fingerTips

SQL Practical

1) SQL DAY-1

Introduction to data, database, database management system.

2) SQL DAY-2 (CREATE DATABASE & TABLES)

```
#show databses
show databases;

#create database
create schema fashionworld;

#use database_name
use fashionworld;

#create table products
CREATE TABLE products (
  id INT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  price DECIMAL(10,2) NOT NULL,
  size VARCHAR(10),
  color VARCHAR(20),
  description VARCHAR(250)
);

#create table customers
CREATE TABLE customers (
  id INT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  phone VARCHAR(20),
  address VARCHAR(255)
);

#create table orders
CREATE TABLE orders (
  id INT PRIMARY KEY,
  customer_id INT NOT NULL,
```

```
  product_id INT NOT NULL,
  quantity INT NOT NULL,
  order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (customer_id) REFERENCES customers(id),
  FOREIGN KEY (product_id) REFERENCES products(id)
);

#Have a look at tables before inserting values!
#select * from table
select * from products;
select * from customers;
select * from orders;

#insert values into products
INSERT INTO products (id, name, price, size, color, description)
VALUES
(1, 'T-shirt', 19.99, 'M', 'Blue', 'A comfortable and stylish t-shirt'),
(2, 'Jeans', 49.99, '32x34', 'Black', 'A classic pair of black jeans'),
(3, 'Sneakers', 79.99, '10.5', 'White', 'A pair of comfortable and
stylish sneakers'),
(4, 'Sweater', 34.99, 'L', 'Gray', 'A cozy and warm sweater'),
(5, 'Dress', 59.99, 'S', 'Red', 'A beautiful and elegant dress'),
(6, 'Jacket', 99.99, 'XL', 'Green', 'A warm and stylish jacket'),
(7, 'Skirt', 29.99, 'M', 'Yellow', 'A cute and flirty skirt'),
(8, 'Blouse', 39.99, 'L', 'Pink', 'A flowy and feminine blouse'),
(9, 'Shorts', 24.99, 'S', 'Orange', 'A comfortable pair of shorts for
summer'),
(10, 'Hoodie', 49.99, 'L', 'Black', 'A cozy and casual hoodie'),
(11, 'Boots', 89.99, '9.5', 'Brown', 'A stylish pair of boots for any
occasion'),
(12, 'Sweatpants', 29.99, 'M', 'Gray', 'A comfortable and casual
pair of sweatpants'),
(13, 'Sunglasses', 19.99, NULL, 'Black', 'A cool and trendy pair of
sunglasses'),
(14, 'Scarf', 14.99, NULL, 'Purple', 'A warm and cozy scarf for the
winter'),
(15, 'Hat', 9.99, 'One size', 'Navy', 'A stylish and versatile hat for
```

any outfit'),
(16, 'Jumpsuit', 69.99, 'M', 'Black', 'A chic and trendy jumpsuit for any occasion'),
(17, 'Blazer', 79.99, 'L', 'White', 'A sophisticated and stylish blazer for work or events'),
(18, 'Sweatshirt', 39.99, 'XL', 'Pink', 'A comfortable and cozy sweatshirt for lounging'),
(19, 'Leggings', 24.99, 'S', 'Black', 'A versatile and comfortable pair of leggings'),
(20, 'Pants', 54.99, '32x30', 'Khaki', 'A classic and stylish pair of khaki pants');

#insert values into customers
INSERT INTO customers (id, name, email, phone, address)
VALUES
(1, 'John Smith', 'john.smith@gmail.com', '+1 555-123-4567', '123 Main St, Anytown, USA'),
(2, 'Jane Doe', 'jane.doe@yahoo.com', '+1 555-987-6543', '456 Maple Ave, Anytown, USA'),
(3, 'Bob Johnson', 'bob.johnson@yahoo.com', NULL, '789 Oak St, Anytown, USA'),
(4, 'Emily Williams', 'emily.williams@gmail.com', '+1 555-555-1212', '321 Elm St, Anytown, USA'),
(5, 'David Lee', 'david.lee@yahoo.com', '+1 555-555-5555', '567 Pine St, Anytown, USA'),
(6, 'Sarah Kim', 'sarah.kim@gmail.com', '+1 555-123-7890', '890 Cedar Ave, Anytown, USA'),
(7, 'Michael Chen', 'michael.chen@yahoo.com', '+1 555-999-8888', '246 Birch Blvd, Anytown, USA'),
(8, 'Jessica Brown', 'jessica.brown@yahoo.com', '+1 555-777-6666', '369 Spruce St, Anytown, USA'),
(9, 'Kevin Garcia', 'kevin.garcia@gmail.com', '+1 555-111-2222', '802 Maplewood Dr, Anytown, USA'),
(10, 'Ashley Davis', 'ashley.davis@gmail.com', NULL, '135 Walnut St, Anytown, USA');

```
#insert values into orders
INSERT INTO orders (id, customer_id, product_id, quantity, order_date)
VALUES
(1, 1, 1, 2, '2022-03-08 14:25:00'),
(2, 2, 1, 1, '2022-03-07 09:32:00'),
(3, 3, 3, 4, '2022-03-06 18:05:00'),
(4, 4, 5, 3, '2022-03-05 10:12:00'),
(5, 5, 2, 2, '2022-03-04 15:22:00'),
(6, 1, 3, 1, '2022-03-03 12:48:00'),
(7, 2, 4, 2, '2022-03-02 17:09:00'),
(8, 3, 1, 3, '2022-03-01 11:35:00'),
(9, 4, 2, 1, '2022-02-28 16:02:00'),
(10, 5, 5, 2, '2022-02-27 13:24:00'),
(11, 1, 2, 3, '2022-02-26 10:49:00'),
(12, 2, 3, 2, '2022-02-25 14:56:00'),
(13, 3, 4, 1, '2022-02-24 09:17:00'),
(14, 4, 1, 2, '2022-02-23 12:40:00'),
(15, 5, 3, 3, '2022-02-22 16:58:00');

#Have a look at tables after we are done with inserting values!
#select * from table
select * from products;
select * from customers;
select * from orders;
```

3) SQL DAY-3 (DQL, DDL, DML, DCL, TCL)

```
#DQL (SELECT)
#SELECT
select name, email from customers;

#DDL (CREATE, ALTER, TRUNCATE, DROP)
#CREATE
create table trial
(id int primary key,
column1 varchar(20) not null
);
```

```
#ALTER
alter table trial
add column2 int;

#TRUNCATE
truncate table trial;
#how the table looks like?-->All the elements will be deleted, only schema remains
Select * from trial;

#DROP
drop table trial;
#schema is deleted
select * from trial;

#DML (INSERT, UPDATE, DELLETE)
#INSERT-->alerady done

#UPDATE (pants-->shirt)
update products
set name='shirt'
where id=20;

select * from products;

#DELETE
delete from products
where id=20;

select * from products;

#TCL (ROLLBACK, COMMIT)
set autocommit =0;
delete from products
where id=19;
select * from products;
```

```
rollback;
select * from products;

delete from products
where id=19;
select * from products;
commit;
select * from products;
rollback;
select * from products;

#DCL (GRANT, REVOKE)-->theoritical
```

4) SQL DAY-4 (SQL Operators, Clauses & RegEx)

```
# Filter (where)
#Q. find details of 'Michael Chen' from csutomers table
select * from customers
where name='Michael Chen';

# Comparison Operators (<, >, =, !=, <=, >=)
# >=    Q. find names of products where price is greater than or
equal to 60
select name from products
where price>=60;

# =    Q. find product details for size 'L'
select *
from products
where size = 'L';

#Arithmetic Operators (Avg, count, min, max, sum)
#COUNT-->How many products of black color are available
select color, count(*)
from products
where color='Black';

#SUM-->Number of quantity ordered by customer with id 1
```

```
select customer_id, sum(quantity)
from orders
where customer_id=1;

#MIN-->What is minimum price of product available
select min(price)
from products;

#MAX-->What is maximum price of product available
select max(price)
from products;

#AVG-->What is the average price od products ordered
select avg(price)
from products;

# Logical Operators (or, and, not)
#OR-->Details of Jeans or Pants
select * from products
where name='Jeans' or  name='Pants';

#AND-->Is Yellow color Skirt available in products, if so what's the price?
select * from products
where name='Skirt' and color='Yellow';

# Special Operators (Between, Like, Is null, In, Not In, Distinct)
#Between-->Details of all the products available in price range from 45 to 60
Select * from products
where price between 45 and 60;

#like-->Details of warm clothes available
select * from products
where description like '%warm%';

#Is null-->Details of all thsoe customers who haven't provided
```

their phone numbers
select * from customers
where phone is null;

#IN-->Details of all the products available in size 'M', 'L' or 'XL'
select * from products
where size In ('M', 'L', 'XL');

#NOT IN-->Deatils of all the products except Black color
select * from products
where color Not In ('Black', 'Brown');

#distinct-->What all size products are available?
select distinct(size) from products;

Group by clause, having clause and Order by clause
#Group By-->What is the count and average price of all size products available
select size, count(*), avg(price) from products
group by size;

#Having-->What is the count and average price of all size products having average price > 60
select size, count(*), avg(price) from products
group by size
having avg(price)>60;

#Order By-->What is the count and average price of all size products and arrange in ascending order of price
select size, count(*), avg(price) from products
group by size
order by avg(price);

#Order By-->What is the count and average price of all size products and arrange in descending order of price
select size, count(*), avg(price) from products
group by size

```
order by avg(price) desc;

# Aliases, limit and offset
#AS-->What is the count of all size products, use total as name of resulting column
select size, count(*) as total
from products
group by size
order by count(*);

#Limit-->details of highest price product available
select * from products
order by price desc
limit 1;

#Offset-->Details of 2nd to 5th highest price product details
select * from products
order by price desc
limit 1,3;
#OR...
select * from products
order by price desc
limit 3 offset 1;

# RegEx (Regular Expressions)
select * from customers;
select * from products;

#Q1. Match beginning of string(^): Give names of customers whose name starts with ‘J’.
SELECT name FROM customers WHERE name regexp '^J';

#Q2. Match the end of a string($): Give names of customers having email id with extension '@gmail.com'.
SELECT email FROM customers WHERE email regexp '@gmail.com$';
```

#Q3.Matches any of the patterns p1, p2, or p3(p1|p2|p3): Give names containing 'Jo' or 'ee' or 'lli'.
SELECT name FROM customers WHERE name regexp 'Jo|ee|lli';

#Q4. Matches any character listed between the square brackets([abc]): Give names of colors containing vowels [aeiou]
SELECT color FROM products WHERE color regexp '[aeiou]';

5) SQL DAY-5 (SQL Joins)

```
select * from customers;
select * from products;
select * from orders;

#Inner join-->Q. How many quantities in total were ordered by customers?
select c.name, sum(o.quantity)
from customers c inner join orders o
on c.id=o.customer_id
group by c.id;
#Conclusion: As we can see there are 10 customers, but we got details of only 5 customers as details of only 5 customers is present in orders table and we have applied inner join.
#inner join or we can also write join.

#Left join-->Q. How many quantities of each sized product are ordered?
select p.size, sum(o.quantity) from
products p left join orders o
on p.id=o.product_id
group by p.size;

#Right join-->Q. Name the products which were ordered and number of quantities ordered.
select distinct(p.name)
from products p right join orders o
on o.product_id=p.id;
```

#Full Outer join/Outer join-->Q. What are the total number of products and average amount spent on each product?
#We can emulate FULL OUTER JOIN using UNION of left join & right join.

```
select p.id, count(p.id), avg(p.price)
from products p left join orders o
on p.id=o.product_id
group by p.id
union
select p.id, count(p.id), avg(p.price)
from products p right join orders o
on p.id=o.product_id
group by p.id;
```

#Cross or Cartesian join-->Join every row of a table to every row of some other table.

```
select *
from products p cross join orders o;
```

#Point to be noted here is, first id column is for products table, while other is for orders table.

#OR, we can also specify a condition on columns.

```
select *
from products p cross join orders o
where p.id=o.product_id;
```

#Self join-->Q. Name the products that are having same price.

```
SELECT A.name as product1, B.name AS product2, A.price
FROM products A, products B
WHERE A.id<> B.id
AND A.price = B.price
ORDER BY A.color;
```

#Equi join-->Q. Details of all those customers who have ordered something.
#1. Inner join can have equality (=) and other operators (like <,>,<>) in the join condition.

#2. Equi join only have an equality (=) operator in the join condition.
#3. Equi join can be an Inner join, Left Outer join, Right Outer join

```
select *
from customers c join orders o
on c.id=o.customer_id;
```

#Natural join-->Q. Join products & orderes table without applying ON condition, check results & make conclusions out of that.

```
select *
from customers c natural join orders o;
```

#Check result-->and that's the drawback of using natural join as it joins tables based on same column names. But we know 'id' in customers table represents customer's id while in orders table it represents order's id.

#Multiple join-->Combine all the 3 tables
#Q. Name the customers who have ordered atleast 6 quantities and for price>140.

```
select c.name, sum(quantity), sum(price)
from (customers c join orders o on c.id=o.customer_id)
join products p on p.id=o.product_id
group by c.id
having sum(quantity)>=6 and sum(price)>140;
```

6) Day-6 (Ranking & Analytical Functions)

#Create a database swiggy

```
use swiggy;
```

#view all tables

```
select * from users1;
select * from restaurants_1;
select* from food;
select* from menu_1;
```

```
select * from orders1;
select * from orderdetails;

#Ranking Functions (Row number, rank, dense rank)
#Q. What are the delivery ratings given by each user, arrange them in descending order?
#Row number-->
select u.name, o.delivery_rating,
row_number() over(partition by u.name order by o.delivery_rating desc) as rank_rating
from users1 u join orders1 o
on u.user_id=o.user_id;

#Rank-->
select u.name, o.delivery_rating,
rank() over(partition by u.name order by o.delivery_rating desc) as rank_rating
from users1 u join orders1 o
on u.user_id=o.user_id;
#Note: Here after two same ranks, and then rank is jumping to next like 1,1,1,4...

#Dense rank-->
select u.name, o.delivery_rating,
dense_rank() over(partition by u.name order by o.delivery_rating desc) as rank_rating
from users1 u join orders1 o
on u.user_id=o.user_id;
#Note: Now we have continuous ranks 1,2,3... as we have used dense rank

#Q. What is the amount of food ordered by each user, arrange in decending order with respect to amount?
#Row number-->
select u.name, o.amount,
row_number() over(partition by u.name order by o.amount desc) as rank_amount
```

```
from users1 u join orders1 o
on u.user_id=o.user_id;

#Rank-->
select u.name, o.amount,
rank() over(partition by u.name order by o.amount desc) as rank_amount
from users1 u join orders1 o
on u.user_id=o.user_id;

#Dense Rank-->
select u.name, o.amount,
dense_rank() over(partition by u.name order by o.amount desc) as rank_amount
from users1 u join orders1 o
on u.user_id=o.user_id;

#Q. Which cuisine is sold for highest price?
#Row number-->
select r.cuisine, m.price,
row_number() over(partition by  r.cuisine order by r.cuisine, m.price desc) as rank_price
from restaurants_1 r join menu_1 m
on r.r_id=m.r_id;

#Rank-->
select r.cuisine, m.price,
rank() over(partition by  r.cuisine order by r.cuisine, m.price desc) as rank_price
from restaurants_1 r join menu_1 m
on r.r_id=m.r_id;

#Dense rank-->
select r.cuisine, m.price,
dense_rank() over(partition by  r.cuisine order by r.cuisine, m.price desc) as rank_price
from restaurants_1 r join menu_1 m
```

```
on r.r_id=m.r_id;

#Note: Observe the difference in all the 3 queries (rank, dense rank & row number) carefully.

#Analytic Functions (Lead, lag)
#Q. For each person find whether he has spent more/less on food than previous day.
#Lag
select u.name, o.date, o.amount,
lag(amount) over(partition by u.name order by o.date) as previous_amount
from orders1 o join users1 u
on o.user_id=u.user_id;

#Q. For each person find whether he has spent more/less on food than next day.
#Lead
select u.name, o.date, o.amount,
lead(amount) over(partition by u.name order by o.date) as previous_amount
from orders1 o join users1 u
on o.user_id=u.user_id;

#Note: So, uisng lag an dlead functions we compare things happened previously or next time
```

7) Day-7 (Subqueries in SQL)

```
#Q1. Find name and email of all those users who have yahoo.com id and are customers of Swiggy but have never ordered anything from it.
select name, email
from users1
where user_id not in (select user_id from orders1)
and email regexp ('@yahoo.com');

#Q2. Find details of user who have spent maximum amount on
```

food on some particular day.

```
select *
from users1
where user_id=(select user_id from orders1 where amount=(Select max(amount) from orders1));
```

#Q3. Find name & email id of gmail user who haven gaven rating 1 on Swiggy.

```
select email
from users1
where user_id in (select user_id from orders1 where restaurant_rating = 1)
and email regexp ('@yahoo.com');
```

#Q4. Find details of highest price food ordering restaurant

```
select *
from restaurants_1
where r_id=(select r_id from menu_1 where price=(Select max(price) from menu_1));
```

#Q5. Name of food offered with minimum price

```
select f_name
from food
where f_id=(select f_id from menu_1 where price=(Select min(price) from menu_1));
```

#Q6. Menu id of lowest price food offered.

```
select menu_id
from menu_1
where price=(Select min(price) from menu_1);
```

#Q7. Which cuisine is offering food with lowest price?

```
select cuisine
from restaurants_1
where r_id=(select r_id from menu_1 where price=(Select min(price) from menu_1));
```

#Q8. Menu id of highest price food offered.
select menu_id
from menu_1 where price=(Select max(price) from menu_1);

#Q9. Details of food offered with maximum price
select *
from food
where f_id=(select f_id from menu_1 where price=(Select max(price) from menu_1));

#Q10. Find address, city, state, phone of restarant where food with minimum price is being offered.
select address, city, state, phone
from restaurants_1
where r_id=(select r_id from menu_1 where price=(Select min(price) from menu_1));

8) Day-8 (Common Table Expressions)

#Q1. What are the least delivery ratings given by each user?
with least_del as
(select u.name, o.delivery_rating,
row_number() over(partition by u.name order by o.delivery_rating) as row_
from users1 u join orders1 o
on u.user_id=o.user_id)
select * from least_del
where row_=1;

#Q2. What is the maximum amount food ordered by each user?
with max_amount as
(select u.name, o.amount,
row_number() over(partition by u.name order by o.amount desc) as row_
from users1 u join orders1 o
on u.user_id=o.user_id)
select * from max_amount

```
where row_=1;

#Q3. What are the highest restaurant ratings given by each user?
with highest_res as
(select u.name, o.delivery_rating,
row_number() over(partition by u.name order by o.restaurant_rating desc) as row_
from users1 u join orders1 o
on u.user_id=o.user_id)
select * from highest_res
where row_=1;

#Q4. What is the minimum amount food ordered by each user?
with min_amount as
(select u.name, o.amount,
row_number() over(partition by u.name order by o.amount) as row_
from users1 u join orders1 o
on u.user_id=o.user_id)
select * from min_amount
where row_=1;
```

9) Day-9 (Views)

```
# Q1. Create a view containing details of users and the number of times they were delivered food in less than 30 minutes
select u.*, count(*)
from orders1 o join users1 u
on o.user_id=u.user_id
where o.delivery_time<30
group by user_id;

create view delivery_30 as
select u.*, count(*)
from orders1 o join users1 u
on o.user_id=u.user_id
where o.delivery_time<30
```

```
group by user_id;
#Note: It can be used by Swiggy to analyze delivery timings and improve the number of counts of deliveries done in less than 30 minutes.

select * from delivery_30;

#Q2. Create a view containing details of all those users who ordered food atleast once for rupees 500 or more
select * from users1 where user_id in
(select o.user_id
from orders1 o join users1 u
on o.user_id=u.user_id
where o.amount>=500);

create view amount_500 as
select * from users1 where user_id in
(select o.user_id
from orders1 o join users1 u
on o.user_id=u.user_id
where o.amount>=500);

select * from amount_500;

#Q3. Create a view containing details of all those restaurants offering food for less than 120
select * from restaurants_1 where r_id in
(select m.r_id
from menu_1 m join restaurants_1 r
on m.r_id=r.r_id
where m.price<120);

Create view price_120 as
select * from restaurants_1 where r_id in
(select m.r_id
from menu_1 m join restaurants_1 r
on m.r_id=r.r_id
```

```
where m.price<120);

select * from price_120;

#Drop a view
drop view price_120;
```

10) Day-10 (Indexes)

```
#Q1. Create index on menu_id
CREATE INDEX idx_menu_id ON menu_1 (menu_id);

SELECT f_id, price
FROM menu_1
WHERE menu_id = 39;

#Q2. Create an index on the combination of columns "menu_id", "r_id", and "f_id"
CREATE INDEX idx_menu_restro_food ON menu_1 (menu_id, r_id, f_id);

SELECT menu_id, r_id, f_id, price
FROM menu_1
WHERE r_id = 3
AND f_id = 7
ORDER BY price DESC;
```

11) Day-11 (Stored Procedures)

```
select * from orders1;

CREATE PROCEDURE `orders_info`()
select * from orders1;

call orders_info;

#IN: Q1. Create stored procedure showing details of all those orders in which restaurants got rating 5/any other.
```

```
select *
from orders1
where restaurant_rating=5;

Create procedure rating_5 ()
select *
from orders1
where restaurant_rating=5;

call rating_5;

#Now let's say we want to make a parameter
Create procedure rating_ (IN rate int)
select *
from orders1
where orders1.restaurant_rating=rate;

call rating_(4);
call rating_(2);

#OUT: Q2. Create stored procedure showing count of orders in
which restaurants got rating 5.
select count(*)
from orders1
where restaurant_rating=5;

Create procedure total_ratings (out records int)
select count(*) into records
from orders1
where orders1.restaurant_rating=5;

call total_ratings(@records);
select @records as Total_Ratings;

#INOUT: Q3. Create stored procedure showing count of all those
orders in which restaurants got rating 4/any other.
select count(*)
```

```
from orders1
where restaurant_rating=4;

Create procedure total_rate (inout records int, in rate int)
select count(*) into records
from orders1
where orders1.restaurant_rating=rate;

call total_rate(@records, 4);
select @records as Total_rate;
```

12) Day-12 (Triggers)

```
#Before insert triggers in SQL
create trigger menu_trigger
before insert on menu_1
for each row
set new.price = new.price+100;

#Insert a record in the table and see if price is being updateed or
not
insert into menu_1(menu_id, r_id, f_id, price) values
(101,20,11,150);

#check table after inserting value
select * from menu_1;

#Note: We have inserted a new record in table menu, price=150
but since we have set a trigger so we can see new price is
updated by 100 and now we have price=250.

#After insert triggers in SQL
#we have to create a new table which stores updated price
create table final_price
(total_price int);

#Insert a record in the table and see if price is being updated or
not
```

insert into menu_1(menu_id, r_id, f_id, price) values (103,19,10,180);

#Note: We have inserted a new record in table menu, price=180 but since we have set a trigger so we can see new price is updated by 100 and now we have price=280.

```
create trigger price_trigger
after insert on menu_1
for each row
insert into final_price values(total_price);

#check table after inserting value
select * from menu_1;

#Show trigger
show triggers;

#Drop trigger in SQL
drop trigger price_trigger;
```