SELECT r_name,COUNT(*) AS 'menu_items' FROM
restaurants t1
JOIN menu t2
ON t1.r_id = t2.r_id
GROUP BY t2.r_id
```

6) **find number of votes and avg rating for all the restaurants**

```
SELECT r_name,COUNT(*) AS
'num_votes',ROUND(AVG(restaurant_rating),2) AS 'rating'
FROM orders t1
JOIN restaurants t2
ON t1.r_id = t2.r_id
WHERE restaurant_rating IS NOT NULL
GROUP BY t1.r_id;
```

7) **find the food that is being sold at most number of restaurants**

```
SELECT f_name,COUNT(*) FROM menu t1
JOIN food t2
ON t1.f_id = t2.f_id
GROUP BY t1.f_id
ORDER BY COUNT(*) DESC LIMIT 1;
```

8) **find restaurant with max revenue in a given month**

```
SELECT MONTHNAME(DATE(date)),date FROM orders
SELECT r_name,SUM(amount) AS 'revenue' FROM orders
t1
JOIN restaurants t2
ON t1.r_id = t2.r_id
WHERE MONTHNAME(DATE(date)) = 'July'
GROUP BY t1.r_id
ORDER BY revenue DESC LIMIT 1;
```

9) **month by month revenue for a particular restautant = kfc**

```
SELECT MONTHNAME(DATE(date)),SUM(amount) AS
'revenue' FROM orders t1
```

```
JOIN restaurants t2
ON t1.r_id = t2.r_id
WHERE r_name = 'box8'
GROUP BY MONTHNAME(DATE(date))
ORDER BY MONTH(DATE(date));
```

10) find restaurants with sales > x

```
SELECT r_name,SUM(amount) AS 'revenue' FROM orders
t1
JOIN restaurants t2
ON t1.r_id = t2.r_id
GROUP BY t1.r_id
HAVING revenue > 1500;
```

11) find customers who have never ordered

```
SELECT user_id,name FROM users
EXCEPT
SELECT t1.user_id,name FROM orders t1;
```

12) Show order details of a particular customer in a given date range

```
SELECT t1.order_id,f_name,date FROM orders t1
JOIN order_details t2
ON t1.order_id = t2.order_id
JOIN food t3
ON t2.f_id = t3.f_id
```

```
WHERE user_id = 5 AND date BETWEEN '2022-05-15' AND
'2022-07-15';
```

13) Customer favorite food

```
SELECT t1.user_id,t3.f_id,COUNT(*) FROM users t1
JOIN orders t2
ON t1.user_id = t2.user_id
JOIN order_details t3
ON t2.order_id = t3.order_id
GROUP BY t1.user_id,t3.f_id
ORDER BY COUNT(*) DESC;
```

14) find most costly restaurants(Avg price/dish)

```
SELECT r_name,SUM(price)/COUNT(*) AS 'Avg_price'
FROM menu t1
JOIN restaurants t2
ON t1.r_id = t2.r_id
GROUP BY t1.r_id
ORDER BY Avg_price ASC LIMIT 1;
```

15) find delivery partner compensation using the formula (#deliveries * 100 + 1000* avg_rating)

```
SELECT partner_name,COUNT(*) * 100  +
AVG(delivery_rating)*1000 AS 'salary'
FROM orders t1
JOIN delivery_partner t2
ON t1.partner_id = t2.partner_id
GROUP BY t1.partner_id
ORDER BY salary DESC;
```

16) find correlation between delivery_time and total rating

```
SELECT CORR(delivery_time,delivery_rating) AS 'corr'
FROM orders;
```

17) find all the veg restaurants

```
SELECT r_name FROM menu t1
JOIN food t2
ON t1.f_id = t2.f_id
JOIN restaurants t3
ON t1.r_id = t3.r_id
GROUP BY t1.r_id
HAVING MIN(type) = 'Veg' AND MAX(type) = 'Veg';
```

18) find min and max order value for all the customers

```
SELECT name,MIN(amount),MAX(amount),AVG(amount)
FROM orders t1
JOIN users t2
ON t1.user_id = t2.user_id
GROUP BY t1.user_id
```

SUB QUERY

- **Q1. Find the movie with highest rating?**

```
SELECT * FROM gourav.movies
WHERE score = (SELECT MAX(score) FROM
gourav.movies);
```

Independent Subquery → Scalar Subquery

- **Find the movies with highest profit**

```
SELECT * FROM gourav.movies
WHERE (gross-budget) = (SELECT MAX(gross-budget)
FROM gourav.movies);
```

- **Find how many movies have a rating > the avg of all the movie ratings (Find the count of above average movies)**

```
SELECT COUNT(*) FROM gourav.movies
WHERE score > (SELECT AVG(score) FROM
gourav.movies);
```

- **Find the highest rated movie of 2000**

```
SELECT * FROM gourav.movies
WHERE year = 2000 AND score = (SELECT MAX(score)
FROM movies WHERE year = 2000);
```

- **Find the highest rated movie among all movies whose number of votes are > the dataset avg votes.**

```
SELECT * FROM gourav.movies
WHERE score = (SELECT MAX(score) FROM movies
WHERE votes  > (SELECT AVG(votes) FROM
gourav.movies));
```

<u>Independent subquery – Row subquery (one column Multiple Rows)</u>

- **Find all users who never ordered?**

```
SELECT * FROM gourav.users
WHERE user_id NOT IN (SELECT DISTINCT(user_id) from
gourav.orders);
```

- **Find all movies made by top 3 directors (in terms of total gross income)**

```
SELECT * FROM gourav.movies
WHERE director IN (SELECT director FROM gourav.movies
GROUP BY director ORDER BY SUM(gross) DESC LIMIT
3);
```

** Throwing an error this version not supporte limit in subquery

ANOTHER APPROACH →

```
WITH top_directors AS (SELECT director FROM
gourav.movies
                                GROUP BY director
                                ORDER BY SUM(gross)
                                        DESC LIMIT 3)

SELECT * FROM gourav.movies
WHERE director IN (SELECT * FROM top_directors)
```

- **Find all movies of all those actors whose filmography's avg rating > 8.5 (take 25000 votes as cutoff)**

```
SELECT * FROM gourav.movies WHERE
star IN (SELECT star FROM gourav.movies
                WHERE votes > 25000
                GROUP BY star
                HAVING AVG(score) > 8.5);
```

Independent subquery – Table subquery (Multiple column Multiple Rows)

- **Find the most profitable movie of each year.**

```
SELECT star FROM gourav.movies
WHERE (year,gross-budget) IN (SELECT year, MAX(gross-
budget) FROM gourav.movies GROUP BY year)
```

- **Find the highest rated movies of each genre votes cut off of 25000?**

```
SELECT * FROM gourav.movies
WHERE (genre, score) IN (SELECT genre, MAX(score)
                         FROM gourav.movies
                         WHERE votes > 25000
                               GROUP BY  genre)
                               AND votes > 25000
```

- **Find the highest grossing movies of top 5 actor/director combo in terms of total gross income.**

```
WITH top_duos AS (
SELECT star, director, MAX(gross)
FROM gourav.movies
GROUP BY star, director
ORDER BY SUM(gross) DESC LIMIT 5
)

SELECT * FROM movies
WHERE (star,director,gross) IN (SELECT * FROM top_duos)
```

<u>Correlated Subquery</u>

- **Find all the movies that have a rating higher than the average rating of movies in the same genre.**

```
SELECT * FROM gourav.movies M1
WHERE score > (SELECT AVG(score) FROM
gourav.movies m2 WHERE m2.genre = m1.genre)
```

- **Find the favourite food of each customer.**

```
WITH fav_food AS (
```

```
SELECT t2.user_id, name, f_name, COUNT(*) AS
'frequency' FROM gourav.users t1
JOIN gourav.orders t2 ON t1.user_id = t2.user_id
JOIN gourav.order_details t3 ON t2.order_id = t3.order_id
JOIN gourav.food t4 ON t3.f_id = t4.f_id
GROUP BY t2.user_id, t3.f_id
)
SELECT * FROM fav_food f1
WHERE frequency = ( SELECT MAX(frequency)
                    FROM fav_food f2
                    WHERE f2.user_id =
f1.user_id)
```

Usage With Select

- **Get the percentage of votes for each movie compared to the total number of votes.**

```
SELECT name, (votes/(SELECT SUM(votes) FROM
movies)) * 100 FROM gourav.movies;
```

- **Display all movie names, genre, score and avg(score) of genre?**

```
SELECT name,  genre,score,
(SELECT AVG(score) FROM gourav.movies m2 WHERE
m2.genre = m1.genre)
FROM gourav.movies m1;
```

Usage With From

- **Display average rating of all the restaurants.**

```
SELECT r_name, avg_rating FROM
(SELECT r_id, AVG(restaurant_rating) AS 'avg_rating'
FROM gourav.orders
GROUP BY r_id) t1 JOIN restaurants t2 ON t1.r_id = t2.r_id
```

Usage with Having

- **Find the genres avg score > avg score of all the movies?**

```
SELECT genre, AVG(score)
FROM gourav.movies
GROUP BY genre
HAVING AVG(score) > (SELECT AVG (score) FROM
gourav.movies);
```

Subquery In Insert

- **Populate a already created loyal_customers table with records of only those customers who have orderd food more than 3 times.**

```
INSERT INTO loyal_users
(user_id,name)
SELECT t1.user_id,name
from gourav.orders t1
JOIN users t2 ON t1.user_id = t2.user_id
GROUP BY user_id
HAVING COUNT(*) > 3
```

Subquery in update

- **Populate the money column of loyal customer table using the orders table. provide 10% off money to all customers based on their order value.**

```
UPDATE loyal_users
SET money = (SELECT SUM(amount)*0.1
                    FROM gourav.orders
                    WHERE orders.user_id =
loyal_users.user_id);
```

Subquery in Delete

- **Delete all the customers record who have never ordered**

```
DELETE FROM gourav.users
WHERE user_id IN (SELECT user_id FROM gourav.users
WHERE user_id NOT IN (SELECT DISTINCT (user_id)
FROM orders));
```

TASK

1) **Display the names of athletes who won a gold medal in the 2008 Olympics and whose height is greater than the average height of all athletes in the 2008 Olympics.**

```
SELECT * FROM olympics
WHERE Year = 2008 AND
Medal = 'Gold' AND
Height > (SELECT AVG(Height) FROM olympics WHERE
Year = 2008);
```

2) **Display the names of athletes who won a medal in the sport of basketball in the 2016 Olympics and whose weight is less than**

the average weight of all athletes who won a medal in the 2016 Olympics.

```
SELECT name FROM olympics
WHERE Year = 2016 AND
Sport = 'Basketball' AND
Medal IS NOT NULL AND
height < (SELECT AVG(Height) FROM olympics WHERE
Year = 2016 AND Medal IS NOT NULL);
```

3) **Display the names of all athletes who have won a medal in the sport of swimming in both the 2008 and 2016 Olympics.**

```
SELECT * FROM olympics
WHERE Sport = 'Swimming' AND
Year IN (2008,2016) AND
Medal IS NOT NULL;
```

4) **Display the names of all countries that have won more than 50 medals in a single year.**

```
SELECT country,Year,COUNT(*) FROM olympics
WHERE Medal IS NOT NULL AND country IS NOT NULL
GROUP BY country,Year
HAVING COUNT(*) > 50
ORDER BY Year,country;
```

5) **Display the names of all athletes who have won medals in more than one sport in the same year.**

```
SELECT DISTINCT name FROM olympics
WHERE ID in (SELECT DISTINCT ID FROM olympics
                WHERE Medal IS NOT NULL
                GROUP BY ID,Year,Sport
                HAVING COUNT(Medal) > 1
                ORDER BY COUNT(Medal) DESC);
```

6) **What is the average weight difference between male and female athletes in the Olympics who have won a medal in the same event?**

```
WITH result AS (
        SELECT * FROM olympics
        WHERE Medal IS NOT NULL
)
SELECT AVG(A.Weight - B.Weight) FROM result A
JOIN result B
ON A.Event = B.Event
ANd A.Gender != B.Gender;
```

7) **How many patients have claimed more than the average claim amount for patients who are smokers and have at least one child, and belong to the southeast region?**

```
SELECT COUNT(claim) FROM insurance
WHERE claim > (SELECT AVG(Claim) FROM insurance
                        WHERE smoker = 'Yes' AND
                        region = 'southwest' AND
                        children >= 1);
```

8) **How many patients have claimed more than the average claim amount for patients who are not smokers and have a BMI greater than the average BMI for patients who have at least one child?**

```
SELECT COUNT(claim) FROM insurance
WHERE claim > (SELECT AVG(claim) FROM insurance
                        WHERE smoker = 'No' AND
                        bmi > (SELECT AVG(bmi) FROM
insurance WHERE children >= 1));
```

9) **How many patients have claimed more than the average claim amount for patients who have a BMI greater than the average**

BMI for patients who are diabetic, have at least one child, and are from the southwest region?

```
SELECT COUNT(claim) FROM insurance
WHERE claim > (SELECT AVG(claim) FROM insurance
WHERE
                        bmi > (SELECT AVG(bmi) FROM
insurance
                                    WHERE children >= 1 AND
                  diabetic = 'Yes' AND region = 'southwest'));
```

10) What is the difference in the average claim amount between patients who are smokers and patients who are non-smokers, and have the same BMI and number of children?

```
SELECT AVG(A.claim - B.claim) FROM insurance A
JOIN insurance B
ON A.bmi = B.bmi
AND A.smoker != B.smoker
AND A.children = B.children
```

Task Completed

Windows function

```
CREATE TABLE gourav.marks (
 student_id INTEGER PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(255),
   branch VARCHAR(255),
   marks INTEGER
);
```

```
INSERT INTO marks (name,branch,marks)VALUES
```

```
('Nitish','EEE',82),
('Rishabh','EEE',91),
('Anukant','EEE',69),
('Rupesh','EEE',55),
('Shubham','CSE',78),
('Ved','CSE',43),
('Deepak','CSE',98),
('Arpan','CSE',95),
('Vinay','ECE',95),
('Ankit','ECE',88),
('Anand','ECE',81),
('Rohit','ECE',95),
('Prashant','MECH',75),
('Amit','MECH',69),
('Sunny','MECH',39),
('Gautam','MECH',51)


SELECT *, AVG(marks) OVER(PARTITION BY branch)
FROM gourav.marks;


SELECT *,
MIN(marks) OVER(),
MAX(marks) OVER()
 FROM gourav.marks;
SELECT *,
AVG(marks) OVER() AS 'Overall_avg',
MIN(marks) OVER(),
MAX(marks) OVER(),
MIN(marks) OVER(PARTITION BY branch),
MAX(marks) OVER(PARTITION BY branch)
FROM gourav.marks
ORDER BY student_id;
```

Aggregate Function with OVER()

- **Find all the students who have marks higher than the avg marks of their respective branch**

```
SELECT * FROM
(SELECT *,
AVG(marks) OVER(PARTITION BY branch) AS 'branch_avg'
FROM gourav.marks) t
WHERE t.marks > t.branch_avg;
```

RANK/ DENSE_RANK/ ROW_NUMBER

- **Create roll no. from branch and marks**

```
SELECT *,
RANK() OVER (PARTITION BY branch ORDER BY marks
DESC)
FROM gourav.marks;

SELECT *,
RANK() OVER (PARTITION BY branch ORDER BY marks
DESC),
DENSE_RANK() OVER (PARTITION BY branch ORDER BY
marks DESC)
FROM gourav.marks;


SELECT *,
ROW_NUMBER() OVER()
FROM gourav.marks;


SELECT *,
ROW_NUMBER() OVER(PARTITION BY branch)
FROM gourav.marks;

SELECT *,
```

```
CONCAT(branch,'-',ROW_NUMBER() OVER(PARTITION
BY branch))
FROM gourav.marks;
```

- **Find top 2 most paying customers of each month**

```
SELECT * FROM ( SELECT MONTHNAME(date) AS 'month'
, user_id,SUM(amount) AS 'total',
RANK() OVER(PARTITION BY MONTHNAME(date)
ORDER BY SUM(amount) DESC) AS 'Month_rank'
FROM gourav.orders
GROUP BY MONTHNAME(date), user_id
ORDER BY MONTH(date)) t
WHERE t.month_rank < 3
ORDER BY month_rank ASC;
```

FIRST_VALUE/LAST_VALUE/NTH_VALUE

```
SELECT * ,
FIRST_VALUE (marks) OVER (ORDER BY marks DESC)
FROM gourav.marks;
```

```
SELECT * ,
LAST_VALUE (name) OVER (PARTITION BY branch
                        ORDER BY marks DESC
                        ROWS BETWEEN UNBOUNDED
PRECEDING AND UNBOUNDED FOLLOWING)
FROM gourav.marks;
```

```
SELECT * ,
```

```
NTH_VALUE (name,2) OVER (PARTITION BY branch
                    ORDER BY marks DESC
                    ROWS BETWEEN UNBOUNDED
PRECEDING AND UNBOUNDED FOLLOWING)
FROM gourav.marks;
```

- **Find the branch toppers**

```
SELECT name, branch, marks FROM
(SELECT *,
FIRST_VALUE(name) OVER (PARTITION BY branch
ORDER BY marks DESC) AS 'topper_name',
FIRST_VALUE(marks) OVER (PARTITION BY branch
ORDER BY marks DESC) AS 'topper_marks'
FROM gourav.marks) t
WHERE t.name = t.topper_name AND t.marks =
t.topper_marks ;
```

```
SELECT name, branch, marks FROM
(SELECT *,
LAST_VALUE(name) OVER (PARTITION BY branch
ORDER BY marks DESC
                                    ROWS BETWEEN
UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS 'topper_name',
```

```
LAST_VALUE(marks) OVER (PARTITION BY branch
ORDER BY marks DESC
                                        ROWS BETWEEN
UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS 'topper_marks'
FROM gourav.marks) t
WHERE t.name = t.topper_name AND t.marks =
t.topper_marks ;
```

LEAD & LAG

```
SELECT *,
LAG (marks) OVER(ORDER BY student_id)
FROM gourav.marks;

SELECT *,
LEAD (marks) OVER(PARTITION BY branch ORDER BY
student_id)
FROM gourav.marks;


SELECT *,
LAG (marks) OVER(PARTITION BY branch ORDER BY
student_id),
LEAD (marks) OVER(PARTITION BY branch ORDER BY
student_id)
FROM gourav.marks;
```

Q1**. Find the Month on Month Revenue of orders.**

```
SELECT MONTHNAME(date),SUM(amount),
```

```
((SUM(amount) - LAG(SUM(amount)) OVER (ORDER BY
MONTH (date)))/LAG(SUM(amount)) OVER (ORDER BY
MONTH (date)))*100
FROM gourav.orders
GROUP BY MONTHNAME(date)
ORDER BY MONTH(date) ASC;
```

TASK

```
USE sql_tasks;
SELECT * FROMinsurance;
```

11) What are the top 5 patients who claimed the highest insurance amounts?

```
SELECT *,DENSE_RANK() OVER(ORDER BY claim DESC)
FROM insurance LIMIT 5;
```

12) What is the average insurance claimed by patients based on the number of children they have?

```
SELECT children,avg_claim FROM (SELECT *, AVG(claim)
OVER(PARTITION BY children) AS avg_claim,
ROW_NUMBER() OVER(PARTITION BY children) AS
row_num FROMinsurance) t WHEREt.row_num = 1;
```

13) What is the highest and lowest claimed amount by patients in each region?

```
SELECT region,min_claim,max_claim FROM (SELECT *,
MIN(claim) OVER(PARTITION BY region) AS min_claim,
MAX(claim) OVER(PARTITION BY region) AS max_claim,
ROW_NUMBER() OVER(PARTITION BY region) AS
row_num FROMinsurance) t WHEREt.row_num = 1;
```

14) What is the difference between the claimed amount of each patient and the claimed amount of the first patient?

```
SELECT *, claim- FIRST_VALUE(claim) OVER() AS diff
FROMinsurance;
```

15) For each patient, calculate the difference between their claimed amount and the average claimed amount of patients with the same number of children.

```
SELECT *, claim- AVG(claim) OVER(PARTITION BY
children) FROMinsurance;
```

16) Show the patient with the highest BMI in each region and their respective overall rank.

```
SELECT * FROM(SELECT *, RANK() OVER(PARTITION BY
region ORDER BY bmi DESC) AS group_rank, RANK()
OVER(ORDER BY bmi DESC) AS overall_rank
FROMinsurance) t WHEREt.group_rank = 1;
```

17) Calculate the difference between the claimed amount of each patient and the claimed amount of the patient who has the highest BMI in their region.

```
SELECT *, claim- FIRST_VALUE(claim) OVER(PARTITION
BY region ORDER BY bmi DESC) FROMinsurance;
```

18) For each patient, calculate the difference in claim amount between the patient and the patient with the highest claim amount among patients with the and smoker status, within the same region. Return the result in descending order difference.

```
SELECT *, (MAX(claim) OVER(PARTITION BY
region,smoker)- claim) AS claim_diff FROMinsurance
ORDERBYclaim_diff DESC;
```

19) For each patient, find the maximum BMI value among their next three records (ordered by age).

```
SELECT *, MAX(bmi) OVER(ORDER BY age ROWS
BETWEEN1FOLLOWINGAND3FOLLOWING)
FROMinsurance;
```

20) For each patient, find the rolling average of the last 2 claims.

```
SELECT *, AVG(claim) OVER(ROWS BETWEEN 2
PRECEDING AND 1 PRECEDING) FROMinsurance;
```

21) Find the first claimed insurance value for male and female patients,within each region order the data by patient age in ascending order, and only include patients who are non-diabetic and have a bmi value between 25 and 30.

```
WITH filtered_data AS ( SELECT * FROMinsurance
WHEREdiabetic = 'No' AND bmi BETWEEN 25 AND 30 )
SELECT region,gender,first_claim FROM (SELECT *,
FIRST_VALUE(claim) OVER(PARTITION BY region,gender
ORDER BY age) AS first_claim, ROW_NUMBER()
OVER(PARTITION BY region,gender ORDER BY age) AS
row_num FROMfiltered_data) t WHEREt.row_num = 1
```

Task Completed

RANKING

```
SELECT * FROM
        (SELECT BattingTeam,batter,SUM(batsman_run) AS
        'total_runs',
        DENSE_RANK() OVER(PARTITION BY BattingTeam
        ORDER BY SUM(batsman_run) DESC)  AS
        'rank_within_team'
        FROM gourav.ipl
        GROUP BY BattingTeam,batter) t
        WHERE t.rank_within_team < 6
        ORDER BY t.BattingTeam, t.rank_within_team;
```

Cumulative Sum

- **Q1. Find the total score of V Kohli after 50th ,100th & 200th match?**

```
SELECT * FROM (SELECT
CONCAT('Match-',ROW_NUMBER() OVER(ORDER BY ID
)) AS 'match_no',
SUM(batsman_run) AS 'runs_scored',
SUM(SUM(batsman_run)) OVER(ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW) AS
'Career_Runs'
FROM gourav.ipl
WHERE batter = 'V Kohli'
GROUP BY ID) t
WHERE match_no = 'Match-50' OR match_no = 'Match-100'
OR match_no = 'Match-200'
```

	match_no	runs_scored	Career_Runs
▶	Match-50	11	1131
	Match-100	13	2650
	Match-200	41	6334

Cumulative Average

```
SELECT * FROM (SELECT
CONCAT('Match-',ROW_NUMBER() OVER(ORDER BY ID
)) AS 'match_no',
SUM(batsman_run) AS 'runs_scored',
SUM(SUM(batsman_run)) OVER w AS 'Career_Runs',
AVG (SUM(batsman_run)) OVER w AS 'career_Avg'


FROM gourav.ipl
WHERE batter = 'V Kohli'
GROUP BY ID
WINDOW w AS (ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW)) t;
```

match_no	runs_scored	Career_Runs	career_Avg
Match-1	1	6634	30.8558
Match-2	23	6633	30.9953
Match-3	13	6610	31.0329
Match-4	12	6597	31.1179
Match-5	1	6585	31.2085

RUNNING AVERAGE

```
SELECT * FROM (SELECT
CONCAT('Match-',ROW_NUMBER() OVER(ORDER BY ID
)) AS 'match_no',
SUM(batsman_run) AS 'runs_scored',
SUM(SUM(batsman_run)) OVER w AS 'Career_Runs',
AVG (SUM(batsman_run)) OVER w AS 'career_Avg',
AVG(SUM(batsman_run)) OVER (ROWS BETWEEN 9
PRECEDING AND CURRENT ROW) AS 'Rolling_Avg'
FROM gourav.ipl
WHERE batter = 'V Kohli'
GROUP BY ID
WINDOW w AS (ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW)) t
```

PERCENT OF TOTAL

```
SELECT f_name,
```

```
(total_value/SUM(total_value) OVER())*100 AS
'Percent_of_total' FROM
(SELECT f_id, SUM(amount) AS 'total_value' FROM
gourav.orders t1
JOIN order_details t2


ON t1.order_id = t2.order_id
WHERE r_id = 1
GROUP BY f_id ) t
JOIN food t3
ON t.f_id = t3.f_id
ORDER BY (total_value/SUM(total_value) OVER())*100
DESC;
```

Percent Change

```
SELECT YEAR(Date), MONTHNAME(Date), SUM(views)
AS 'views',
((SUM(views) - LAG(SUM(views)) OVER(ORDER BY
YEAR(Date),MONTH(Date))) / LAG(SUM(views))
OVER(ORDER BY YEAR(Date),MONTH(Date)))* 100 AS
'percent_change'
FROM gourav.youtube_views
GROUP BY YEAR(Date), MONTHNAME(Date)
ORDER BY YEAR(Date),MONTH(Date);


SELECT YEAR(Date), QUARTER(Date), SUM(views) AS
'views',
((SUM(views) - LAG(SUM(views)) OVER(ORDER BY
YEAR(Date), QUARTER(Date))) / LAG(SUM(views))
```

```
OVER(ORDER BY YEAR(Date), QUARTER(Date)))* 100 AS
'percent_change'
FROM gourav.youtube_views
GROUP BY YEAR(Date), QUARTER(Date)
ORDER BY YEAR(Date), QUARTER(Date);

SELECT * ,
((Views - LAG(Views,7) OVER (ORDER BY Date)) /
LAG(Views,7) OVER(ORDER BY Date)) * 100 AS
'weekly_percent_change'
FROM gourav.youtube_views;
```

PERCENTILE & QUANTILE

- **Find the median marks of all the students**

```
SELECT *,
PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY
marks) OVER() AS 'median_marks'
FROM gourav.marks
```

- **Find branch wise median of student marks**

```
SELECT *,
PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY
marks) OVER(PARTITION BY branch) AS 'median_marks'
FROM gourav.marks
```

- **For removal of outliers**

```
SELECT * FROM
```

```
(SELECT *,
PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY
marks) OVER () AS 'Q1',
PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY
marks) OVER () AS 'Q3'
FROM gourav.marks) t
WHERE t.marks > t.Q1 - (1.5*(t.Q3 - t.Q1)) AND
t.marks < t.Q3 + (1.5* (t.Q3 - t.Q1))
ORDER BY t.student_id;
```

```
SELECT * FROM
(SELECT *,
PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY
marks) OVER () AS 'Q1',
PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY
marks) OVER () AS 'Q3'
FROM gourav.marks) t
WHERE t.marks <= t.Q1 - (1.5*(t.Q3 - t.Q1));
```

Segmentation

```
SELECT brand_name, model, price,
CASE
        WHEN bucket = 1 THEN 'budget'
   WHEN bucket = 2 THEN 'mid-range'
   WHEN bucket = 3 THEN 'premium'
END AS 'phone_type'
FROM (SELECT brand_name, model, price,
NTILE (3) OVER (ORDER BY price) AS 'bucket'
```

```
FROM gourav.smartphones) t


SELECT brand_name, model, price,
CASE
        WHEN bucket = 1 THEN 'budget'
   WHEN bucket = 2 THEN 'mid-range'
   WHEN bucket = 3 THEN 'premium'
END AS 'phone_type'
FROM (SELECT brand_name, model, price,
NTILE (3) OVER (PARTITION BY brand_name ORDER BY
price) AS 'bucket'
FROM gourav.smartphones) t
```

CUMULATIVE DISTRIBUTION

```
SELECT * FROM
(SELECT *, CUME_DIST() OVER (ORDER BY marks) AS
'Percentile_Score'
FROM gourav.marks ) t
where t.Percentile_Score > 0.90
```

- **Partition By multiple columns**

```
SELECT * FROM (SELECT
source,destination,airline,AVG(price) as 'avg_fare',
DENSE_RANK() OVER (PARTITION BY source, destination
ORDER BY AVG(price)) AS 'rank'
FROM gourav.flights
GROUP BY source, destination,airline) t
WHERE t.rank < 2
```

WILDCARDS

- **There are two types of wild cards**

  Underscore (_)
  Percent (%)

- **-- How to use underscore(_) wildcards**

```
USE gourav;
```

- **Select the movie name which have present five characters**

```
SELECT name
FROM movies
WHERE name LIKE '_____';
```

- **Find the movie which starts with A and have present five characters**

```
SELECT name
FROM movies
WHERE name LIKE 'A_____';
```

- **Find the movie name which has include man word at any position in their name**

```
SELECT name FROM movies
Where name LIKE '%man%';
```

- **If you find name starts with man then:**

```
SELECT name FROM movies
Where name LIKE 'man%';
```

- **If you find name ends with man**

```
SELECT name FROM movies
Where name LIKE '%man';
```

STRING FUNCTION

- **<u>UPPER/LOWER</u>**

```
SELECT name ,UPPER(name), LOWER(name) FROM
movies;
```

- **<u>CONCAT & CONCAT_WS (ws- with separater)</u>**

SELECT CONCAT(name," -- ", director) FROM movies;
SELECT CONCAT(name," -- ", director, " ---- ", star) FROM movies;

- **CONCAT_WS (ws- with separater)**

SELECT CONCAT_WS(" ---- ", name, director, star) FROM movies;

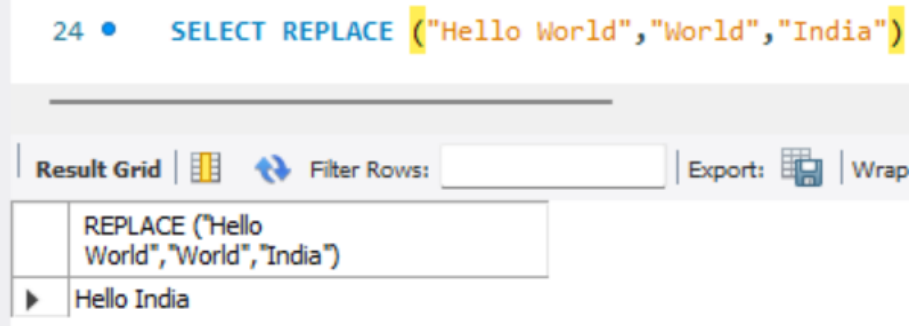
- **SUBSTR→ LAST 5 CHARS**

SELECT name, SUBSTR(name,1,5) FROM movies;
SELECT name, SUBSTR(name,1) FROM movies;
SELECT name, SUBSTR(name,5,5) FROM movies;

SELECT name, SUBSTR(name,-5,1) FROM movies;
SELECT name, SUBSTR(name,-7) FROM movies;

- **REPLACE**

SELECT REPLACE ("Hello World","World","India")



SELECT name, REPLACE (name, “man” , “woman”) FROM movies;

- **REVERSE**

```
SELECT REVERSE("HELLO");

SELECT name FROM movies
WHERE name = REVERSE(name);
```

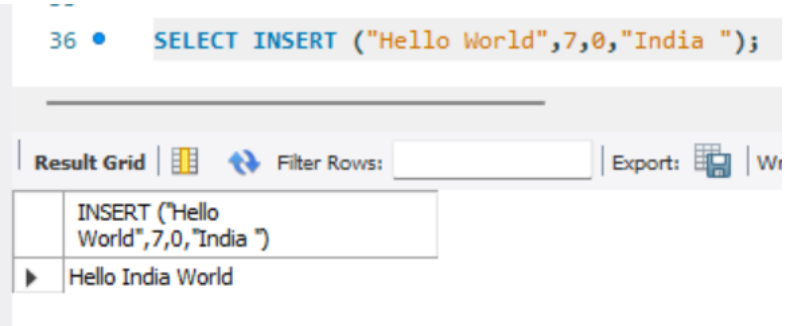
- **CHAR_LENGTH vs LENGTH**

```
SELECT name, LENGTH(name) FROM movies;
SELECT name, CHAR_LENGTH(name) FROM movies;


SELECT name, LENGTH(name),CHAR_LENGTH(name)
FROM movies
WHERE LENGTH(name) != CHAR_LENGTH(name);
```

- **INSERT**

```
SELECT INSERT ("Hello World",7,0,"India ");
```




```
SELECT INSERT ("Hello World",7,5,"India ");
```

```
37 •  SELECT INSERT ("Hello World",7,5,"India ");
```

Result Grid | Filter Rows: | Export: | W

INSERT ("Hello World",7,5,"India ")
Hello India

- **LEFT AND RIGHT**

```
SELECT name, LEFT(name,3) FROM movies;
SELECT name, RIGHT(name,3) FROM movies;
```

- **REPEAT**

```
SELECT REPEAT (name,3) FROM movies;
```

- **TRIM**

```
SELECT TRIM("                    GOURAB            ");
SELECT TRIM(BOTH "." FROM
".....................GOURAB.............")
SELECT TRIM(LEADING "." FROM
".....................GOURAB.............")
SELECT TRIM(TRAILING "." FROM
".....................GOURAB.............")
```

- **LTRIM / RTRIM**

```
SELECT LTRIM("        GOURAB            ");
SELECT RTRIM("        GOURAB            ");
```

- **SUBSTRING_INDEX(SPLIT)**

```
SELECT
SUBSTRING_INDEX("gourabsingh09@gmail.com","@",1);
SELECT
SUBSTRING_INDEX("gourabsingh09@gmail@com","@",2);
SELECT
SUBSTRING_INDEX("gourabsingh09@gmail@com","@",-
1);
```

- **STRCMP**

```
SELECT STRCMP("Delhi" , "Mumbai");
SELECT STRCMP("Mumbai" , "Delhi");
SELECT STRCMP("Delhi" , " DELHI");
```

- **LOCATE**

```
SELECT LOCATE("w", "hello world");
SELECT LOCATE("w", "hello world",5);
```

- **LPAD & RPAD**

```
SELECT LPAD('9895629488','13','+91');
SELECT RPAD('9895629488','13','+91');
```

DATA CLEANING USING SQL On LAPTOP DATASET

DATA CLEANING

1) Create Backup
2) Check Number Of Rows
3) Check memory consumption for refrence
4) Drop Non important columns
5) Drop Null Values

6) Drop Duplicates
7) Clean RAM -> Change column data type
8) Round price column and change to integer
9) Change the Opsys col
10) GPU
11) CPU

USE singh;

1) Create a backup file for dataset

```
CREATE TABLE laptops_backup LIKE laptopdata;
```

Move all the data from current table to another table

```
INSERT INTO laptops_backup
SELECT * FROM laptopdata;
```

2) Check Number Of Rows

```
SELECT count(*) FROM singh.laptopdata;
```

3) Check memory consumption for refrence

```
 SELECT DATA_LENGTH/1024 FROM
information_schema.TABLES
WHERE TABLE_SCHEMA = 'singh'
AND TABLE_NAME = 'laptopdata';
```

➔ Output unit will be in KB

4) Drop Non important columns

```
ALTER TABLE laptopdata DROP COLUMN `Unnamed: 0`;
```

Note → rename the column name

```
ALTER TABLE laptopdata
CHANGE COLUMN `Unnamed: 0` `Index` INT;
```

5) Drop Null Values

```
DELETE FROM laptopdata
WHERE `index` IN (SELECT `Index` FROM laptopdata
WHERE Company IS NULL AND TypeName IS NULL AND
Inches IS NULL
AND ScreenResolution IS NULL AND Cpu IS NULL AND
Ram IS NULL
AND Memory IS NULL AND Gpu IS NULL AND OpSys IS
NULL AND
WEIGHT IS  NULL AND Price IS NULL);
```

```
DELETE FROM laptopdata
WHERE `index` IN (
   SELECT * FROM (
      SELECT `index` FROM laptopdata
      WHERE Company IS NULL AND TypeName IS NULL
AND Inches IS NULL
      AND ScreenResolution IS NULL AND Cpu IS NULL
AND Ram IS NULL
```

```
        AND Memory IS NULL AND Gpu IS NULL AND OpSys
IS NULL AND
        WEIGHT IS NULL AND Price IS NULL
    ) AS tempTable
);
```

6) Drop Duplicates

```
DELETE FROM singh.laptopdata
WHERE id NOT IN (SELECT MIN(id)
FROM singh.laptopdata
GROUP BY
Company,TypeName,ScreenResolution,Cpu,Ram,Memory,G
pu,OpSys,WEIGHT,Price);
```

7) Clean RAM -> Change column data type

Ram Memory Update:

Description: Updating the `ram_memory` column in the `laptopsdata` table by replacing 'GB' in the `Ram` column.

*** Query*:**

```
UPDATE laptopdata
    SET ram = REPLACE(Ram, 'GB', '');
```

→Change data type

```
ALTER TABLE laptopdata MODIFY COLUMN Ram INTEGER;
```

Weight Update:

Description: Updating the `weight` column in the `laptopsdata` table by replacing 'kg' in the `Weight` column.

*** Query*:**

```
UPDATE laptopdata
    SET weight = REPLACE(Weight, 'kg', '');
```

8) Round price column and change to integer

Price Update:

Description: Updating the `price` column in the `laptopdata` table by rounding the `Price` column.

*** Query*:**

```
UPDATE laptopdata
    SET price = ROUND(Price);


ALTER TABLE laptopdata MODIFY COLUMN Price
INTEGER;
```

9) Change the Opsys col

```
SELECT OpSys,
CASE
        WHEN OpSys LIKE '%mac%' THEN 'macos'
   WHEN OpSys LIKE 'windows%' THEN 'windows'
   WHEN OpSys LIKE '%linux%' THEN 'linux'
   WHEN OpSys = 'No OS' THEN 'N/A'
   ELSE 'other'
END AS 'os_brand'
FROM laptopdata;

UPDATE laptopdata
SET OpSys =
CASE
        WHEN OpSys LIKE '%mac%' THEN 'macos'
   WHEN OpSys LIKE 'windows%' THEN 'windows'
   WHEN OpSys LIKE '%linux%' THEN 'linux'
   WHEN OpSys = 'No OS' THEN 'N/A'
   ELSE 'other'
END;
```

10) GPU

```
ALTER TABLE laptopdata
ADD COLUMN gpu_brand VARCHAR(255) AFTER Gpu,
ADD COLUMN gpu_name VARCHAR(255) AFTER
gpu_brand;
```

GPU Brand Update:

Description: Updating the `gpu_brand` column in the `laptopsdata` table by extracting the brand from the `Gpu` column.

*** Query*:**

```
UPDATE laptopdata
    SET gpu_brand = SUBSTRING_INDEX(Gpu, ' ', 1);
```

GPU Name Update:

Description: Updating the `gpu_name` column in the `laptopsdata` table by removing the `gpu_brand` from the `Gpu` column.

*** Query*:**

```
UPDATE laptopdata
    SET gpu_name = REPLACE(Gpu, gpu_brand, '');
```

```
ALTER TABLE laptopdata DROP COLUMN Gpu;
```

11) CPU

CPU Brand Update:

Description: Updating the `cpu_brand` column in the `laptopsdata` table by extracting the brand from the `Cpu` column.

*** Query*:**

```
SELECT * FROM laptopdata;

ALTER TABLE laptopdata
ADD COLUMN cpu_brand VARCHAR(255) AFTER Cpu,
ADD COLUMN cpu_name VARCHAR(255) AFTER
cpu_brand,
ADD COLUMN cpu_speed DECIMAL(10,1) AFTER
cpu_name;


    UPDATE laptopdata
    SET cpu_brand = SUBSTRING_INDEX(Cpu, ' ', 1);
```

CPU Speed Update:

Description: Updating the `cpu_speed` column in the `laptopsdata` table by extracting and converting the CPU speed from the `Cpu` column.

*** Query*:**

```
UPDATE laptopdata
    SET cpu_speed =
CAST(REPLACE(SUBSTRING_INDEX(Cpu, ' ', -1), 'GHz', '')
AS DECIMAL(10,2));
```

CPU Name Update:

Description: Updating the `cpu_name` column in the `laptopsdata` table by extracting and cleaning the CPU name from the `Cpu` column.

*** Query*:**

```
UPDATE laptopdata
    SET cpu_name = REPLACE(REPLACE(Cpu, cpu_brand, ''), SUBSTRING_INDEX(REPLACE(Cpu, cpu_brand, ''), ' ', -1), '');



SELECT * FROM laptopdata;

ALTER TABLE laptopdata DROP COLUMN Cpu;
```

- ❖ Make 3 New Columns From Screen Resolution

1) Resolution Height
2) Resolution Width
3) TouchScreen
   - ➔ In the touchscreen column fill with 1/0 Where the laptop will be touchscreen fill with one and if the laptop is not a touchscreen fill with 0.

```
USE singh;

SELECT ScreenResolution,
SUBSTRING_INDEX(SUBSTRING_INDEX(ScreenResolution,' ',-1),'x',1),
SUBSTRING_INDEX(SUBSTRING_INDEX(ScreenResolution,' ',-1),'x',-1)
FROM laptopdata;


ALTER TABLE laptopdata
ADD COLUMN resolution_width INTEGER AFTER ScreenResolution,
```

```
ADD COLUMN resolution_height INTEGER AFTER
resolution_width;

UPDATE laptopdata
SET resolution_width =
SUBSTRING_INDEX(SUBSTRING_INDEX(ScreenResolutio
n,' ',-1),'x',1),
resolution_height =
SUBSTRING_INDEX(SUBSTRING_INDEX(ScreenResolutio
n,' ',-1),'x',-1);



ALTER TABLE laptopdata
ADD COLUMN touchscreen INTEGER AFTER
resolution_height;


UPDATE laptopdata
SET touchscreen = ScreenResolution LIKE '%Touch%';
```

- Drop the Screen Resolution Column

```
ALTER TABLE laptopdata
DROP COLUMN ScreenResolution;
```

- Want to remove extra information from cpu column

```
SELECT cpu_name,
SUBSTRING_INDEX(TRIM(cpu_name),' ',2)
FROM laptopdata;
```

- UPDATE

```
UPDATE laptopdata
```

```
SET cpu_name = SUBSTRING_INDEX(TRIM(cpu_name),'
',2);
```

- Breakdown the memory column in three column for keep the all information separately

1) Type
2) Primary storage
3) Secondary Storage

```
ALTER TABLE laptopdata
ADD COLUMN memory_type VARCHAR(255) AFTER
Memory,
ADD COLUMN primary_storage INTEGER AFTER
memory_type,
ADD COLUMN secondary_storage INTEGER AFTER
primary_storage;
```

```
SELECT Memory,
CASE
        WHEN Memory LIKE '%SSD%' AND Memory LIKE
'%HDD%' THEN 'Hybrid'
   WHEN Memory LIKE '%SSD%' THEN 'SSD'
   WHEN Memory LIKE '%HDD%' THEN 'HDD'
   WHEN Memory LIKE '%Flash Storage%' THEN 'Flash
Storage'
   WHEN Memory LIKE '%Hybrid%' THEN 'Hybrid'
   WHEN Memory LIKE '%Flash Storage%' AND Memory
LIKE '%HDD%' THEN 'Hybrid'
   ELSE NULL
END AS 'memory_type'
FROM laptopdata;
```

```
UPDATE laptopdata
SET memory_type = CASE
        WHEN Memory LIKE '%SSD%' AND Memory LIKE
'%HDD%' THEN 'Hybrid'
   WHEN Memory LIKE '%SSD%' THEN 'SSD'
   WHEN Memory LIKE '%HDD%' THEN 'HDD'
   WHEN Memory LIKE '%Flash Storage%' THEN 'Flash
Storage'
   WHEN Memory LIKE '%Hybrid%' THEN 'Hybrid'
   WHEN Memory LIKE '%Flash Storage%' AND Memory
LIKE '%HDD%' THEN 'Hybrid'
   ELSE NULL
END;
```

```
SELECT Memory,
REGEXP_SUBSTR(SUBSTRING_INDEX(Memory,'+',1),'[0-
9]+'),
CASE WHEN Memory LIKE '%+%' THEN
REGEXP_SUBSTR(SUBSTRING_INDEX(Memory,'+',-1),'[0-
9]+') ELSE 0 END
FROM laptopdata;
```

```
UPDATE laptopdata
SET primary_storage =
REGEXP_SUBSTR(SUBSTRING_INDEX(Memory,'+',1),'[0-
9]+'),
secondary_storage = CASE WHEN Memory LIKE '%+%'
THEN
REGEXP_SUBSTR(SUBSTRING_INDEX(Memory,'+',-1),'[0-
9]+') ELSE 0 END;
```

```
SELECT
primary_storage,
CASE WHEN primary_storage <= 2 THEN
primary_storage*1024 ELSE primary_storage END,
secondary_storage,
CASE WHEN secondary_storage <= 2 THEN
secondary_storage*1024 ELSE secondary_storage END
FROM laptopdata;
```

```
UPDATE laptopdata
SET primary_storage = CASE WHEN primary_storage <= 2
THEN primary_storage*1024 ELSE primary_storage END,
secondary_storage = CASE WHEN secondary_storage <= 2
THEN secondary_storage*1024 ELSE secondary_storage
END;
```

EDA

- **head -> tail -> sample**

USE singh;

```
SELECT * FROM laptopdata
ORDER BY `index` LIMIT 5;
```

```
SELECT * FROM laptopdata
ORDER BY `index` DESC LIMIT 5;
```

```
SELECT * FROM laptopdata
ORDER BY rand() LIMIT 5;
```

2. for numerical cols

8 number summary [count, min, max, mean, std,q1,q2,q3]

missing values

outliers

```
SELECT
  COUNT(Price) OVER () AS TotalCount,
  MIN(Price) OVER () AS MinPrice,
  MAX(Price) OVER () AS MaxPrice,
  AVG(Price) OVER () AS AvgPrice,
  STD(Price) OVER () AS StdDevPrice,
  PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY
Price) OVER () AS `Q1`,
  PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY
Price) OVER () AS `Median`,
  PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY
Price) OVER () AS `Q3`
FROM laptopdata
ORDER BY `index`
LIMIT 1;
```

- **missing value**

```
SELECT COUNT(Price)
FROM laptopdata
WHERE Price IS NULL;
```

- **outliers**

```
SELECT * FROM (SELECT *,
PERCENTILE_CONT(0.25) WITHIN GROUP(ORDER BY
Price) OVER() AS 'Q1',
PERCENTILE_CONT(0.75) WITHIN GROUP(ORDER BY
Price) OVER() AS 'Q3'
FROM laptopdata) t
WHERE t.Price < t.Q1 - (1.5*(t.Q3 - t.Q1)) OR
t.Price > t.Q3 + (1.5*(t.Q3 - t.Q1));
```

- **Horizontal/Vertical Histogram**

```
SELECT t.buckets,REPEAT('*',COUNT(*)/5) FROM
(SELECT price,
CASE
        WHEN price BETWEEN 0 AND 25000 THEN '0-25K'
   WHEN price BETWEEN 25001 AND 50000 THEN '25K-
50K'
   WHEN price BETWEEN 50001 AND 75000 THEN '50K-
75K'
   WHEN price BETWEEN 75001 AND 100000 THEN '75K-
100K'
        ELSE '>100K'
END AS 'buckets'
FROM laptopdata) t
GROUP BY t.buckets;
```

- **Find the distinct company and count their frequency**

```
SELECT Company,COUNT(Company) FROM laptopdata
GROUP BY Company;
```

❖ Bivariate Analysis

```
SELECT cpu_speed,Price FROM laptopdata;

SELECT * FROM laptopdata;

SELECT Company,
SUM(CASE WHEN Touchscreen = 1 THEN 1 ELSE 0 END)
AS 'Touchscreen_yes',
SUM(CASE WHEN Touchscreen = 0 THEN 1 ELSE 0 END)
AS 'Touchscreen_no'
FROM laptopdata
GROUP BY Company;

SELECT DISTINCT cpu_brand FROM laptopdata;

SELECT Company,
SUM(CASE WHEN cpu_brand = 'Intel' THEN 1 ELSE 0
END) AS 'intel',
SUM(CASE WHEN cpu_brand = 'AMD' THEN 1 ELSE 0
END) AS 'amd',
SUM(CASE WHEN cpu_brand = 'Samsung' THEN 1 ELSE 0
END) AS 'samsung'
FROM laptopdata
GROUP BY Company;
```

❖ Categorical Numerical Bivariate analysis

```
SELECT Company,MIN(price),
MAX(price),AVG(price),STD(price)
FROM laptopdata
GROUP BY Company;
```

- **Dealing with missing values**

```
SELECT * FROM laptopdata
WHERE price IS NULL;
-- UPDATE laptopdata
-- SET price = NULL
-- WHERE `index` IN
(7,869,1148,827,865,821,1056,1043,692,1114)
```

- **replace missing values with mean of price**

```
UPDATE laptopdata
SET price = (SELECT AVG(price) FROM laptopdata)
WHERE price IS NULL;
```

- **replace missing values with mean price of corresponding company**

```
UPDATE laptopdata l1
SET price = (SELECT AVG(price) FROM laptopdata l2
WHERE
                    l2.Company = l1.Company)
WHERE price IS NULL;

SELECT * FROM laptopdata
WHERE price IS NULL;
```

- **corresponsing company + processor**

```
SELECT * FROM laptopdata;
```

- **Feature Engineering**

```
ALTER TABLE laptopdata ADD COLUMN ppi INTEGER;

UPDATE laptopdata
SET ppi = ROUND(SQRT(resolution_width*resolution_width
+ resolution_height*resolution_height)/Inches);

SELECT * FROM laptopdata
ORDER BY ppi DESC;

ALTER TABLE laptopdata ADD COLUMN screen_size
VARCHAR(255) AFTER Inches;

UPDATE laptopdata
SET screen_size =
CASE
        WHEN Inches < 14.0 THEN 'small'
    WHEN Inches >= 14.0 AND Inches < 17.0 THEN 'medium'
        ELSE 'large'
END;

SELECT screen_size,AVG(price) FROM laptopdata
GROUP BY screen_size;
```

- **One Hot Encoding**

```
SELECT gpu_brand,
CASE WHEN gpu_brand = 'Intel' THEN 1 ELSE 0 END AS
'intel',
CASE WHEN gpu_brand = 'AMD' THEN 1 ELSE 0 END AS
'amd',
CASE WHEN gpu_brand = 'nvidia' THEN 1 ELSE 0 END AS
'nvidia',
CASE WHEN gpu_brand = 'arm' THEN 1 ELSE 0 END AS
'arm'
FROM laptopdata
```

```
USE singh;
```

create a new table name of uber rides

```
CREATE TABLE uber_rides(
ride_id INTEGER PRIMARY KEY AUTO_INCREMENT,
user_id INTEGER,
cab_id INTEGER,
start_time DATETIME,
end_time DATETIME
)
```

Insert The Values

```
INSERT INTO uber_rides
(user_id,cab_id,start_time,end_time) VALUES
(1,1,'2023-12-22 08:00:00','2023-12-22 04:54:00');
```

DATETIME FUNCTIONS

1. CURR_DATE() → Current date
2. CURR_TIME() → Current time
3. NOW() → Both Date & Time

```
SELECT CURRENT_DATE();
SELECT CURRENT_TIME();
SELECT NOW();
```

```
INSERT INTO uber_rides
(user_id,cab_id,start_time,end_time) VALUES
(1,1,'2023-12-22 08:00:00',NOW());
```

Extraction Function

1. DATE() and TIME()
2. YEAR()
3. DAY() or DAYOFMONTH()
4. DAYOFWEEK()
5. DAYOFYEAR()
6. MONTH() and MONTHNAME()
7. QUARTER()
8. WEEK() or WEEKOFYEAR()
9. HOUR() → MINUTE() → SECOND()
10. LAST_DAY()

```
SELECT *,DATE(start_time) FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time)
FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time)
FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time),
MONTH(start_time)
FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time),
MONTH(start_time),
MONTHNAME(start_time)
FROM uber_rides;
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time),
MONTH(start_time),
MONTHNAME(start_time),
DAY(start_time)
FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time),
MONTH(start_time),
MONTHNAME(start_time),
DAY(start_time),
DAYOFWEEK(start_time)
FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time),
MONTH(start_time),
MONTHNAME(start_time),
DAY(start_time),
DAYOFWEEK(start_time),
DAYNAME(start_time)
FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time),
MONTH(start_time),
MONTHNAME(start_time),
DAY(start_time),
DAYOFWEEK(start_time),
DAYNAME(start_time),
QUARTER(start_time)
FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time),
MONTH(start_time),
MONTHNAME(start_time),
DAY(start_time),
DAYOFWEEK(start_time),
DAYNAME(start_time),
QUARTER(start_time),
HOUR(start_time)
FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time),
MONTH(start_time),
MONTHNAME(start_time),
DAY(start_time),
DAYOFWEEK(start_time),
DAYNAME(start_time),
QUARTER(start_time),
HOUR(start_time),
MINUTE(start_time)
FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time),
MONTH(start_time),
MONTHNAME(start_time),
DAY(start_time),
DAYOFWEEK(start_time),
DAYNAME(start_time),
QUARTER(start_time),
HOUR(start_time),
MINUTE(start_time),
SECOND(start_time)
FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time),
MONTH(start_time),
MONTHNAME(start_time),
DAY(start_time),
DAYOFWEEK(start_time),
DAYNAME(start_time),
QUARTER(start_time),
HOUR(start_time),
MINUTE(start_time),
SECOND(start_time),
DAYOFYEAR(start_time)
FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time),
MONTH(start_time),
MONTHNAME(start_time),
DAY(start_time),
DAYOFWEEK(start_time),
DAYNAME(start_time),
QUARTER(start_time),
HOUR(start_time),
MINUTE(start_time),
SECOND(start_time),
DAYOFYEAR(start_time),
WEEKOFYEAR(start_time)
FROM uber_rides;
```

```
SELECT *,DATE(start_time),
TIME(start_time),
YEAR(start_time),
MONTH(start_time),
MONTHNAME(start_time),
DAY(start_time),
DAYOFWEEK(start_time),
DAYNAME(start_time),
QUARTER(start_time),
HOUR(start_time),
MINUTE(start_time),
SECOND(start_time),
DAYOFYEAR(start_time),
WEEKOFYEAR(start_time),
LAST_DAY(start_time)
FROM uber_rides;
```

DATETIME FORMATTING

DATE FORMATTING

```
SELECT start_time,
DATE_FORMAT(start_time,'%d %b %y') FROM uber_rides;
```

start_time	DATE_FORMAT(start_time,'%d %b %y')
2023-12-22 08:00:00	22 Dec 23
2023-12-22 23:00:00	22 Dec 23
2023-12-20 21:00:00	20 Dec 23
2023-12-21 10:00:00	21 Dec 23
2024-02-09 10:00:00	09 Feb 24

TIME FORMATTING

```
SELECT start_time, DATE_FORMAT(start_time,'%l :%i %p')
FROM uber_rides;
```

start_time	DATE_FORMAT(start_time,'%l :%i %p')
2023-12-22 08:00:00	8 :00 AM
2023-12-22 23:00:00	11 :00 PM
2023-12-20 21:00:00	9 :00 PM
2023-12-21 10:00:00	10 :00 AM
2024-02-09 10:00:00	10 :00 AM

TYPE CONVERSION

1. IMPLICIT TYPE CONVERSION
2. EXPLICIT TYPE CONVERSION

```
SELECT DAYNAME(STR_TO_DATE('9-FEB hello
2024','%e-%b hello %Y'));
```

DAYNAME(STR_TO_DATE('9-FEB hello 2024','%e-%b hello %Y'))
Friday

DATETIME ARITHMETIC

1. DATEDIFF()
2. TIMEDIFF()
3. DATE_ADD() and DATE_SUB()
4. ADDTIME() and SUBTIME()

```
SELECT DATEDIFF(CURRENT_DATE,'1997-02-09');

SELECT TIMEDIFF(CURRENT_TIME,'17:00:00');
```

```
SELECT NOW(), DATE_ADD(NOW(), INTERVAL 10 YEAR);
SELECT NOW(), DATE_ADD(NOW(), INTERVAL 10 MONTH);
SELECT NOW(), DATE_ADD(NOW(), INTERVAL 10 HOUR);
SELECT NOW(), DATE_ADD(NOW(), INTERVAL 10 MINUTE);
SELECT NOW(), DATE_ADD(NOW(), INTERVAL 10 SECOND);
SELECT NOW(), DATE_ADD(NOW(), INTERVAL 10 WEEK);
SELECT NOW(), DATE_ADD(NOW(), INTERVAL 10 QUARTER);
```

SUB:→

```
SELECT NOW(), DATE_SUB(NOW(), INTERVAL 10 QUARTER);
```

```
CREATE TABLE posts(
post_id INTEGER PRIMARY KEY AUTO_INCREMENT,
user_id INTEGER,
content TEXT,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),
updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP
);
```

```
INSERT INTO posts (user_id,content) VALUES (1,'Hello World');


UPDATE posts
SET content = 'No More Hello World'
where post_id = 1;
```

DATE TIME Case Study On Flights Dataset

Flights Case Study

USE singh;

1) Find the month with most number of flights

```
SELECT MONTHNAME(date_of_journey),COUNT(*)
FROM flights
GROUP BY MONTHNAME(date_of_journey)
ORDER BY COUNT(*) DESC LIMIT 1;
```

2) Which week day has most costly flights

```
SELECT DAYNAME(date_of_journey),AVG(price) FROM
flights
GROUP BY DAYNAME(date_of_journey)
ORDER BY AVG(price) DESC LIMIT 1;
```

3) **Find number of indigo flights every month**

```
SELECT MONTHNAME(date_of_journey), COUNT(*)
FROM flights
WHERE airline = 'Indigo'
GROUP BY MONTHNAME(date_of_journey),
MONTH(date_of_journey)
ORDER BY MONTH(date_of_journey) ASC;
```

4) **Find list of all flights that depart between 10AM and 2PM from Delhi to Banglore**

```
SELECT * FROM flights
WHERE source = 'Banglore' AND
destination = 'Delhi' AND
dep_time > '10:00:00' AND dep_time < '14:00:00';
```

5) **Find the number of flights departing on weekends from Bangalore**

```
SELECT COUNT(*) FROM flights
WHERE source = 'banglore' AND
DAYNAME(date_of_journey) IN ('saturday','sunday');
```

6) Calculate the arrival time for all flights by adding the duration to the departure time.

```
ALTER TABLE flights ADD COLUMN departure DATETIME;
```

```
UPDATE flights
SET departure =
STR_TO_DATE(CONCAT(date_of_journey,'
',dep_time),'%Y-%m-%d %H:%i');
```

```
ALTER TABLE flights
ADD COLUMN duration_mins INTEGER,
ADD COLUMN arrival DATETIME;
```

```
SELECT Duration,
REPLACE(SUBSTRING_INDEX(duration,' ',1),'h','')*60 +
CASE
WHEN SUBSTRING_INDEX(duration,' ',-1) =
SUBSTRING_INDEX(duration,' ',1) THEN
0
ELSE REPLACE(SUBSTRING_INDEX(duration,' ',-1),'m','')
END AS 'mins'
FROM flights;
```

```
UPDATE flights
SET duration_mins =
CASE
WHEN duration LIKE '%h %m' THEN
SUBSTRING_INDEX(duration,'h',1)*60 +
SUBSTRING_INDEX(SUBSTRING_INDEX(duration,'',-
1),'m',1)
WHEN duration LIKE '%h' THEN
SUBSTRING_INDEX(duration,'h',1)*60
WHEN duration LIKE '%m' THEN
SUBSTRING_INDEX(duration,'m',1)
END;
```

```
UPDATE flights
SET arrival = DATE_ADD(departure,INTERVAL
duration_mins MINUTE);
```

```
SELECT TIME(arrival) FROM flights;
```

7) Calculate the arrival date for all the flights

```
SELECT DATE(arrival) FROM flights;
```

8) Find the number of flights which travel on multiple dates.

```
SELECT COUNT(*) FROM flights
WHERE DATE(departure) != DATE(arrival);
```

9) Calculate the average duration of flights between all city pairs. The answer should In xh ym format

```
SELECT source,destination,
TIME_FORMAT(SEC_TO_TIME(AVG(duration_mins)*60),'%
kh %im') AS 'avg_duration' FROM
flights
GROUP BY source,destination;
```

10) Find all flights which departed before midnight but arrived at their destination after midnight having only 0 stops.

```
SELECT * FROM flights
WHERE total_stops = 'non-stop' AND
DATE(departure) < DATE(arrival);
```

11) Find quarter wise number of flights for each airline

```
SELECT airline,QUARTER(departure),COUNT(*)
FROM flights
GROUP BY airline,QUARTER(departure);
```

12) Average time duration for flights that have 1 stop vs more than 1 stops

```
WITH temp_table AS (SELECT *,
CASE
WHEN total_stops = 'non-stop' THEN 'non-stop'
ELSE 'with stop'
END AS 'temp'
FROM flights)

SELECT temp,
TIME_FORMAT(SEC_TO_TIME(AVG(duration_mins)*60),'%
kh %im') AS 'avg_duration',
AVG(price) AS 'avg_price'
FROM temp_table
GROUP BY temp;
```

13) Find all Air India flights in a given date range originating from Delhi

```
SELECT * FROM flights
WHERE source = 'Delhi' AND
DATE(departure) BETWEEN '2019-03-01' AND '2019-03-10';
```

14) Find the longest flight of each airline

```
SELECT airline,
TIME_FORMAT(SEC_TO_TIME(MAX(duration_mins)*60),'%
kh %im') AS 'max_duration'
FROM flights
GROUP BY airline
ORDER BY MAX(duration_mins) DESC;
```

15) Find all the pair of cities having average time duration > 3 hours

```
SELECT source,destination,
TIME_FORMAT(SEC_TO_TIME(AVG(duration_mins)*60),'%
kh %im') AS 'avg_duration' FROM
flights
GROUP BY source,destination
HAVING AVG(duration_mins) > 180;
```

16) Make a weekday vs time grid showing frequency of flights from Banglore and Delhi

```
SELECT DAYNAME(departure),
      SUM(CASE WHEN HOUR(departure) BETWEEN 0
AND 5 THEN 1 ELSE 0 END) AS '12AM - 6AM',
      SUM(CASE WHEN HOUR(departure) BETWEEN 6
AND 11 THEN 1 ELSE 0 END) AS '6AM - 12PM',
      SUM(CASE WHEN HOUR(departure) BETWEEN 12
AND 17 THEN 1 ELSE 0 END) AS '12PM - 6PM',
      SUM(CASE WHEN HOUR(departure) BETWEEN 18
AND 23 THEN 1 ELSE 0 END) AS '6PM - 12PM'
FROM flights
WHERE source = 'Banglore' AND destination = 'Delhi'
GROUP BY DAYNAME(departure),
DAYOFWEEK(departure)
ORDER BY DAYOFWEEK(departure) ASC;
```

17) Make a weekday vs time grid showing avg flight price from Banglore and Delhi

```
SELECT DAYNAME(departure),
      AVG(CASE WHEN HOUR(departure) BETWEEN 0 AND
5 THEN price ELSE NULL END) AS '12AM - 6AM',
      AVG(CASE WHEN HOUR(departure) BETWEEN 6 AND
11 THEN price ELSE NULL END) AS '6AM - 12PM',
      AVG(CASE WHEN HOUR(departure) BETWEEN 12
AND 17 THEN price ELSE NULL END) AS '12PM - 6PM',
      AVG(CASE WHEN HOUR(departure) BETWEEN 18
AND 23 THEN price ELSE NULL END) AS '6PM - 12PM'
FROM flights
WHERE source = 'Banglore' AND destination = 'Delhi'
GROUP BY DAYNAME(departure),
DAYOFWEEK(departure)
ORDER BY DAYOFWEEK(departure) ASC;
```

VIEWS & USER DEFINED FUNCTIONS

USE SINGH;

create a view from this table which will hold the row only which has airline will be indigo

```
CREATE VIEW indigo AS
SELECT * FROM flights
WHERE airline = 'Indigo';
```

```
SELECT * FROM indigo;
```

```
SHOW TABLES;
```

Create a complex view by merging some table for example merge the restaurants table users table & orders table.

```
CREATE VIEW joined_ordered_data AS
SELECT
order_id,amount,r_name,name,date,delivery,delivery_rating,r
estaurant_rating
FROM orders t1
JOIN users t2
ON t1.user_id = t2.user_id
JOIN restaurants t3
ON t1.r_id = t3.r_id;
```

```
SELECT r_name , MONTHNAME(date), SUM(amount)
FROM joined_order_data
GROUP BY r_name, MONTH(date);
```

➔ IF UPDATE IN ORIGINAL TABLE THAT MEANS VIEW TABLE WILL AUTOMATICALLY BE UPDATED IN UPDATABLE VIEW.

```
UPDATE flights
SET Airline = ‘Indigo Airline’
WHERE Airline = ‘Indigo’;
```

```
UPDATE flights
SET Airline = 'Bengaluru'
WHERE Airline = 'Banglore';
```

```
DROP VIEW indigo;
```

```
CREATE VIEW indigo AS
SELECT * FROM flights
WHERE Airline = 'Indigo Airline';
```

```
UPDATE indigo
SET destination = 'Delhi'
WHERE destination = 'New Delhi';
```

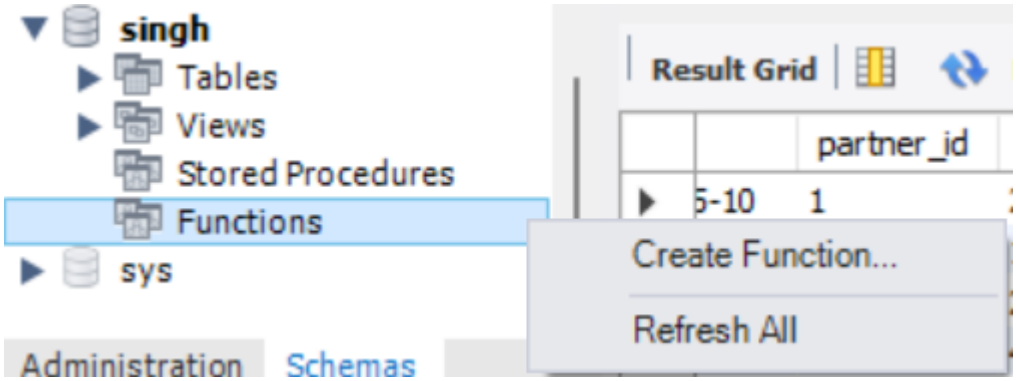
MATERIALIZED VIEW (IMPORTANT) – BUT NOT PRESENT IN MY SQL

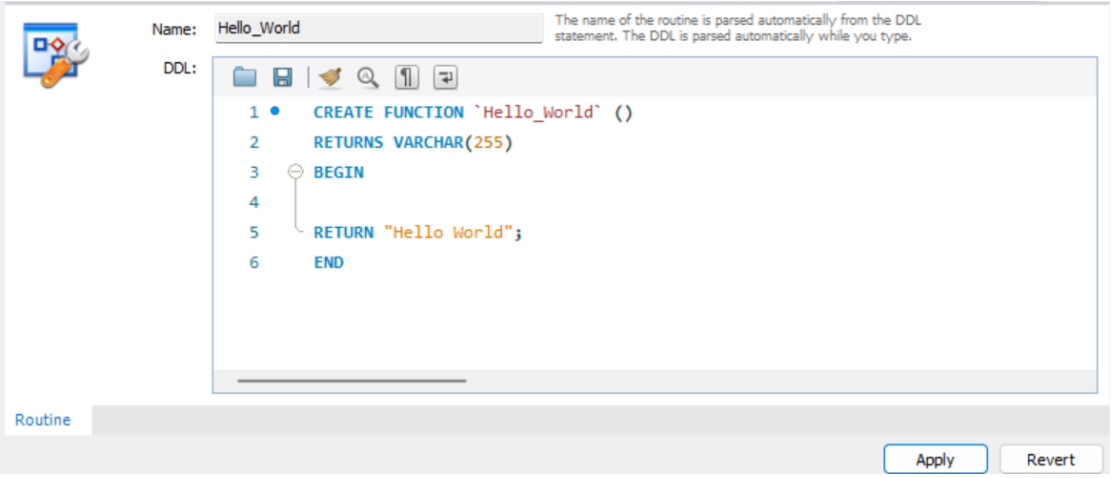
USER DEFINED FUNCTIONS

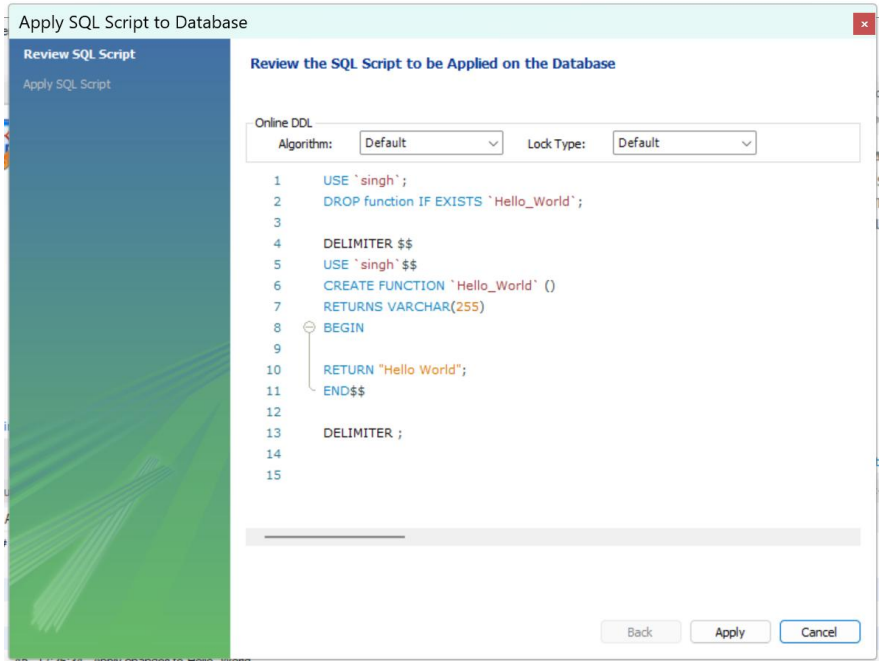
```
DELIMITER $$

CREATE FUNCTION Function_Name(
    Parameter_1 DataType,
    Parameter_2 DataType,
    Parameter_n DataType,
)
RETURNS Return_Datatype
[NOT] DETERMINISTIC
BEGIN
        Function Body
        Return Return_Value
END $$

DELIMITER ;
```





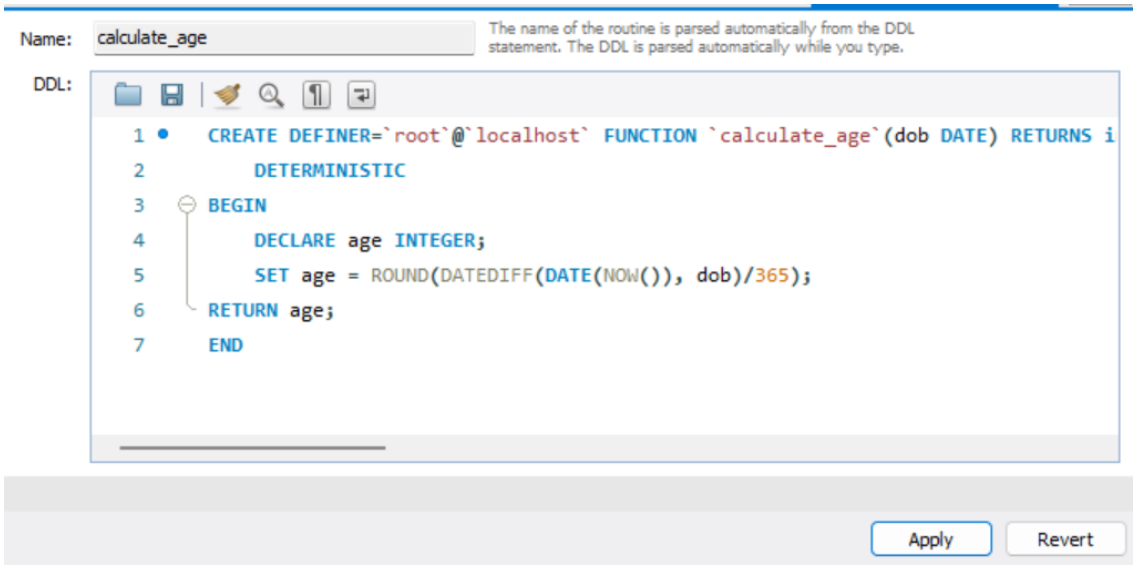


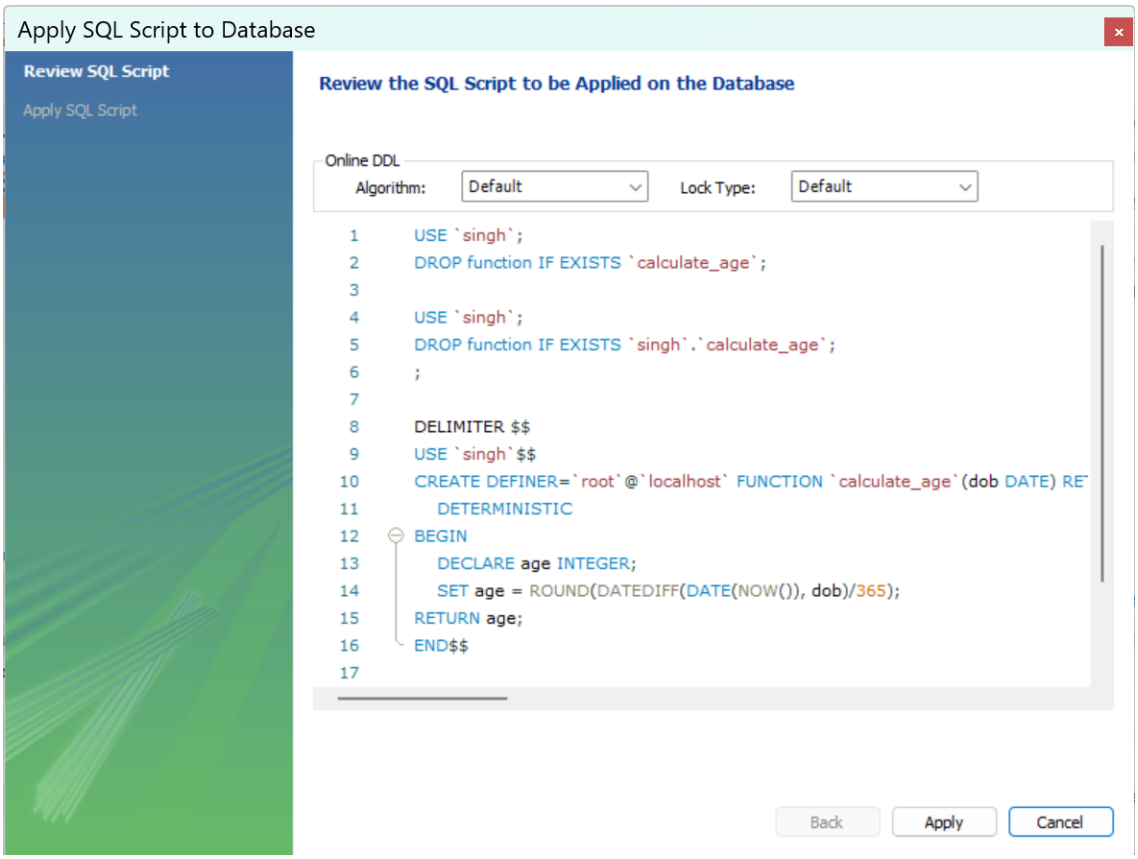
CREATE FUNCTION `Hello_World`()

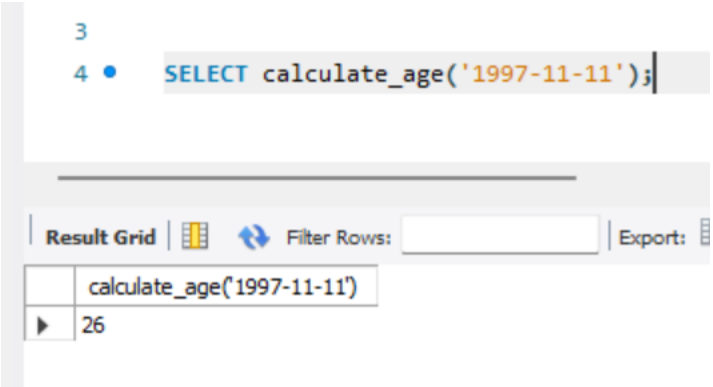
```
RETURNS VARCHAR(255)
DETERMINISTIC
BEGIN
  RETURN "Hello World";
END;
```

Parameterised Function

➔ Create a function if you give dob as input you will get age in output








```
CREATE FUNCTION proper_name (name
VARCHAR(255),gender VARCHAR(255), married
VARCHAR(255))
RETURNS VARCHAR(255)
DETERMINISTIC
BEGIN
        DECLARE title VARCHAR(255);

    IF gender = 'M' THEN
                SET title = CONCAT('Mr', ' ',name);
        ELSE
                IF married = 'Y' THEN
```

```
                SET title = CONCAT('Mrs', ' ',name);
            ELSE
                SET title = CONCAT ('Ms',' ',name);
            END IF;
        END IF;
RETURN title;
END
```

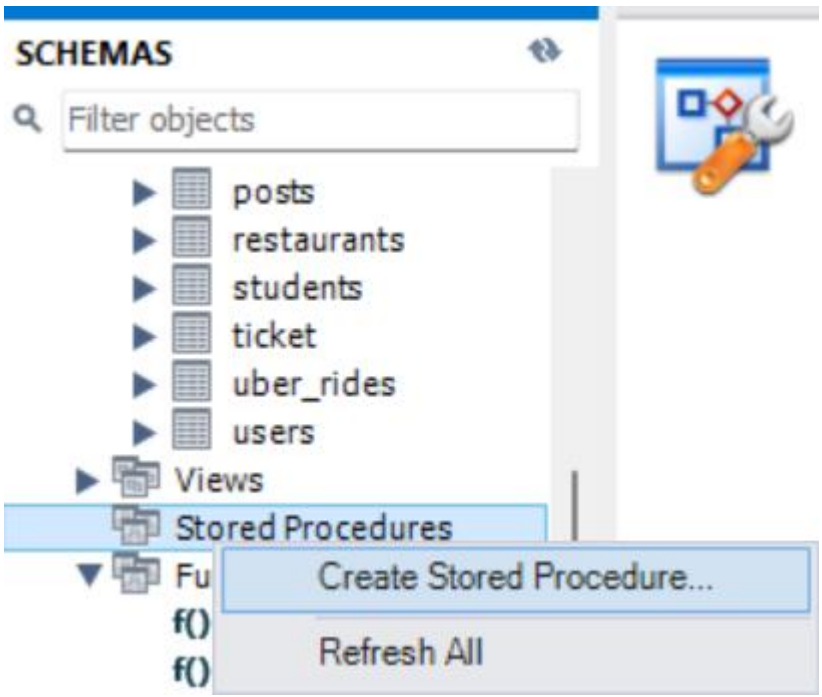
FOR CAPITALIZE FIRST LETTER

```
CREATE DEFINER=`root`@`localhost` FUNCTION
`proper_name`(name VARCHAR(255),gender VARCHAR(255),
married VARCHAR(255)) RETURNS varchar(255) CHARSET
utf8mb4
    DETERMINISTIC
BEGIN
        DECLARE title VARCHAR(255);
    SET name =
CONCAT(UPPER(LEFT(name,1)),LOWER(SUBSTRING(name,2))
);
    IF gender = 'M' THEN
            SET title = CONCAT('Mr', ' ',name);
        ELSE
            IF married = 'Y' THEN
                SET title = CONCAT('Mrs', ' ',name);
            ELSE
                SET title = CONCAT ('Ms',' ',name);
            END IF;
        END IF;
RETURN title;
END
```

```
CREATE FUNCTION flights_between (city1 VARCHAR(255),city2
VARCHAR(255))
RETURNS INTEGER
DETERMINISTIC
BEGIN

RETURN (
        SELECT COUNT(*) FROM flights
    WHERE source = city1 AND destination = city2
);
END
```

STORED PROCEDURES



Name: new_procedure

The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:

```
1  CREATE PROCEDURE `new_procedure` ()
2  BEGIN
3
4  END
5
```

Routine

Apply Revert

Name: new_procedure

The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:

```
1  CREATE PROCEDURE hello_world ()
2  BEGIN
3      SELECT "Hello World";
4  END
5
```

Routine

Apply Revert

```
5  CALL hello_world();
```

Result Grid Filter Rows:

	Hello World
▶	Hello World

```
CREATE PROCEDURE add_user(IN input_name VARCHAR (255),IN input_email
VARCHAR(255))
BEGIN
        -- check if input_email exists in users table
   DECLARE user_count INTEGER;
   SELECT COUNT(*) INTO user_count FROM users WHERE email = input_email;


   -- INSERT the new user
   IF user_count = 0 THEN
                INSERT INTO users (name,email) VALUES (input_name, input_email);
        END IF;
END
```

```
CALL add_user('Gourab','gourab@gmail.com')
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `add_user`(IN input_name
VARCHAR (255),IN input_email VARCHAR(255), OUT message VARCHAR (255))
BEGIN
        -- check if input_email exists in users table
   DECLARE user_count INTEGER;
   SELECT COUNT(*) INTO user_count FROM users WHERE email = input_email;


   -- INSERT the new user
   IF user_count = 0 THEN
                INSERT INTO users (name,email) VALUES (input_name, input_email);
      SET message = 'User_Inserted';
        ELSE
                SET message = 'Email already exists';
        END IF;
END
```

```
SET @message = '';
CALL add_user('Gourab','gourab@gmail.com',@message);


SELECT @message
```

```
CREATE PROCEDURE user_orders (IN input_email VARCHAR(255))
BEGIN
                        DECLARE id INTEGER;
        SELECT user_id INTO id FROM users WHERE email = input_email;

        SELECT * FROm orders WHERE user_id = id;
END


CALL user_orders('saurav@gmail.com');




CREATE PROCEDURE place_order (IN input_user_id INTEGER, IN input_r_id
INTEGER, IN input_f_ids VARCHAR(255), OUT total_amount INTEGER)
BEGIN
        -- insert into orders table
        DECLARE new_order_id INTEGER;
   DECLARE f_id1 INTEGER;
   DECLARE f_id2 INTEGER;

   SET f_id1 = SUBSTRING_INDEX(input_f_ids,',',1);
   SET f_id2 = SUBSTRING_INDEX(input_f_ids,',',-1);


        SELECT MAX(order_id) + 1 INTO new_order_id FROM orders;


   SELECT SUM(price) INTO total_amount FROM menu
   WHERE r_id = input_r_id AND f_id IN (f_id1,f_id2);


   INSERT INTO orders (order_id,user_id,r_id,amount,date) VALUES
   (new_order_id, input_user_id, input_r_id,total_amount, DATE(NOW()));
        -- insert into order_details table
   INSERT INTO order_details (order_id,f_id) VALUES
   (new_order_id,f_id1),(new_order_id,f_id2);
END
```

TRANSACTION

Autocommit

```
UPDATE person SET balance = 50000 WHERE id = 1;
```

Off Autocommit

```
SET autocommit = 0;


INSERT INTO person (name) VALUES ('Rishabh');


SELECT * FROM person;
```

After off the autocommit you insert the values and after inserting the value you will refresh the database connection your insertion value will be deleted becoz you did not committed the values. For this reason your changes will not impacted on memory level.

Transaction

```
START TRANSACTION;
UPDATE person SET balance = 40000 WHERE id = 1;
UPDATE person SET balance = 15000 WHERE id = 4;



START TRANSACTION;
UPDATE person SET balance = 40000 WHERE id = 1;
UPDATE person SET balance = 15000 WHERE id = 4;
COMMIT;
```

For Reverse the transaction

```
START TRANSACTION;
UPDATE person SET balance = 40000 WHERE id = 1;
UPDATE person SET balance = 15000 WHERE id = 4;
ROLLBACK;
```

```
START TRANSACTION;
SAVEPOINT A;
UPDATE person SET balance = 40000 WHERE id = 1;
SAVEPOINT B;
UPDATE person SET balance = 15000 WHERE id = 4;
ROLLBACK TO B;
```