**--- get into database**

use SQL\_Assignment;

--- Q51

**Write an SQL query to report the name, population, and area of the big countries.**

create table if not exists world

(

name varchar(50),

continent varchar(50),

area bigint,

population bigint,

gdp bigint,

primary key(name)

);

insert into world values ('Afghanistan', 'Asia', 652230, 25500100, 20343000000),

('Albania', 'Europe', 28748, 2831741, 12960000000),

('Algeria', 'Africa', 2381741, 37100000, 188681000000),

('Andorra', 'Europe', 468, 78115, 3712000000),

('Angola', 'Africa', 1246700, 20609294, 100990000000);

select name, population, area

from world

where area >= 3000000 or population >= 25000000;

--- Q52

**Write an SQL query to report the names of the customer that are not referred by the customer with id = 2.**

create table if not exists customer\_1

(

id int,

name varchar(50),

referee\_id int,

primary key (id)

);

insert into customer\_1 values (1, 'Will', null),

(2, 'Jane', null),

(3, 'Alex', 2),

(4, 'Bill', null),

(5,' Zack', 1),

(6, 'Mark', 2);

select name

from customer\_1

where referee\_id != 2 or referee\_id is null;

--- Q53

**Write an SQL query to report all customers who never order anything.**

create table if not exists customers\_1

(

id int,

name varchar(50),

primary key(id)

);

create table if not exists Orders\_4

(

id int,

customerId int,

primary key(id)

);

insert into customers\_1 values (1, 'Joe'),

(2, 'Henry'),

(3, 'Sam'),

(4, 'Max');

insert into orders\_4 values (1, 3),

(2, 1);

select name as customers

from customers\_1

where id not in

(select customerId

from orders\_4);

--- Q54

**Write an SQL query to find the team size of each of the employees**.

create table employee\_2

(

employee\_id int,

team\_id int,

primary key(employee\_id)

);

insert into employee\_2 values (1, 8),

(2, 8),

(3, 8),

(4, 7),

(5, 9),

(6, 9);

select e1.employee\_id, count(e1.team\_id) as team\_size

from employee\_2 e1 left join employee\_2 e2

on e1.team\_id = e2.team\_id

group by e1.employee\_id;

--- Q55

**A telecommunications company wants to invest in new countries. The company intends to invest in**

**the countries where the average call duration of the calls in this country is strictly greater than the**

**global average call duration.**

**Write an SQL query to find the countries where this company can invest.**

**Return the result table in any order.**

create table person

(

id int,

name varchar(50),

phone\_number varchar(20),

primary key(id)

);

create table country

(

name varchar(50),

country\_code varchar(20),

primary key(country\_code)

);

create table calls\_1

(

caller\_id int,

callee\_id int,

duration int

);

insert into person values (3, 'Jonathan', '051-1234567'),

(12, 'Elvis', '051-7654321'),

(1, 'Moncef', '212-1234567'),

(2, 'Maroua', '212-6523651'),

(7, 'Meir', '972-1234567'),

(9, 'Rachel', '972-0011100');

insert into country values ('Peru', 51),

('Israel', 972),

('Morocco', 212),

('Germany', 49),

('Ethiopia', 251);

insert into calls\_1 values (1, 9, 33),

(2, 9, 4),

(1, 2, 59),

(3, 12, 102),

(3, 12, 330),

(12, 3, 5),

(7, 9, 13),

(7, 1, 3),

(9, 7, 1),

(1, 7, 7);

WITH TEMP\_PERSON AS (

SELECT ID, NAME,

CASE

WHEN LEFT(SUBSTR(PHONE\_NUMBER, 1,3),1) = '0' THEN RIGHT(SUBSTR(PHONE\_NUMBER, 1,3), (LENGTH(SUBSTR(PHONE\_NUMBER, 1,3))-1))

ELSE SUBSTR(PHONE\_NUMBER, 1,3) END AS COUNTRY\_CODE

FROM PERSON

)

SELECT NAME

FROM (

SELECT C.NAME, SUM(CA.DURATION) AS CALL\_DURATION, COUNT(C.COUNTRY\_CODE) AS NUMBER\_OF\_CALLS

FROM TEMP\_PERSON

JOIN COUNTRY C

ON TEMP\_PERSON.COUNTRY\_CODE = C.COUNTRY\_CODE

JOIN CALLS\_1 CA

ON TEMP\_PERSON.ID = CA.CALLER\_ID

GROUP BY C.NAME

UNION ALL

SELECT C.NAME,SUM(CA.DURATION) AS CALL\_DURATION, COUNT(C.COUNTRY\_CODE) AS NUMBER\_OF\_CALLS

FROM TEMP\_PERSON

JOIN COUNTRY C

ON TEMP\_PERSON.COUNTRY\_CODE = C.COUNTRY\_CODE

JOIN CALLS\_1 CA

ON TEMP\_PERSON.ID = CA.CALLEE\_ID

GROUP BY C.NAME

)TEMP

GROUP BY NAME

HAVING SUM(CALL\_DURATION)/SUM(NUMBER\_OF\_CALLS) > (SELECT AVG(DURATION) FROM CALLS);

--- Q56

**Write an SQL query to report the device that is first logged in for each player.**

**Return the result table in any order.**

select player\_id, device\_id

from activity

where (player\_id, event\_date) in

(select player\_id, min(event\_date)

from activity

group by player\_id);

--- Q57

**Write an SQL query to find the customer\_number for the customer who has placed the largest**

**number of orders.**

create table orders\_5

(

order\_number int,

customer\_number int,

primary key(order\_number)

);

insert into orders\_5 values(1,1),(2,2),(3,3),(4,3);

select customer\_number

from orders\_5

order by order\_number desc limit 1;

--- Q58

**Write an SQL query to report all the consecutive available seats in the cinema.**

**Return the result table ordered by seat\_id in ascending order.**

create table cinema

(

seat\_id int auto\_increment,

free bool,

primary key(seat\_id)

);

insert into cinema values (1,1), (2,0), (3,1), (4,1), (5,1);

select distinct a.seat\_id

from cinema a join cinema b

on abs(a.seat\_id - b.seat\_id) = 1

and a.free = true and b.free = true

order by a.seat\_id;

--- Q59

**Write an SQL query to report the names of all the salespersons who did not have any orders related to**

**the company with the name "RED".**

create table SalesPerson

(

sales\_id int,

name varchar(50),

salary int,

commission\_rate int,

hire\_date date,

primary key (sales\_id)

);

create table Company

(

com\_id int,

name varchar(50),

city varchar(50),

primary key (com\_id)

);

create table orders\_6

(

order\_id int,

order\_date date,

com\_id int,

sales\_id int,

amount int,

primary key (order\_id)

);

insert into SalesPerson values (1, 'John', 100000, 6, '2006-04-01'),

(2, 'Amy', 12000, 5, '2010-05-01'),

(3, 'Mark', 65000, 12, '2008-12-25'),

(4, 'Pam', 25000, 25, '2005-01-01'),

(5, 'Alex', 5000, 10, '2007-02-03');

insert into Company values (1, 'RED', 'Boston'),

(2, 'ORANGE', 'New York'),

(3, 'YELLOW', 'Boston'),

(4, 'GREEN', 'Austin');

insert into orders\_6 values (1, '2014-01-01', 3, 4, 10000),

(2, '2014-02-01', 4, 5, 5000),

(3, '2014-03-01', 1, 1, 50000),

(4, '2014-04-01', 1, 4, 25000);

SELECT

s.name

FROM

salesperson s

WHERE

s.sales\_id NOT IN (

SELECT o.sales\_id

FROM orders o LEFT JOIN company c ON o.com\_id = c.com\_id

WHERE c.name = 'RED'

);

--- Q60

**Write an SQL query to report for every three line segments whether they can form a triangle.**

create table Triangle

(

x int,

y int,

z int,

primary key (x,y,z)

);

insert into triangle values (13, 15, 30), (10, 20, 15);

select x, y, z,

case

when x < y + z and y < x + z and z < x + y then 'Yes'

else 'No'

end as triangle

from triangle;

--- Q61

**Write an SQL query to report the shortest distance between any two points from the Point table.**

create table Point

(

x int,

primary key (x)

);

insert into Point values (-1), (0), (2);

select min(abs(a.x - b.x)) as shortest

from point a join point b on a.x != b.x;

--- Q62

**Write a SQL query for a report that provides the pairs (actor\_id, director\_id) where the actor has**

**cooperated with the director at least three times.**

create table ActorDirector

(

actor\_id int,

director\_id int,

timestamp int

);

insert into ActorDirector values (1, 1, 0),

(1, 1, 1),

(1, 1, 2),

(1, 2, 3),

(1, 2, 4),

(2, 1, 5),

(2, 1, 6);

select actor\_id, director\_id

from ActorDirector

group by actor\_id, director\_id

having count(actor\_id) >=3;

--- Q63

**Write an SQL query that reports the product\_name, year, and price for each sale\_id in the Sales table.**

**Return the resulting table in any order.**

create table sales\_2

(

sale\_id int,

product\_id int,

year int,

quantity int,

price int,

primary key (sale\_id, year)

);

create table product\_3

(

product\_id int,

product\_name varchar(50),

primary key(product\_id)

);

insert into sales\_2 values (1, 100, 2008, 10, 5000),

(2, 100, 2009, 12, 5000),

(7, 200, 2011, 15, 9000);

insert into product\_3 values (100, 'Nokia'),

(200, 'Apple'),

(300, 'Samsung');

select product\_name, year, price

from sales\_2 s join product\_3 p on s.product\_id = p.product\_id;

--- Q64

**Write an SQL query that reports the average experience years of all the employees for each project,**

**rounded to 2 digits.**

create table project\_1

(

project\_id int,

employee\_id int,

primary key(project\_id, employee\_id)

);

create table employee\_3

(

employee\_id int,

name varchar(50),

experience\_years int,

primary key(employee\_id)

);

insert into project\_1 values (1, 1),

(1, 2),

(1, 3),

(2, 1),

(2, 4);

insert into employee\_3 values (1, 'Khaled', 3),

(2, 'Ali', 2),

(3, 'John', 1),

(4, 'Doe', 2);

select project\_id, avg(experience\_years)

from project\_1 p join employee\_3 e on p.employee\_id = e.employee\_id

group by project\_id;

--- Q65

**Write an SQL query that reports the best seller by total sales price, If there is a tie, report them all.**

**Return the result table in any order.**

create table Product

(

product\_id int,

product\_name varchar(50),

unit\_price int,

primary key (product\_id)

);

create table Sales

(

seller\_id int,

product\_id int,

buyer\_id int,

sale\_date date,

quantity int,

price int

);

insert into Product values (1,'S8',1000),

(2,'G4',800),

(3,'iPhone',1400);

insert into Sales values (1,1,1,'2019-01-21',2,2000),

(1,2,2,'2019-02-17',1,800),

(2,2,3,'2019-06-02',1,800),

(3,3,4,'2019-05-13',2,2800);

select seller\_id

from sales

group by seller\_id

having sum(price) = (select sum(price)

from sales

group by seller\_id

order by sum(price) desc limit 1);

--- Q66

**Write an SQL query that reports the buyers who have bought S8 but not iPhone. Note that S8 and**

**iPhone are products present in the Product table.**

create table Product

(

product\_id int,

product\_name varchar(50),

unit\_price int,

primary key (product\_id)

);

create table Sales

(

seller\_id int,

product\_id int,

buyer\_id int,

sale\_date date,

quantity int,

price int

);

insert into Product values (1,'S8',1000),

(2,'G4',800),

(3,'iPhone',1400);

insert into Sales values (1,1,1,'2019-01-21',2,2000),

(1,2,2,'2019-02-17',1,800),

(2,2,3,'2019-06-02',1,800),

(3,3,4,'2019-05-13',2,2800);

select buyer\_id

from sales

where product\_id = 1 and product\_id != 3;

--- Q67

**Write an SQL query to compute the moving average of how much the customer paid in a seven days**

**window (i.e., current day + 6 days before). average\_amount should be rounded to two decimal places.**

**Return result table ordered by visited\_on in ascending order.**

create table customer\_2

(

customer\_id int,

name varchar(50),

visited\_on date,

amount int,

primary key (customer\_id, visited\_on)

);

insert into customer\_2 values (1, 'Jhon', '2019-01-01', 100),

(2, 'Daniel', '2019-01-02', 110),

(3, 'Jade', '2019-01-03', 120),

(4, 'Khaled', '2019-01-04', 130),

(5, 'Winston', '2019-01-05', 110),

(6, 'Elvis', '2019-01-06', 140),

(7, 'Anna', '2019-01-07', 150),

(8, 'Maria', '2019-01-08', 80),

(9, 'Jaze', '2019-01-09', 110),

(1, 'Jhon', '2019-01-10', 130),

(3, 'Jade', '2019-01-10', 150);

select visited\_on, amount, average\_amount

from (select distinct visited\_on as visited\_on,

sum(amount) over (order by visited\_on range between interval '6' day preceding and current row) as amount,

round(avg(amount) over (order by visited\_on range between interval '6' day preceding and current row),2) as average\_amount

from customer\_2) as temp

where datediff(visited\_on, (select min(visited\_on) from customer\_2)) >= 6

group by visited\_on

order by visited\_on;

--- Q68

**Write an SQL query to find the total score for each gender on each day.**

**Return the result table ordered by gender and day in ascending order.**

create table scores

(

player\_name varchar(50),

gender varchar(6),

day date,

score\_points int,

primary key (gender, day)

);

insert into scores values ('Aron', 'F', '2020-01-01', 17),

('Alice', 'F', '2020-01-07', 23),

('Bajrang', 'M', '2020-01-07', 7),

('Khali', 'M', '2019-12-25', 11),

('Slaman', 'M', '2019-12-30', 13),

('Joe', 'M', '2019-12-31', 3),

('Jose', 'M', '2019-12-18', 2),

('Priya', 'F', '2019-12-31', 23),

('Priyanka', 'F', '2019-12-30', 17);

select gender, day, sum(score\_points) over (partition by gender order by day) as day

from scores

group by day, gender

order by gender;

--- Q69

**Write an SQL query to find the start and end number of continuous ranges in the table Logs.**

**Return the result table ordered by start\_id.**

create table logs

(

log\_id int,

primary key (log\_id)

);

insert into logs values (1), (2), (3), (7), (8), (10);

select start\_log.log\_id as start\_id, end\_log.log\_id as end\_id

from (select log\_id

from logs

where log\_id - 1 not in (select log\_id from logs))as start\_log,

(select log\_id

from logs

where log\_id + 1 not in (select log\_id from logs))as end\_log

where start\_log.log\_id <= end\_log.log\_id

group by start\_log.log\_id;

--- Q70

**Write an SQL query to find the number of times each student attended each exam.**

**Return the result table ordered by student\_id and subject\_name.**

create table students\_1

(

student\_id int,

student\_name varchar(50),

primary key (student\_id)

);

create table subjects

(

subject\_name varchar(30),

primary key (subject\_name)

);

create table examinations

(

student\_id int,

subject\_name varchar(30)

);

insert into students\_1 values (1, 'Alice'),

(2, 'Bob'),

(13, 'John'),

(6, 'Alex');

insert into subjects values ('Math'), ('Physics'), ('Programming');

insert into examinations values (1, 'Math'),

(1, 'Physics'),

(1, 'Programming'),

(2, 'Programming'),

(1, 'Physics'),

(1, 'Math'),

(13, 'Math'),

(13, 'Programming'),

(13, 'Physics'),

(2, 'Math'),

(1, 'Math');

select s.student\_id, student\_name,

case

when su.subject\_name in (select subject\_name from subjects) then su.subject\_name

when su.subject\_name not in (select subject\_name from examinations) then 'maths'

end as subject\_name,

case

when s.student\_id in (select student\_id from examinations) then count(e.subject\_name)

when s.student\_id not in (select student\_id from examinations) then 0

end as attended\_exams

from (students\_1 s join subjects su on 1 = 1) left join Examinations e

on (s.student\_id, su.subject\_name) = (e.student\_id, e.subject\_name)

group by student\_id, su.subject\_name, s.student\_name

order by s.student\_id, su.subject\_name;

--- Q71

**Write an SQL query to find employee\_id of all employees that directly or indirectly report their work to**

**the head of the company.**

create table employees\_1

(

employee\_id int,

employee\_name varchar(50),

manager\_id int,

primary key (employee\_id)

);

insert into employees\_1 values (1, 'Boss', 1),

(3, 'Alice', 3),

(2, 'Bob', 1),

(4, 'Daniel', 2),

(7, 'Luis', 4),

(8, 'Jhon', 3),

(9, 'Angela', 8),

(77, 'Robert', 1);

select employee\_id as employee\_id

from employees\_1

where manager\_id in (

select employee\_id

from employees\_1

where manager\_id in(

select employee\_id

from employees\_1

where manager\_id = 1))

and employee\_id <> 1;

--- Q72

**Write an SQL query to find for each month and country, the number of transactions and their total**

**amount, the number of approved transactions and their total amount.**

create table transactions

(

id int,

country varchar(50),

state enum("approved", "declined"),

amount int,

trans\_date date,

primary key (id)

);

insert into transactions values (121, 'US', 'approved', 1000, '2018-12-18'),

(122, 'US', 'declined', 2000, '2018-12-19'),

(123, 'US', 'approved', 2000, '2019-01-01'),

(124, 'DE', 'approved', 2000, '2019-01-07');

select date\_format(trans\_date, '%Y-%m') as month,

country, count(\*) as trans\_count,

sum(if(state = 'approved', 1, 0)) as approved\_count,

sum(amount) as trans\_total\_amount,

sum(if(state = 'approved', amount, 0)) as approved\_total\_amount

from transactions

group by date\_format(trans\_date, '%y-%m'), country;

--- Q73

**Write an SQL query to find the average daily percentage of posts that got removed after being**

**reported as spam, rounded to 2 decimal places.**

create table actions

(

user\_id int,

post\_id int,

action\_date date,

action enum ('view', 'like', 'reaction', 'comment', 'report', 'share'),

extra varchar(50)

);

create table removals

(

post\_id int,

remove\_date date,

primary key (post\_id)

);

insert into actions values (1, 1, '2019-07-01', 'view', 'null'),

(1, 1, '2019-07-01', 'like', 'null'),

(1, 1, '2019-07-01', 'share', 'null'),

(2, 2, '2019-07-04', 'view', 'null'),

(2, 2, '2019-07-04', 'report', 'spam'),

(3, 4, '2019-07-04', 'view', 'null'),

(3, 4, '2019-07-04', 'report', 'spam'),

(4, 3, '2019-07-02', 'view', 'null'),

(4, 3, '2019-07-02', 'report', 'spam'),

(5, 2, '2019-07-03', 'view', 'null'),

(5, 2, '2019-07-03', 'report', 'racism'),

(5, 5, '2019-07-03', 'view', 'null'),

(5, 5, '2019-07-03', 'report', 'racism');

insert into removals values (2, '2019-07-20'), (3, '2019-07-18');

--- Q74

--- Same as Q43

--- Q75

--- Same as Q43

--- Q76

**Write an SQL query to find the salaries of the employees after applying taxes. Round the salary to the**

**nearest integer.**

create table salaries

(

company\_id int,

employee\_id int,

employee\_name varchar(50),

salary int,

primary key (company\_id, employee\_id)

);

insert into salaries values (1, 1, 'Tony', 2000),

(1, 2, 'Pronub', 21300),

(1, 3, 'Tyrrox', 10800),

(2, 1, 'Pam', 300),

(2, 7, 'Bassem', 450),

(2, 9, 'Hermione', 700),

(3, 7, 'Bocaben', 100),

(3, 2, 'Ognjen', 2200),

(3, 13, 'Nyan Cat', 3300),

(3, 15, 'Morning Cat', 7777);

select s.company\_id, s.employee\_id, s.employee\_name, (

round(case

when (select max(t.salary) from Salaries t where t.company\_id = s.company\_id) < 1000 then s.salary

when (select max(t.salary) from Salaries t where t.company\_id = s.company\_id) between 1000 and 10000 then s.salary \* (1 - 0.24)

else s.salary \* (1 - 0.49)

end, 0)

) as salary from Salaries s;

--- Q77

**Write an SQL query to report the difference between the number of apples and oranges sold each day.**

**Return the result table ordered by sale\_date.**

create table variables

(

name varchar(50),

value int,

primary key (name)

);

create table expressions

(

left\_operand varchar(30),

operator enum ('<', '>', '='),

right\_operand varchar(30),

primary key (left\_operand, operator, right\_operand)

);

insert into variables values ('x', 66), ('y', 77);

insert into expressions values ('x', '>', 'y'),

('x', '<', 'y'),

('x', '=', 'y'),

('y', '>', 'x'),

('y', '<', 'x'),

('x', '=', 'x');

select e.left\_operand, e.operator, e.right\_operand,

case e.operator

when '>' then if(v1.value > v2.value, 'true', 'false')

when '<' then if(v1.value < v2.value, 'true', 'false')

else if(v1.value = v2.value, 'true', 'false')

end

as value

from Expressions e

left join Variables v1 on v1.name = e.left\_operand

left join Variables v2 on v2.name = e.right\_operand;

--- Q78

--- Samae as Q55

--- Q79

**Write an SQL query to:**

**● Find the name of the user who has rated the greatest number of movies. In case of a tie,**

**return the lexicographically smaller user name.**

**● Find the movie name with the highest average rating in February 2020. In case of a tie, return**

**the lexicographically smaller movie name.**

create table employee\_data

(

employee\_id int,

name varchar(50),

month int,

salary int

);

insert into employee\_data values (12228, 'Rose', 15, 1968),

(33645, 'Angela', 1, 3443),

(45692, 'Frank', 17, 1608),

(56118, 'Patrick', 7, 1345),

(59725, 'Lisa', 11, 2330),

(74197, 'Kimberly', 16, 4372),

(78454, 'Bonnie', 8, 1771),

(83565, 'Michael', 6, 2017),

(98607, 'Todd', 5, 3396),

(99989, 'Joe', 9, 3573);

select name

from employee\_data

order by name;

--- Q80

**Write an SQL query to find the countries where this company can invest.**

create table user\_transactions

(

transaction\_id int,

product\_id int,

spend decimal(20,2),

transaction\_date datetime

);

insert into user\_transactions values (1341, 123424, 1500.60, '2019-12-31 12:00:00'),

(1423, 123424, 1000.20, '2020-12-31 12:00:00'),

(1623, 123424, 1246.44, '2021-12-31 12:00:00'),

(1322, 123424, 2145.32, '2022-12-31 12:00:00');

WITH yearly\_spend AS (

SELECT

EXTRACT(YEAR FROM transaction\_date) AS year,

product\_id,

spend AS curr\_year\_spend

FROM user\_transactions

),

yearly\_variance AS (

SELECT

\*,

LAG(curr\_year\_spend, 1) OVER (

PARTITION BY product\_id

ORDER BY product\_id, year) AS prev\_year\_spend

FROM yearly\_spend)

SELECT

year,

product\_id,

curr\_year\_spend,

prev\_year\_spend,

ROUND(100 \* (curr\_year\_spend - prev\_year\_spend)/ prev\_year\_spend,2) AS yoy\_rate

FROM yearly\_variance;

--- Q81

**Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by**

**the last three characters of each name. If two or more students both have names ending in the same**

**last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.**

create table inventory

(

item\_id int,

item\_type varchar(20),

item\_category varchar(20),

square\_footage decimal (20,2)

);

insert into inventory values (1374, 'prime\_eligible', 'mini refrigerator', 68.00),

(4245, 'not\_prime', 'standing lamp', 26.40),

(2452, 'prime\_eligible', 'television', 85.00),

(3255, 'not\_prime', 'side table', 22.60),

(1672, 'prime\_eligible', 'laptop', 8.50);

WITH summary AS (

SELECT

item\_type,

SUM(square\_footage) AS total\_sqft,

COUNT(\*) AS item\_count

FROM inventory

GROUP BY item\_type

),

prime\_items AS (

SELECT

DISTINCT item\_type,

total\_sqft,

TRUNC(500000/total\_sqft,0) AS prime\_item\_combo,

(TRUNC(500000/total\_sqft,0) \* item\_count) AS prime\_item\_count

FROM summary

WHERE item\_type = 'prime\_eligible'

),

non\_prime\_items AS (

SELECT

DISTINCT item\_type,

total\_sqft,

TRUNC(

(500000 - (SELECT prime\_item\_combo \* total\_sqft FROM prime\_items))

/ total\_sqft, 0) \* item\_count AS non\_prime\_item\_count

FROM summary

WHERE item\_type = 'not\_prime')

SELECT

item\_type,

prime\_item\_count AS item\_count

FROM prime\_items

UNION ALL

SELECT

item\_type,

non\_prime\_item\_count AS item\_count

FROM non\_prime\_items;

--- Q82

**Write a query that prints a list of employee names (i.e.: the name attribute) from the Employee table in**

**alphabetical order**

create table user\_actions

(

user\_id int,

event\_id int,

event\_type enum ("sign-in", "like", "comment"),

event\_date datetime

);

insert into user\_actions values (445, 7765, 'sign-in', '2022-05-31 12:00:00'),

(742, 6458, 'sign-in', '2022-06-03 12:00:00'),

(445, 3634, 'like', '2022-06-05 12:00:00'),

(742, 1374, 'comment', '2022-06-05 12:00:00'),

(648, 3124, 'like', '2022-06-18 12:00:00');

SELECT

EXTRACT(MONTH FROM curr\_month.event\_date) AS mth,

COUNT(DISTINCT curr\_month.user\_id) AS monthly\_active\_users

FROM user\_actions AS curr\_month

WHERE EXISTS (

SELECT last\_month.user\_id

FROM user\_actions AS last\_month

WHERE last\_month.user\_id = curr\_month.user\_id

AND EXTRACT(MONTH FROM last\_month.event\_date) =

EXTRACT(MONTH FROM curr\_month.event\_date - interval '1 month')

)

AND EXTRACT(MONTH FROM curr\_month.event\_date) = 7

AND EXTRACT(YEAR FROM curr\_month.event\_date) = 2022

GROUP BY EXTRACT(MONTH FROM curr\_month.event\_date);

--- Q83

**Write a query that prints a list of employee names (i.e.: the name attribute) for employees in**

**Employee having a salary greater than $2000 per month who have been employees for less than 10**

**months. Sort your result by ascending employee\_id**

create table search\_frequency

(

searches int,

num\_users int

);

insert into search\_frequency values (1, 2), (2, 2), (3, 3), (4, 1);

WITH searches\_expanded AS (

SELECT searches

FROM search\_frequency

GROUP BY

searches,

GENERATE\_SERIES(1, num\_users))

SELECT

ROUND(PERCENTILE\_CONT(0.50) WITHIN GROUP (

ORDER BY searches)::DECIMAL, 1) AS median

FROM searches\_expanded;

--- Q84

**Write a query identifying the type of each record in the TRIANGLES table using its three side lengths.**

**Output one of the following statements for each record in the table:**

**● Equilateral: It's a triangle with sides of equal length.**

**● Isosceles: It's a triangle with sides of equal length.**

**● Scalene: It's a triangle with sides of differing lengths.**

**● Not A Triangle: The given values of A, B, and C don't form a triangle.**

create table advertiser

(

user\_id varchar(30),

status varchar(30)

);

insert into advertiser values ('bing', 'NEW'),

('yahoo', 'NEW'),

('alibaba', 'EXISTING');

create table daily\_pay

(

user\_id varchar(30),

paid decimal(20,2)

);

insert into daily\_pay values ('yahoo', 45.00),

('alibaba', 100.00),

('target', 13.00);

WITH payment\_status AS (

SELECT

advertiser.user\_id,

advertiser.status,

payment.paid

FROM advertiser

LEFT JOIN daily\_pay AS payment

ON advertiser.user\_id = payment.user\_id

UNION

SELECT

payment.user\_id,

advertiser.status,

payment.paid

FROM daily\_pay AS payment

LEFT JOIN advertiser

ON advertiser.user\_id = payment.user\_id

)

SELECT

user\_id,

CASE WHEN paid IS NULL THEN 'CHURN'

WHEN status != 'CHURN' AND paid IS NOT NULL THEN 'EXISTING'

WHEN status = 'CHURN' AND paid IS NOT NULL THEN 'RESURRECT'

WHEN status IS NULL THEN 'NEW'

END AS new\_status

FROM payment\_status

ORDER BY user\_id;

--- Q85

**Write a query to obtain the year-on-year growth rate for the total spend of each product for**

**each year.**

**Output the year (in ascending order) partitioned by product id, current year's spend, previous year's**

**spend and year-on-year growth rate (percentage rounded to 2 decimal places).**

create table server\_utilization

(

server\_id int,

status\_time timestamp,

session\_status varchar(15)

);

insert into server\_utilization values (1, '2022-08-02 10:00:00', 'start'),

(1, '2022-08-04 10:00:00', 'stop'),

(2, '2022-08-17 10:00:00', 'start'),

(2, '2022-08-24 10:00:00', 'stop');

--- Q86

**Write a SQL query to find the number of prime and non-prime items that can be stored in the 500,000**

**square feet warehouse. Output the item type and number of items to be stocked.**

create table transactions\_1

(

transaction\_id int,

merchant\_id int,

credit\_card\_id int,

amount int,

transaction\_timestamp datetime

);

insert into transactions\_1 values (1, 101, 1, 100, '2022-09-25 12:00:00'),

(2, 101, 1, 100, '2022-09-25 12:08:00'),

(3, 101, 1, 100, '2022-09-25 12:28:00'),

(4, 102, 2, 300, '2022-09-25 12:00:00'),

(6, 102, 2, 400, '2022-09-25 14:00:00');

--- Q87

**Write a query to obtain the active user retention in July 2022. Output the month (in numerical format 1, 2, 3) and the**

**number of monthly active users (MAUs).**

create table orders\_7

(

order\_id int,

customer\_id int,

trip\_id int,

status enum ('completed successfully', 'completed incorrectly', 'never received'),

order\_timestamp timestamp

);

insert into orders\_7 values (727424, 8472, 100463, 'completed successfully', '2022-06-05 09:12:00'),

(242513, 2341, 100482, 'completed incorrectly', '2022-06-05 14:40:00'),

(141367, 1314, 100362, 'completed incorrectly', '2022-06-07 15:03:00'),

(582193, 5421, 100657, 'never received', '2022-07-07 15:22:00'),

(253613, 1314, 100213, 'completed successfully', '2022-06-12 13:43:00');

create table trips

(

dasher\_id int,

trip\_id int,

estimated\_delivery\_timestamp timestamp,

actual\_delivery\_timestamp timestamp

);

insert into trips values (101, 100463, '2022-06-05 09:42:00', '2022-06-05 09:38:00'),

(102, 100482, '2022-06-05 15:10:00', '2022-06-05 15:46:00'),

(101, 100362, '2022-06-07 15:33:00', '2022-06-07 16:45:00'),

(102, 100657, '2022-07-07 15:52:00', null),

(103, 100213, '2022-06-12 14:13:00', '2022-06-12 14:10:00');

create table customers\_2

(

customer\_id int,

signup\_timestamp timestamp

);

insert into customers\_2 values (8472, '2022-05-30 00:00:00'),

(2341, '2022-06-01 00:00:00'),

(1314, '2022-06-03 00:00:00'),

(1435, '2022-06-05 00:00:00'),

(5421, '2022-06-07 00:00:00');

--- Q88

--- same as Q68

--- Q89

--- same as Q55

--- Q90

**Write a query that calculates the total time that the fleet of servers was running. The output should be**

**in units of full days.**

create table numbers

(

num int,

frequency int,

primary key (num)

);

insert into numbers values (0,7), (1,1), (2,3), (3,1);

select avg(Num) as median from (

select n1.Num from Numbers n1 join Numbers n2 on n1.Num >= n2.Num

group by n1.Num

having

sum(n2.Frequency) >= (select sum(Frequency) from Numbers) / 2

and

sum(n2.Frequency) - avg(n1.Frequency) <= (select sum(Frequency) from Numbers) / 2

) med;

--- Q91

**Sometimes, payment transactions are repeated by accident; it could be due to user error, API failure or**

**a retry error that causes a credit card to be charged twice.**

**Using the transactions table, identify any payments made at the same merchant with the same credit**

**card for the same amount within 10 minutes of each other. Count such repeated payments.**

create table salary

(

id int,

employee\_id int,

amount int,

pay\_date date,

primary key(id)

);

create table employee\_4

(

employee\_id int,

department\_id int,

primary key(employee\_id)

);

insert into salary values (1, 1, 9000, '2017-03-31'),

(2, 2, 6000, '2017-03-31'),

(3, 3, 10000, '2017-03-31'),

(4, 1, 7000, '2017-02-28'),

(5, 2, 6000, '2017-02-28'),

(6, 3, 8000, '2017-02-28');

insert into employee\_4 values (1,1), (2,2), (3,2);

select department\_salary.pay\_month, department\_id,

case

when department\_avg > company\_avg then 'higher'

when department\_avg < company\_avg then 'lower'

else 'same'

end as comparison

from (

select department\_id, avg(amount) as department\_avg, date\_format(pay\_date, '%Y-%m') as pay\_month

from salary join employee\_4 on salary.employee\_id = employee\_4.employee\_id

group by department\_id, pay\_month

) as department\_salary

join (

select avg(amount) as company\_avg, date\_format(pay\_date, '%Y-%m') as pay\_month

from salary group by date\_format(pay\_date, '%Y-%m')

) as company\_salary

on department\_salary.pay\_month = company\_salary.pay\_month;

--- Q92

**Write an SQL query to report for each install date, the number of players that installed the game on**

**that day, and the day one retention.**

create table Activity\_2

(

player\_id int,

device\_id int,

event\_date date,

games\_played int,

primary key(player\_id, event\_date)

);

insert into Activity\_2 values (1, 2, '2016-03-01', 5),

(1, 2, '2016-03-02', 6),

(2, 3, '2017-06-25', 1),

(3, 1, '2016-03-01', 0),

(3, 4, '2016-07-03', 5);

select a1.event\_date as install\_dt, count(a1.player\_id) as installs, round(count(a3.player\_id) / count(a1.player\_id), 2) as Day1\_retention

from Activity\_2 a1 left join Activity\_2 a2

on a1.player\_id = a2.player\_id and a1.event\_date > a2.event\_date

left join Activity\_2 a3

on a1.player\_id = a3.player\_id and datediff(a3.event\_date, a1.event\_date) = 1

where a2.event\_date is null

group by a1.event\_date;

--- Q93

--- same as Q50

--- Q94

**Write an SQL query to report the students (student\_id, student\_name) being quiet in all exams. Do not**

**return the student who has never taken any exam**

create table student\_1

(

student\_id int,

student\_name varchar(50),

primary key(student\_id)

);

create table exam

(

exam\_id int,

student\_id int,

score int,

primary key(exam\_id, student\_id)

);

insert into student\_1 values (1, 'Daniel'),

(2, 'Jade'),

(3, 'Stella'),

(4, 'Jonathan'),

(5, 'Will');

insert into exam values (10, 1, 70),

(10, 2, 80),

(10, 3, 90),

(20, 1, 80),

(30, 1, 70),

(30, 3, 80),

(30, 4, 90),

(40, 1, 60),

(40, 2, 70),

(40, 4, 80);

select student\_id, student\_name

from

(

select distinct student\_id

from Exam

where student\_id not in

(

select distinct student\_id

from Exam e left join

( -- highest and lowest scores

select exam\_id, max(score) maxs, min(score) mins

from Exam

group by exam\_id

) t

using(exam\_id)

where score = maxs or score = mins

)

) t

left join Student\_1

using(student\_id)

order by student\_id;

--- Q95

--- same as Q94

--- Q96

**Write a query to output the user id, song id, and cumulative count of song plays as of 4 August 2022**

**sorted in descending order.**

create table songs\_history

(

history\_id integer,

user\_id integer,

song\_id integer,

song\_plays integer

);

insert into songs\_history values (10011, 777, 1238, 11), (12452, 695, 4520, 1);

create table songs\_weekly

(

user\_id integer,

song\_id integer,

listen\_time datetime

);

insert into songs\_weekly values (777, 1238, '2022-08-01 12:00:00'),

(695, 4520, '2022-08-04 08:00:00'),

(125, 9630, '2022-08-04 16:00:00'),

(695, 9852, '2022-08-07 12:00:00');

WITH history AS (

SELECT user\_id, song\_id, song\_plays

FROM songs\_history

UNION ALL

SELECT user\_id, song\_id, COUNT(song\_id) AS song\_plays

FROM songs\_weekly

WHERE listen\_time <= '08/04/2022 23:59:59'

GROUP BY user\_id, song\_id

)

SELECT user\_id, song\_id, SUM(song\_plays) AS song\_count

FROM history

GROUP BY user\_id, song\_id

ORDER BY song\_count DESC;

--- Q97

**Write a query to find the confirmation rate of users who confirmed their signups with text messages.**

**Round the result to 2 decimal places.**

create table emails

(

email\_id integer,

user\_id integer,

signup\_date datetime

);

insert into emails values (125, 7771, '2022-06-14 00:00:00'),

(236, 6950, '2022-07-01 00:00:00'),

(433, 1052, '2022-07-09 00:00:00');

create table texts

(

text\_id integer,

email\_id integer,

signup\_action varchar(20)

);

insert into texts values (6878, 125, 'Confirmed'),

(6920, 236, 'Not Confirmed'),

(6994, 236, 'Confirmed');

WITH rate AS (

SELECT

user\_id,

CASE WHEN texts.email\_id IS NOT NULL THEN 1

ELSE 0 END AS signup

FROM emails

LEFT JOIN texts

ON emails.email\_id = texts.email\_id

AND signup\_action = 'Confirmed')

SELECT ROUND(SUM(signup)::DECIMAL / COUNT(user\_id), 2) AS confirmation\_rate

FROM rate;

--- Q98

**The table below contains information about tweets over a given period of time. Calculate the 3-day**

**rolling average of tweets published by each user for each date that a tweet was posted. Output the**

**user id, tweet date, and rolling averages rounded to 2 decimal places.**

create table tweets

(

tweet\_id integer,

user\_id integer,

tweet\_date timestamp

);

insert into tweets values (214252, 111, '2022-06-01 12:00:00'),

(739252, 111, '2022-06-01 12:00:00'),

(846402, 111, '2022-06-02 12:00:00'),

(241425, 254, '2022-06-02 12:00:00'),

(137374, 111, '2022-06-04 12:00:00');

SELECT

user\_id,

tweet\_date,

ROUND(

AVG(tweet\_num) OVER (

PARTITION BY user\_id

ORDER BY user\_id, tweet\_date

ROWS BETWEEN 2 PRECEDING AND CURRENT ROW), 2)

AS rolling\_avg\_3d

FROM (

SELECT

user\_id,

tweet\_date,

COUNT(DISTINCT tweet\_id) AS tweet\_num

FROM tweets

GROUP BY user\_id, tweet\_date) AS tweet\_count;

--- Q99

**Write a query to obtain a breakdown of the time spent**

**sending vs. opening snaps (as a percentage of total time spent on these activities) for each age**

**group.**

create table activities

(

activity\_id integer,

user\_id integer,

activity\_type enum ('send', 'open', 'chat'),

time\_spent decimal(20,2),

activity\_date datetime

);

insert into activities values (7274, 123, 'open', 4.50, '2022-06-22 12:00:00'),

(2425, 123, 'send', 3.50, '2022-06-22 12:00:00'),

(1413, 456, 'send', 5.67, '2022-06-23 12:00:00'),

(1414, 789, 'chat', 11.00, '2022-06-25 12:00:00'),

(2536, 456, 'open', 3.00, '2022-06-25 12:00:00');

create table age\_breakdown

(

user\_id integer,

age\_bucket enum ('21-25', '26-30', '31-35')

);

insert into age\_breakdown values (123, '31-35'),

(456, '26-30'),

(789, '21-25');

WITH snaps\_statistics AS (

SELECT

age.age\_bucket,

SUM(CASE WHEN activities.activity\_type = 'send'

THEN activities.time\_spent ELSE 0 END) AS send\_timespent,

SUM(CASE WHEN activities.activity\_type = 'open'

THEN activities.time\_spent ELSE 0 END) AS open\_timespent,

SUM(activities.time\_spent) AS total\_timespent

FROM activities

INNER JOIN age\_breakdown AS age

ON activities.user\_id = age.user\_id

WHERE activities.activity\_type IN ('send', 'open')

GROUP BY age.age\_bucket)

SELECT

age\_bucket,

ROUND(100.0 \* send\_timespent / total\_timespent, 2) AS send\_perc,

ROUND(100.0 \* open\_timespent / total\_timespent, 2) AS open\_perc

FROM snaps\_statistics;

--- Q100

**Write a query to return the IDs of these LinkedIn power creators in ascending order.**

**Assumptions:**

**● A person can work at multiple companies.**

**● In the case of multiple companies, use the one with largest follower base.**

create table personal\_profiles

(

profile\_id integer,

name varchar(50),

followers integer

);

insert into personal\_profiles values (1, 'Nick Singh', 92000),

(2, 'Zach Wilson', 199000),

(3, 'Daliana Liu', 171000),

(4, 'Ravit Jain', 107000), (5, 'Vin Vashishta', 139000),

(6, 'Susan Wojcicki', 39000);

create table employee\_company

(

personal\_profile\_id integer,

company\_id integer

);

insert into employee\_company values (1, 4), (1, 9), (2, 2), (3, 1), (4, 3), (5, 6), (6, 5);

create table company\_pages

(

company\_id integer,

name varchar(50),

followers integer

);

insert into company\_pages values (1, 'The Data Science Podcast', 8000),

(2, 'Airbnb', 700000),

(3, 'The Ravit Show', 6000),

(4, 'DataLemur', 200),

(5, 'YouTube', 16000000),

(6, 'DataScience.Vin', 4500),

(9, 'Ace The Data Science Interview', 4479);