Q1. Query all columns for all American cities in the CITY table with populations larger than 100000. The CountryCode for America is USA. The CITY table is described as follows:

Ans:- -- SELECT \* FROM test.`city table - sheet1`;

SELECT \* FROM test.`city table - sheet1`

WHERE POPULATION > 100000 AND COUNTRYCODE = "USA";

Q2. Query the NAME field for all American cities in the CITY table with populations larger than 120000. The CountryCode for America is USA. The CITY table is described as follows

Ans:- -- SELECT \* FROM test.`city table - sheet1`;

SELECT NAME FROM test.`city table - sheet1`

WHERE POPULATION > 120000 AND COUNTRYCODE = "USA";

Q3. Query all columns (attributes) for every row in the CITY table. The CITY table is described as follows:

Ans:-

SELECT \* FROM test.`city table - sheet1`;

select \* from city

Q4. Query all columns for a city in CITY with the ID 1661. The CITY table is described as follows:

Ans:-

SELECT \* FROM test.`city table - sheet1`;

select \* from test.`city table - sheet1` where ID = 1661;

Q5. Query all attributes of every Japanese city in the CITY table. The COUNTRYCODE for Japan is JPN. The CITY table is described as follows:

Ans:-

select \* from test.`city table - sheet1` where COUNTRYCODE = 'JPN' ;

Q6. Query the names of all the Japanese cities in the CITY table. The COUNTRYCODE for Japan is JPN. The CITY table is described as follows

Ans:-

SELECT NAME FROM test.`city table - sheet1` WHERE COUNTRYCODE="JPN";

Q7. Query a list of CITY and STATE from the STATION table. The STATION table is described as follows: where LAT\_N is the northern latitude and LONG\_W is the western longitude.

Ans:-

SELECT City,State FROM test.`stationdata - sheet1` ORDER BY City, State;

Q8. Query a list of CITY names from STATION for cities that have an even ID number. Print the results in any order, but exclude duplicates from the answer. The STATION table is described as follows:

Ans:-

select distinct city from test.`stationdata - sheet1` where id%2=0

Q9. Find the difference between the total number of CITY entries in the table and the number of distinct CITY entries in the table. The STATION table is described as follows:

Ans:-

SELECT COUNT(CITY) - COUNT(DISTINCT CITY) FROM test.`stationdata - sheet1`;

Q10. Query the two cities in STATION with the shortest and longest CITY names, as well as their

respective lengths (i.e.: number of characters in the name). If there is more than one smallest or

largest city, choose the one that comes first when ordered alphabetically.

The STATION table is described as follows:

Q11. Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) from . Your result cannot contain duplicates. Input Format The STATION table is described as follows:

Ans:-

SELECT DISTINCT CITY FROM STATION WHERE CITY LIKE 'A%' OR CITY LIKE 'E%' OR CITY LIKE 'I%' OR CITY LIKE 'O%' OR CITY LIKE 'U%';

Q12. Query the list of CITY names ending with vowels (a, e, i, o, u) from STATION. Your result cannot contain duplicates. Input Format The STATION table is described as follows

Ans:-

SELECT DISTINCT(CITY) FROM STATION WHERE CITY LIKE '%A' OR CITY LIKE '%E' OR CITY LIKE '%I' OR CITY LIKE '%O' OR CITY LIKE '%U';

Q13. Query the list of CITY names from STATION that do not start with vowels. Your result cannot contain duplicates. Input Format The STATION table is described as follows:

ANS:-

SELECT DISTINCT CITY FROM STATION WHERE CITY LIKE 'A%A' OR CITY LIKE 'E%E' OR CITY LIKE 'I%I' OR CITY LIKE 'O%O' OR CITY LIKE 'U%U';

Q14. Query the list of CITY names from STATION that do not end with vowels. Your result cannot contain duplicates

Ans:-

SELECT DISTINCT CITY FROM STATION

WHERE lcase(CITY) NOT LIKE '%a'

AND lcase(CITY) NOT LIKE '%e'

AND lcase(CITY) NOT LIKE '%i'

AND lcase(CITY) NOT LIKE '%o'

AND lcase(CITY) NOT LIKE '%u'

ORDER BY CITY;

Q15. Query the list of CITY names from STATION that either do not start with vowels or do not end with vowels. Your result cannot contain duplicates.

Ans:

select distinct CITY from STATION where CITY not regexp '^[aeiou]' or city not regexp '[aeiou]$';

Q16. Query the list of CITY names from STATION that do not start with vowels and do not end with vowels. Your result cannot contain duplicates.

Ans:

select distinct CITY from STATION where CITY not regexp '^[aeiou]' AND city not regexp '[aeiou]$';

# Q17.

Table: Product

|  |  |
| --- | --- |
| Column Name | Type |
| product\_id | int |
| product\_name | varchar |
| unit\_price | int |

product\_id is the primary key of this table.

Each row of this table indicates the name and the price of each product. Table: Sales

|  |  |
| --- | --- |
| Column Name | Type |
| seller\_id | int |
| product\_id | int |
| buyer\_id | int |
| sale\_date | date |
| quantity | int |
| price | int |

This table has no primary key, it can have repeated rows product\_id is a foreign key to the Product table.

Each row of this table contains some information about one sale.

Write an SQL query that reports the products that were only sold in the ﬁrst quarter of 2019. That is, between 2019-01-01 and 2019-03-31 inclusive.

Return the result table in any order.

The query result format is in the following example.

Input:

Product table:

|  |  |  |
| --- | --- | --- |
| product\_id | product\_name | unit\_price |
| 1 | S8 | 1000 |
| 2 | G4 | 800 |
| 3 | iPhone | 1400 |

Sales table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| seller\_id | product\_id | buyer\_id | sale\_date | quantity | price |
| 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 |

Output:

|  |  |
| --- | --- |
| product\_id | product\_name |
| 1 | S8 |

Explanation:

The product with id 1 was only sold in the spring of 2019.

The product with id 2 was sold in the spring of 2019 but was also sold after the spring of 2019. The product with id 3 was sold after spring 2019.

We return only product 1 as it is the product that was only sold in the spring of 2019.

Ans:-

SELECT product\_id, product\_name

FROM PRODUCT

WHERE product\_id IN (SELECT product\_id

FROM SALES

WHERE sale\_date BETWEEN

'2019-01-01' AND '2019-03-31')

**Q18**.

Table: Views

|  |  |
| --- | --- |
| Column Name | Type |
| article\_id | int |
| author\_id | int |
| viewer\_id | int |
| view\_date | date |

There is no primary key for this table, it may have duplicate rows.

Each row of this table indicates that some viewer viewed an article (written by some author) on some date.

Note that equal author\_id and viewer\_id indicate the same person.

Write an SQL query to ﬁnd all the authors that viewed at least one of their own articles. Return the result table sorted by id in ascending order.

The query result format is in the following example.

Input: Views table:

|  |  |  |  |
| --- | --- | --- | --- |
| article\_id | author\_id | viewer\_id | view\_date |
| 1 | 3 | 5 | 2019-08-01 |
| 1 | 3 | 6 | 2019-08-02 |
| 2 | 7 | 7 | 2019-08-01 |
| 2 | 7 | 6 | 2019-08-02 |
| 4 | 7 | 1 | 2019-07-22 |
| 3 | 4 | 4 | 2019-07-21 |
| 3 | 4 | 4 | 2019-07-21 |

Output:

|  |
| --- |
| id |
| 4 |
| 7 |

Ans:-

SELECT DISTINCT author\_id AS id

FROM Views

WHERE author\_id = viewer\_id

ORDER BY id ASC;

**Q.19**

Table: Delivery

|  |  |
| --- | --- |
| Column Name | Type |
| delivery\_id | int |
| customer\_id | int |
| order\_date | date |
| customer\_pref\_delivery\_date | date |

delivery\_id is the primary key of this table.

The table holds information about food delivery to customers that make orders at some date and specify a preferred delivery date (on the same order date or after it).

If the customer's preferred delivery date is the same as the order date, then the order is called immediately; otherwise, it is called scheduled.

Write an SQL query to ﬁnd the percentage of immediate orders in the table, rounded to 2 decimal places.

The query result format is in the following example.

Input: Delivery table:

|  |  |  |  |
| --- | --- | --- | --- |
| delivery\_id | customer\_id | order\_date | customer\_pref\_ delivery\_date |
| 1 | 1 | 2019-08-01 | 2019-08-02 |
| 2 | 5 | 2019-08-02 | 2019-08-02 |
| 3 | 1 | 2019-08-11 | 2019-08-11 |
| 4 | 3 | 2019-08-24 | 2019-08-26 |
| 5 | 4 | 2019-08-21 | 2019-08-22 |
| 6 | 2 | 2019-08-11 | 2019-08-13 |

Output:

33.33

immediate\_percentage

Explanation: The orders with delivery id 2 and 3 are immediate while the others are scheduled.

Ans

SELECT ROUND(COUNT(CASE WHEN order\_date = customer\_pref\_delivery\_date THEN 1 END) \* 100.0 / COUNT(\*), 2) AS immediate\_percentage

FROM Delivery;

# Q20.

Table: Ads

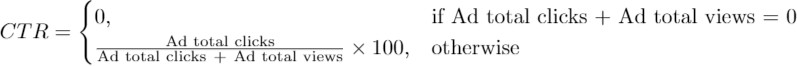
|  |  |
| --- | --- |
| Column Name | Type |
| ad\_id | int |
| user\_id | int |
| action | enum |

(ad\_id, user\_id) is the primary key for this table.

Each row of this table contains the ID of an Ad, the ID of a user, and the action taken by this user regarding this Ad.

The action column is an ENUM type of ('Clicked', 'Viewed', 'Ignored').

A company is running Ads and wants to calculate the performance of each Ad. Performance of the Ad is measured using Click-Through Rate (CTR) where:



Write an SQL query to ﬁnd the ctr of each Ad. Round ctr to two decimal points.

Return the result table ordered by ctr in descending order and by ad\_id in ascending order in case of a tie.

The query result format is in the following example.

Input:

Ads table:

|  |  |  |
| --- | --- | --- |
| ad\_id | user\_id | action |
| 1 | 1 | Clicked |
| 2 | 2 | Clicked |
| 3 | 3 | Viewed |
| 5 | 5 | Ignored |
| 1 | 7 | Ignored |
| 2 | 7 | Viewed |
| 3 | 5 | Clicked |
| 1 | 4 | Viewed |
| 2 | 11 | Viewed |
| 1 | 2 | Clicked |

Output:

|  |  |
| --- | --- |
| ad\_id | ctr |
| 1 | 66.67 |
| 3 | 50 |
| 2 | 33.33 |
| 5 | 0 |

Explanation:

for ad\_id = 1, ctr = (2/(2+1)) \* 100 = 66.67 for ad\_id = 2, ctr = (1/(1+2)) \* 100 = 33.33 for ad\_id = 3, ctr = (1/(1+1)) \* 100 = 50.00

for ad\_id = 5, ctr = 0.00, Note that ad\_id = 5 has no clicks or views. Note that we do not care

Ans:-

SELECT ad\_id, ROUND(IFNULL(SUM(CASE WHEN action = 'Clicked' THEN 1 ELSE 0 END) / NULLIF(SUM(CASE WHEN action IN ('Clicked', 'Viewed') THEN 1 ELSE 0 END), 0), 0) \* 100, 2) AS ctr

FROM Ads

GROUP BY ad\_id

ORDER BY ctr DESC, ad\_id ASC;

# Q21.

Table: Employee

|  |  |
| --- | --- |
| Column Name | Type |
| employee\_id | int |
| team\_id | int |

employee\_id is the primary key for this table.

Each row of this table contains the ID of each employee and their respective team.

Write an SQL query to ﬁnd the team size of each of the employees. Return result table in any order.

The query result format is in the following example.

Input:

Employee Table:

|  |  |
| --- | --- |
| employee\_id | team\_id |
| 1 | 8 |
| 2 | 8 |
| 3 | 8 |
| 4 | 7 |
| 5 | 9 |
| 6 | 9 |

Output:

|  |  |
| --- | --- |
| employee\_id | team\_size |
| 1 | 3 |
| 2 | 3 |
| 3 | 3 |
| 4 | 1 |
| 5 | 2 |
| 6 | 2 |

Explanation:

Employees with Id 1,2,3 are part of a team with team\_id = 8. An employee with Id 4 is part of a team with team\_id = 7.

Employees with Id 5,6 are part of a team with team\_id = 9.

Ans:-

SELECT e.employee\_id, COUNT(\*) AS team\_size

FROM Employee

JOIN (

SELECT team\_id, COUNT(\*) AS team\_count

FROM Employee

GROUP BY team\_id

) ON e.team\_id = t.team\_id

GROUP BY e.employee\_id;

# Q22.

Table: Countries

|  |  |
| --- | --- |
| Column Name | Type |
| country\_id | int |
| country\_name | varchar |

country\_id is the primary key for this table.

Each row of this table contains the ID and the name of one country.

Table: Weather

|  |  |
| --- | --- |
| Column Name | Type |
| country\_id | int |
| weather\_state | int |
| day | date |

(country\_id, day) is the primary key for this table.

Each row of this table indicates the weather state in a country for one day.

Write an SQL query to ﬁnd the type of weather in each country for November 2019. The type of weather is:

* Cold if the average weather\_state is less than or equal 15,
* Hot if the average weather\_state is greater than or equal to 25, and
* Warm otherwise. Return result table in any order.

The query result format is in the following example.

Input: Countries table:

|  |  |
| --- | --- |
| country\_id | country\_name |
| 2 | USA |
| 3 | Australia |
| 7 | Peru |
| 5 | China |
| 8 | Morocco |
| 9 | Spain |

Weather table:

|  |  |  |
| --- | --- | --- |
| country\_id | weather\_state | day |
| 2 | 15 | 2019-11-01 |
| 2 | 12 | 2019-10-28 |

|  |  |  |
| --- | --- | --- |
| 2 | 12 | 2019-10-27 |
| 3 | -2 | 2019-11-10 |
| 3 | 0 | 2019-11-11 |
| 3 | 3 | 2019-11-12 |
| 5 | 16 | 2019-11-07 |
| 5 | 18 | 2019-11-09 |
| 5 | 21 | 2019-11-23 |
| 7 | 25 | 2019-11-28 |
| 7 | 22 | 2019-12-01 |
| 7 | 20 | 2019-12-02 |
| 8 | 25 | 2019-11-05 |
| 8 | 27 | 2019-11-15 |
| 8 | 31 | 2019-11-25 |
| 9 | 7 | 2019-10-23 |
| 9 | 3 | 2019-12-23 |

Output:

|  |  |
| --- | --- |
| country\_name | weather\_type |
| USA | Cold |
| Australia | Cold |
| Peru | Hot |
| Morocco | Hot |
| China | Warm |

Explanation:

Average weather\_state in the USA in November is (15) / 1 = 15 so the weather type is Cold.

Average weather\_state in Australia in November is (-2 + 0 + 3) / 3 = 0.333 so the weather type is Cold. Average weather\_state in Peru in November is (25) / 1 = 25 so the weather type is Hot.

The average weather\_state in China in November is (16 + 18 + 21) / 3 = 18.333 so the weather type is warm.

Average weather\_state in Morocco in November is (25 + 27 + 31) / 3 = 27.667 so the weather type is Hot.

We know nothing about the average weather\_state in Spain in November so we do not include it in the result table.

Ans:-

SELECT c.country\_name,

CASE

WHEN AVG(w.weather\_state) <= 15 THEN 'Cold'

WHEN AVG(w.weather\_state) >= 25 THEN 'Hot'

ELSE 'Warm'

END AS weather\_type

FROM Countries c

JOIN Weather w

ON c.country\_id = w.country\_id

WHERE w.day >= '2019-11-01' AND w.day < '2019-12-01'

GROUP BY c.country\_id

**Q24**.

--Table: Activity

**Ans:**

create table if not exists Activity

(

player\_id int,

device\_id int,

event\_date date,

games\_played INT DEFAULT 0,

constraint pk PRIMARY KEY (player\_id, event\_date)

);

INSERT into Activity values (1,2,'2016-03-01',5), (1,2,'2016-05-02',6), (2,3,'2017-06-25',1), (3,1,'2016-03-02',0), (3,4,'2018-07-03',5);

--Write an SQL query to report the first login date for each player.

Return the result table in any order.

select player\_id,event\_date as first\_login,

row\_number() over(partition by player\_id) as row\_num

from Activity;

select

tmp.player\_id,tmp.event\_date as first\_login

from (select \*,

row\_number() over(partition by player\_id ) as row\_num

from Activity) tmp

where tmp.row\_num = 1;

**Q25**.

--Table: Activity

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

--Return the result table in any order.

**Ans:**

select

tmp.player\_id,tmp.device\_id

from (select \*,

row\_number() over(partition by player\_id ) as row\_num

from Activity) tmp

where tmp.row\_num = 1;

**Q26**.

--Table: Products

**Ans:**

create table if not exists Products

(

product\_id int,

product\_name VARCHAR(50),

product\_category VARCHAR(50),

constraint pk PRIMARY KEY (product\_id)

);

insert into Products values (1,'Leetcode Solutions','Book'), (2,'Jewels of Stringology','Book'), (3,'HP','Laptop'), (4,'Lenovo','Laptop'), (5,'Leetcode Kit','T-shirt');

select \* from Products;

--Table: Orders

create table if not exists Orders

(

product\_id int,

order\_date date,

unit int,

constraint fk FOREIGN KEY (product\_id) REFERENCES Products(product\_id)

);

insert into Orders values (1,'2020-02-05',60), (1,'2020-02-10',70), (2,'2020-01-18',30), (2,'2020-02-11',80), (3,'2020-02-17',2), (3,'2020-02-24',3), (4,'2020-03-01',20), (4,'2020-03-04',30), (4,'2020-03-04',60), (5,'2020-02-25',50), (5,'2020-02-27',50), (5,'2020-03-01',50);

select \* from Orders;

--Write an SQL query to get the names of products that have at least 100 units ordered in February 2020 and their amount.

--Return result table in any order.

select a.product\_name, sum(unit) as unit

from Products a

left join Orders b

on a.product\_id = b.product\_id

where b.order\_date between '2020-02-01' and '2020-02-29'

group by a.product\_id

having sum(unit) >= 100;

**Q27**.

--Table: Users

**Ans:**

create table if not exists Users

(

user\_id int,

name varchar(50),

mail varchar(50),

constraint pk PRIMARY KEY (user\_id)

);

Insert into Users VALUES (1,'Winston','winston@leetcode.com'),(2,'Jonathan','jonathanisgreat'),(3,'Annabelle','bella@leetcode.com'),(4,'Sally','sally.come@leetcode.com'),(5,'Marwan','quarz#2020@leetcode.com'),(6,'David','david69@gmail.com'),(7,'Shapiro','.shapo@leetcode.com');

select \* from Users;

--Write an SQL query to find the users who have valid emails.

--A valid e-mail has a prefix name and a domain where:

--● The prefix name is a string that may contain letters (upper or lower case), digits, underscore

--'\_', period '.', and/or dash '-'. The prefix name must start with a letter.

--● The domain is '@leetcode.com'.

--Return the result table in any order.

SELECT \*

FROM Users

WHERE REGEXP\_LIKE(mail, '^[a-zA-Z][a-zA-Z0-9\\_\.\-]\*@leetcode.com');

**Q28.**

--Table: Customers

**Ans:**

create table if not exists Customers

(

customer\_id int,

name varchar(50),

country varchar(50),

constraint pk PRIMARY KEY (customer\_id)

);

insert into Customers VALUES (1,'Winston','USA'),(2,'Jonathan','Peru'),(3,'Moustafa','Egypt');

select \* from Customers;

--Table: Product

create table if not exists Product

(

product\_id int,

description varchar(255),

price int,

constraint pk PRIMARY KEY (product\_id)

);

insert into Product values (10,'LC Phone',300),(20,'LC T-Shirt',10),(30,'LC Book',45),(40,'LC Keychain',2);

select \* from Product;

--Table: Orders

create table if not exists Orders

(

order\_id int,

customer\_id int,

product\_id int,

order\_date DATE,

quantity int,

constraint pk PRIMARY KEY (order\_id)

-- constraint fk FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id),

-- constraint fk FOREIGN KEY (product\_id) REFERENCES Product(product\_id)

);

insert into Orders VALUES (1,1,10,'2020-06-10',1),(2,1,20,'2020-07-01',1),(3,1,30,'2020-07-08',2),(4,2,10,'2020-06-15',2),(5,2,40,'2020-07-01',10),(6,3,20,'2020-06-24',2),(7,3,30,'2020-06-25',2),(9,3,30,'2020-05-08',3);

select \* from Orders;

--Write an SQL query to report the customer\_id and customer\_name of customers who have spent at

--least $100 in each month of June and July 2020.

--Return the result table in any order.

select o.customer\_id, c.name

from Customers c, Product p, Orders o

where c.customer\_id = o.customer\_id and p.product\_id = o.product\_id

group by o.customer\_id

having

(

sum(case when o.order\_date like '2020-06%' then o.quantity\*p.price else 0 end) >= 100

and

sum(case when o.order\_date like '2020-07%' then o.quantity\*p.price else 0 end) >= 100

);

**Q29**.

--Table: TVProgram

**Ans:**

create table if not exists TVProgram

(

program\_date date,

content\_id int,

channel varchar(50),

constraint pk PRIMARY KEY (program\_date, content\_id)

);

insert into TVProgram VALUES ('2020-06-10 08:00',1,'LC-Channel'),('2020-05-11 12:00',2,'LC-Channel'),('2020-05-12 12:00',3,'LC-Channel'),('2020-05-13 14:00',4,'Disney Ch'),('2020-06-18 14:00',4,'Disney Ch'),('2020-07-15 16:00',5,'Disney Ch');

select \* from TVProgram;

--Table: Content

create table if not exists Content

(

content\_id varchar(50),

title varchar(50),

Kids\_content enum('Y','N'),

content\_type varchar(50),

constraint pk PRIMARY KEY (content\_id)

);

insert into Content VALUES (1,'Leetcode Movie','N','Movies'),(2,'Alg. for Kids','Y','Series'),(3,'Database Sols','N','Series'),(4,'Aladdin','Y','Movies'),(5,'Cinderella','Y','Movies');

select \* from Content;

--Write an SQL query to report the distinct titles of the kid-friendly movies streamed in June 2020.

--Return the result table in any order.

SELECT DISTINCT title

FROM Content ctt

INNER JOIN TVProgram tv

ON ctt.content\_id = tv.content\_id

WHERE content\_type = 'Movies'

AND Kids\_content = 'Y'

AND program\_date BETWEEN '2020-06-01' AND '2020-06-30';

**Q30.**

--Table: NPV

**Ans:**

create table if not exists NPV

(

id int,

year int,

npv int,

constraint pk PRIMARY KEY (id, year)

);

Insert into NPV VALUES (1,2018,100),(7,2020,30),(13,2019,40),(1,2019,113),(2,2008,121),(3,2009,12),(11,2020,99),(7,2019,0);

select \* from NPV;

--Table: Queries

create table if not exists Queries

(

id int,

year int,

constraint pk PRIMARY KEY (id, year)

);

insert into Queries VALUES (1, 2019),(2,2008),(3,2009),(7,2018),(7,2019),(7,2020),(13,2019);

select \* from Queries;

--Write an SQL query to find the npv of each query of the Queries table.

--Return the result table in any order.

SELECT q.id, q.year, COALESCE(n.npv,0) AS npv

FROM Queries q

LEFT JOIN NPV n

ON q.id = n.id AND q.year=n.year;

**Q32**.

--Table: Employees

**Ans:**

create table if not exists Employees

(

id int,

name varchar(50),

constraint pk PRIMARY KEY (id)

);

insert into Employees VALUES (1,'Alice'),(7,'Bob'),(11,'Meir'),(90,'Winston'),(3,'Jonathan');

select \* from Employees;

--Table: EmployeeUNI

create table if not exists EmployeeUNI

(

id int,

unique\_id int,

constraint pk PRIMARY KEY (id, unique\_id)

);

insert into EmployeeUNI VALUES (3,1),(11,2),(90,3);

select \* from EmployeeUNI;

--Write an SQL query to show the unique ID of each user, If a user does not have a unique ID replace just show null.

--Return the result table in any order.

select unique\_id, name

from Employees

left join EmployeeUNI

on if (Employees.id = EmployeeUNI.id, Employees.id, null);

**Q33**.

--Table: Users

create table if not exists Users

(

id int,

name VARCHAR(50),

constraint pk PRIMARY KEY (id)

);

insert into Users VALUES (1,'Alice'),(2,'Bob'),(3,'Alex'),(4,'Donald'),(7,'Lee'),(13,'Jonathan'),(19,'Elvis');

select \* from Users;

--Table: Rides

create table if not exists Rides

(

id int,

user\_id int,

distance int,

constraint pk PRIMARY KEY (id),

constraint fk FOREIGN KEY (user\_id) REFERENCES Users(id)

);

insert into Rides VALUES (1,1,120), (2,2,317), (3,3,222), (4,7,100), (5,13,312), (6,19,50), (7,7,120), (8,19,400), (9,7,230);

select \* from Rides;

--Write an SQL query to report the distance travelled by each user.

--Return the result table ordered by travelled\_distance in descending order, if two or more users travelled the same distance, order them by their name in ascending order.

select name, sum(ifnull(distance, 0)) as travelled\_distance

from Rides r

right join Users u

on r.user\_id = u.id

group by name

order by 2 desc,1 asc;

**Q34**.

--Table: Products

**Ans:**

create table if not exists Products

(

product\_id int,

product\_name varchar(50),

product\_category VARCHAR(50),

constraint pk PRIMARY KEY (product\_id)

);

insert into Products VALUES (1,'Leetcode Solutions','Book'), (2,'Jewels of Stringology','Book'), (3,'HP','Laptop'), (4,'Lenovo','Laptop'), (5,'Leetcode Kit','T-shirt');

select \* from Products;

--Table: Orders

create table if not exists Orders

(

product\_id int,

order\_date date,

unit int

);

insert into Orders values (1,'2020-02-05',60), (1,'2020-02-10',70), (2,'2020-01-18',30), (2,'2020-02-11',80), (3,'2020-02-17',2), (3,'2020-02-24',3), (4,'2020-03-01',20), (4,'2020-03-04',30), (4,'2020-03-04',60), (5,'2020-02-25',50), (5,'2020-02-27',50), (5,'2020-03-01',50);

select \* from Orders;

--Write an SQL query to get the names of products that have at least 100 units ordered in February 2020 and their amount.

--Return result table in any order.

select a.product\_name, sum(unit) as unit

from Products a

left join Orders b

on a.product\_id = b.product\_id

where b.order\_date between '2020-02-01' and '2020-02-29'

group by a.product\_id

having sum(unit) >= 100;

**Q35**.

--Table: Movies

**Ans:**

create table if not exists Movies

(

movie\_id int,

title varchar(50),

constraint pk PRIMARY KEY (movie\_id)

);

insert into Movies VALUES (1,'Avengers'),(2,'Frozen 2'),(3,'Joker');

select \* from Movies;

--Table: Users

create table if not exists Users

(

user\_id int,

name varchar(50),

constraint pk PRIMARY KEY (user\_id)

);

insert into Users VALUES (1,'Daniel'),(2,'Monica'),(3,'Maria'),(4,'James');

select \* from Users;

--MovieRating table:

create table if not exists MovieRating

(

movie\_id int,

user\_id int,

rating int,

created\_at date,

constraint pk PRIMARY KEY (movie\_id, user\_id)

);

insert into MovieRating VALUES (1,1,3,'2020-01-12'),(1,2,4,'2020-02-11'),(1,3,2,'2020-02-12'),(1,4,1,'2020-01-01'),(2,1,5,'2020-02-17'),(2,2,2,'2020-02-01'),(2,3,2,'2020-03-01'),(3,1,3,'2020-02-22'),(3,2,4,'2020-02-25');

select \* from MovieRating;

--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 username.

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

--the lexicographically smaller movie name.

SELECT user\_name AS results FROM

(

SELECT a.name AS user\_name, COUNT(\*) AS counts FROM MovieRating AS b

JOIN Users AS a

on a.user\_id = b.user\_id

GROUP BY b.user\_id

ORDER BY counts DESC, user\_name ASC LIMIT 1

) first\_query

UNION

SELECT movie\_name AS results FROM

(

SELECT c.title AS movie\_name, AVG(d.rating) AS rate FROM MovieRating AS d

JOIN Movies AS c

on c.movie\_id = d.movie\_id

WHERE substr(d.created\_at, 1, 7) = '2020-02'

GROUP BY d.movie\_id

ORDER BY rate DESC, movie\_name ASC LIMIT 1

) second\_query;

**Q36**.Table: Users

**Ans:**

create table if not exists Users

(

id int,

name varchar(50),

constraint pk PRIMARY KEY (id)

);

insert into Users VALUES (1,'Alice'),(2,'Bob'),(3,'Alex'),(4,'Donald'),(7,'Lee'),(13,'Jonathan'),(19,'Elvis');

select \* from Users;

--Table: Rides

create table if not exists Rides

(

id int,

user\_id int,

distance int,

constraint pk PRIMARY KEY (id),

constraint fk FOREIGN KEY (user\_id) REFERENCES Users(id)

);

insert into Rides VALUES (1,1,120), (2,2,317), (3,3,222), (4,7,100), (5,13,312), (6,19,50), (7,7,120), (8,19,400), (9,7,230);

select \* from Rides;

--Write an SQL query to report the distance travelled by each user.

--Return the result table ordered by travelled\_distance in descending order, if two or more users

--travelled the same distance, order them by their name in ascending order.

select name, sum(ifnull(distance, 0)) as travelled\_distance

from Rides r

right join Users u

on r.user\_id = u.id

group by name

order by 2 desc,1 asc;

**Q37**.

--Table: Employees

**Ans:**

create table if not exists Employees

(

id int,

name varchar(50),

constraint pk PRIMARY KEY (id)

);

insert into Employees VALUES (1,'Alice'),(7,'Bob'),(11,'Meir'),(90,'Winston'),(3,'Jonathan');

select \* from Employees;

--Table: EmployeeUNI

create table if not exists EmployeeUNI

(

id int,

unique\_id int,

constraint pk PRIMARY KEY (id, unique\_id)

);

insert into EmployeeUNI VALUES (3,1),(11,2),(90,3);

select \* from EmployeeUNI;

--Write an SQL query to show the unique ID of each user, If a user does not have a unique ID replace just show null.

--Return the result table in any order.

select unique\_id, name

from Employees

left join EmployeeUNI

on if (Employees.id = EmployeeUNI.id, Employees.id, null);

**Q38**.

--Table: Departments

**Ans:**

create table if not exists Departments

(

id int,

name varchar(50),

constraint pk PRIMARY KEY (id)

);

insert into Departments VALUES (1,'Electrical Engineering'),(7,'Computer Engineering'),(13,'Business Administration');

select \* from Departments;

--Table: Students

create table if not exists Students

(

id int,

name varchar(50),

department\_id int,

constraint pk PRIMARY KEY (id)

);

insert into Students VALUES (23,'Alice',1), (1,'Bob',7), (5,'Jennifer',13), (2,'John',14), (4,'Jasmine',77), (3,'Steve',74), (6,'Luis',1),(8,'Jonathan',7),(7,'Daiana',33),(11,'Madelynn',1);

select \* from Students;

--Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exist.

--Return the result table in any order.

select s.id, s.name

from Students s

left join Departments d

on s.department\_id = d.id

where d.id is null;

**Q39**.

--Table: Calls

**Ans:**

create table if not exists Calls

(

from\_id int,

to\_id int,

duration int

);

insert into Calls VALUES (1,2,59),(2,1,11),(1,3,20),(3,4,100),(3,4,200),(3,4,200),(4,3,499);

select \* from Calls;

--Write an SQL query to report the number of calls and the total call duration between each pair of distinct persons (person1, person2) where person1 < person2.

--Return the result table in any order.

SELECT LEAST(from\_id,to\_id) as person1,

GREATEST(from\_id,to\_id) as person2,

COUNT(\*) as call\_count,

SUM(duration) as total\_duration

FROM Calls

GROUP BY person1,person2;

**Q40**.

--Table: Prices

create table if not exists Prices

(

product\_id int,

start\_date date,

end\_date date,

price int,

constraint pk PRIMARY KEY (product\_id, start\_date, end\_date)

);

insert into Prices VALUES (1,'2019-02-17','2019-02-28',5), (1,'2019-03-01','2019-03-22',20), (2,'2019-02-01','2019-02-20',15), (2,'2019-02-21','2019-03-31',30);

select \* from Prices;

--UnitsSold table:

create table if not exists UnitsSold

(

product\_id int,

purchase\_date date,

units int

);

insert into UnitsSold VALUES (1,'2019-02-25',100),(1,'2019-03-01',15),(2,'2019-02-10',200),(2,'2019-03-22',30);

select \* from UnitsSold;

--Write an SQL query to find the average selling price for each product. average\_price should be rounded to 2 decimal places.

--Return the result table in any order.

select p.product\_id,

round(sum(p.price \* u.units)/sum(u.units), 2) as average\_price

from Prices p

left join UnitsSold u

on p.product\_id = u.product\_id and

datediff(u.purchase\_date, p.start\_date) >= 0 and

datediff(p.end\_date, u.purchase\_date) >= 0

group by p.product\_id;

**Q41**.

--Table: Warehouse

**Ans:**

create table if not exists Warehouse

(

name VARCHAR(50),

product\_id int,

units int,

constraint pk PRIMARY KEY (name, product\_id)

);

insert into Warehouse VALUES ('LCHouse1',1,1), ('LCHouse1',2,10), ('LCHouse1',3,5), ('LCHouse2',1,2), ('LCHouse2',2,2), ('LCHouse3',4,1);

select \* from Warehouse;

--Table: Products

create table if not exists Products

(

product\_id int,

product\_name VARCHAR(50),

Width int,

Length int,

Height int,

constraint pk PRIMARY KEY (product\_id)

);

insert into Products VALUES (1,'LC-TV',5,50,40),(2,'LC-KeyChain',5,5,5),(3,'LC-Phone',2,10,10),(4,'LC-T-Shirt',4,10,20);

select \* from Products;

--Write an SQL query to report the number of cubic feet of volume the inventory occupies in each warehouse.

--Return the result table in any order.

select name as warehouse\_name, sum(units \* vol) as volume

from Warehouse w

join (select product\_id, Width\*Length\*Height as vol

from Products) p

on w.product\_id = p.product\_id

group by name;

**Q42**.

--Table: Sales

**Ans:**

create table if not exists Sales

(

sale\_date date,

fruit enum("apples","oranges"),

sold\_num int,

constraint pk PRIMARY KEY (sale\_date, fruit)

);

insert into Sales VALUES ('2020-05-01','apples',10),('2020-05-01','oranges',8),('2020-05-02','apples',15),('2020-05-02','oranges',15),('2020-05-03','apples',20),('2020-05-03','oranges',0),('2020-05-04','apples',15),('2020-05-04','oranges',16);

select \* from Sales;

--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.

select a.sale\_date, (a.sold\_num - b.sold\_num) as diff

from Sales a left join Sales b

on a.sale\_date = b.sale\_date

where a.fruit = 'apples' and b.fruit = 'oranges';

**Q43**.

--Table: Activity

**Ans:**

create table if not exists Activity

(

player\_id int,

device\_id int,

event\_date date,

games\_played int,

constraint pk PRIMARY KEY (player\_id, event\_date)

);

insert into Activity VALUES (1,2,'2016-03-01',5), (1,2,'2016-03-02',6), (2,3,'2017-06-25',1), (3,1,'2016-03-02',0), (3,4,'2018-07-03',5);

select \* from Activity;

--Write an SQL query to report the fraction of players that logged in again on the day after the day they first logged in, rounded to 2 decimal places. In other words, you need to count the number of players that logged in for at least two consecutive days starting from their first login date, then divide that number by the total number of players.

WITH CTE AS (

SELECT

player\_id, min(event\_date) as event\_start\_date

from

Activity

group by player\_id )

SELECT

round((count(distinct c.player\_id) / (select count(distinct player\_id) from Activity)),2)as fraction

FROM

CTE c

JOIN Activity a

on c.player\_id = a.player\_id

and datediff(c.event\_start\_date, a.event\_date) = -1;

**Q44**.

--Table: Employee

**Ans:**

create table if not exists Employee

(

id int,

name VARCHAR(50),

department VARCHAR(50),

managerId int default null,

constraint pk PRIMARY KEY (id)

);

insert into Employee VALUES (101,'John','A',null), (102,'Dan','A',101), (103,'James','A',101), (104,'Amy','A',101), (105,'Anne','A',101), (106,'Ron','B',101);

select \* from Employee;

--Write an SQL query to report the managers with at least five direct reports.

--Return the result table in any order.

select

a.name

from

Employee a

inner join

Employee b

on (a.id = b.managerid)

group by a.name

having count(distinct b.id) >= 5;

-- select Name from Employee

-- where Id in

-- (

-- select ManagerId from Employee

-- group by 1

-- having count(\*) >= 5

-- );

**Q45**.

--Table: Department

**Ans:**

create table if not exists Department

(

dept\_id int,

dept\_name VARCHAR(50),

constraint pk PRIMARY KEY (dept\_id)

);

insert into Department VALUES (1,'Engineering'),(2,'Science'),(3,'Law');

select \* from Department;

--Table: Student

create table if not exists Student

(

student\_id int,

student\_name VARCHAR(50),

gender VARCHAR(50),

dept\_id int,

constraint pk PRIMARY KEY (student\_id),

constraint fk FOREIGN KEY (dept\_id) REFERENCES Department(dept\_id)

);

insert into Student VALUES (1,'Jack','M',1),(2,'Jane','F',1),(3,'Mark','M',2);

select \* from Student;

--Write an SQL query to report the respective department name and number of students majoring in each department for all departments in the Department table (even ones with no current students).

--Return the result table ordered by student\_number in descending order. In case of a tie, order them by dept\_name alphabetically.

select

a.dept\_name,

coalesce(count(student\_id), 0) student\_number

from

Department a

left join

Student b

on

(a.dept\_id = b.dept\_id)

group by a.dept\_name

order by student\_number desc, a.dept\_name asc;

**Q46**.

--Table: Product

**Ans:**

create table if not exists Product

(

product\_key int,

constraint pk PRIMARY KEY (product\_key)

);

insert into Product VALUES (5),(6);

select \* from Product;

--Table: Customer

create table if not exists Customer

(

customer\_id int,

product\_key int,

constraint fk FOREIGN KEY (product\_key) REFERENCES Product(product\_key)

);

insert into Customer VALUES (1,5),(2,6),(3,5),(3,6),(1,6);

select \* from Customer;

--Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table.

--Return the result table in any order.

SELECT

customer\_id

FROM customer

GROUP BY customer\_id

HAVING COUNT( DISTINCT product\_key) = (SELECT COUNT(\*) FROM product);

**Q47**.

--Table: Employee

**Ans:**

create table if not exists Employee

(

employee\_id int,

name VARCHAR(50),

experience\_years int,

constraint pk PRIMARY KEY (employee\_id)

);

insert into Employee VALUES (1,'Khaled',3),(2,'Ali',2),(3,'John',3),(4,'Doe',2);

select \* from Employee;

--Table: Project

create table if not exists Project

(

project\_id int,

employee\_id int,

constraint pk PRIMARY KEY (project\_id, employee\_id),

constraint fk FOREIGN KEY (employee\_id) REFERENCES Employee(employee\_id)

);

insert into Project VALUES (1,1),(1,2),(1,3),(2,1),(2,4);

select \* from Project;

--Write an SQL query that reports the most experienced employees in each project. In case of a tie, report all employees with the maximum number of experience years.

--Return the result table in any order.

SELECT

project\_id,

employee\_id

FROM (

SELECT

p.project\_id,

p.employee\_id,

DENSE\_RANK() OVER(PARTITION BY p.project\_id ORDER BY e.experience\_years DESC) as rnk

FROM Project as p JOIN Employee as e

ON p.employee\_id = e.employee\_id

) x

WHERE rnk = 1;

**Q48**.

--Table: Books

**Ans:**

create table if not exists Books

(

book\_id int,

name VARCHAR(50),

available\_from date,

constraint pk PRIMARY KEY (book\_id)

);

insert into Books VALUES (1,'"Kalila And Demna"','2010-01-01'),(2,'"28 Letters"','2012-05-12'),(3,'"The Hobbit"','2019-06-10'),(4,'"13 Reasons Why"','2010-01-01'),(5,'"The Hunger Games"','2008-09-21');

select \* from Books;

--Table: Orders

create table if not exists Orders

(

order\_id int,

book\_id int,

quantity int,

dispatch\_date date,

constraint pk PRIMARY KEY (order\_id),

constraint fk FOREIGN KEY (book\_id) REFERENCES Books(book\_id)

);

insert into Orders VALUES (1,1,2,'2018-07-26'), (2,1,1,'2018-11-05'), (3,3,8,'2019-06-11'), (4,4,6,'2019-06-05'), (5,4,5,'2019-06-20'), (6,5,9,'2009-02-02'),(7,5,8,'2010-04-13');

select \* from Orders;

--Write an SQL query that reports the books that have sold less than 10 copies in the last year, excluding books that have been available for less than one month from today. Assume today is 2019-06-23.

--Return the result table in any order.

select Books.book\_id, name from Books join Orders

on Books.book\_id = Orders.book\_id

where available\_from < '2019-05-23'

and dispatch\_date between '2018-06-23' and '2019-06-23'

group by Books.book\_id

having sum(quantity) < 10

union

select book\_id, name from Books

where available\_from < '2019-05-23'

and book\_id not in (

select distinct book\_id from Orders where dispatch\_date between '2018-06-23' and '2019-06-23'

);

**Q49**.

--Table: Enrollments

**Ans:**

create table if not exists Enrollments

(

student\_id int,

course\_id int,

grade int,

constraint pk PRIMARY KEY (student\_id, course\_id)

);

insert into Enrollments VALUES (2,2,95),(2,3,95),(1,1,90),(1,2,99),(3,1,80),(3,2,75),(3,3,82);

select \* from Enrollments;

--Write a SQL query to find the highest grade with its corresponding course for each student. In case of a tie, you should find the course with the smallest course\_id.

--Return the result table ordered by student\_id in ascending order.

select e.student\_id, e.course\_id, e.grade

from (

select \*, row\_number() over (partition by student\_id order by grade desc) rn

from Enrollments

) e

where e.rn = 1;

**Q50**.

--Table: Teams

--Players table:

**Ans:**

create table if not exists Players

(

player\_id int,

group\_id int,

constraint pk PRIMARY KEY (player\_id)

);

insert into Players VALUES (15,1), (25,1), (30,1), (45,1), (10,2), (35,2), (50,2), (20,3), (40,3);

select \* from Players;

--Table: Matches

create table if not exists Matches

(

match\_id int,

first\_player int,

second\_player int,

first\_score int,

second\_score int,

constraint pk PRIMARY KEY (match\_id)

);

insert into Matches VALUES (1,15,45,3,0),(2,30,25,1,2),(3,30,15,2,0),(4,40,20,5,2),(5,35,50,1,1);

select \* from Matches;

--Write an SQL query to find the winner in each group.

--Return the result table in any order.

select group\_id,player\_id from

(select group\_id,player\_id,sum((

case when player\_id = first\_player then first\_score

when player\_id = second\_player then second\_score

end

)) as totalScores

from Players p,Matches m

where p.player\_id = m.first\_player

or p.player\_id = m.second\_player

group by group\_id,player\_id

order by group\_id,totalScores desc,player\_id) as temp

group by group\_id

order by group\_id,totalScores desc,player\_id;

**Q-51**

**Ans-**

CREATE TABLE world

(

name VARCHAR(25),

continent VARCHAR(10),

area INT,

population INT,

gdp BIGINT,

CONSTRAINT pk\_world PRIMARY KEY (name)

);

INSERT INTO world VALUES('Afghanistan', 'Asia', 652230, 25500100, 20343000000);

INSERT INTO world VALUES('Albania', 'Europe', 28748, 2831741, 12960000000);

INSERT INTO world VALUES('Algeria', 'Africa', 2381741, 37100000, 188681000000);

INSERT INTO world VALUES('Andorra', 'Europe', 468, 78115, 3712000000);

INSERT INTO world VALUES('Angola', 'Africa', 1246700, 20609294, 100990000000);

SELECT

name,

population,

area

FROM

world

WHERE

area >= 3000000

OR population >= 25000000

;

--drop tables

DROP TABLE world;

**Q-52**

**Ans-**

CREATE TABLE customer

(

id INT,

name VARCHAR(25),

referee\_id BIGINT,

CONSTRAINT pk\_customer PRIMARY KEY (id)

);

INSERT INTO customer VALUES(1, 'Will', null);

INSERT INTO customer VALUES(2, 'Jane', null);

INSERT INTO customer VALUES(3, 'Alex', 2);

INSERT INTO customer VALUES(4, 'Bill', null);

INSERT INTO customer VALUES(5, 'Zack', 1);

INSERT INTO customer VALUES(6, 'Mark', 2);

SELECT

name

FROM

customer

WHERE

referee\_id <> 2

OR referee\_id IS NULL

;

--drop tables

DROP TABLE customer;

**Q-53**

**Ans-**

CREATE TABLE customers

(

id INT,

name VARCHAR(25),

CONSTRAINT pk\_customers PRIMARY KEY (id)

);

CREATE TABLE orders

(

id INT,

customer\_id INT,

CONSTRAINT pk\_orders PRIMARY KEY (id),

CONSTRAINT fk\_customer\_order FOREIGN KEY(customer\_id)

REFERENCES customers(id)

);

INSERT INTO customers VALUES(1, 'Joe');

INSERT INTO customers VALUES(2, 'Henry');

INSERT INTO customers VALUES(3, 'Sam');

INSERT INTO customers VALUES(4, 'Max');

INSERT INTO orders VALUES(1, 3);

INSERT INTO orders VALUES(2, 1);

SELECT

name

FROM

customers c

WHERE

NOT EXISTS(

SELECT

\*

FROM

orders o

WHERE c.id = o.customer\_id

)

;

--drop tables

DROP TABLE orders;

DROP TABLE customers;

**Q-54**

**Ans-**

CREATE TABLE employee

(

employee\_id INT,

team\_id INT,

CONSTRAINT pk\_employee PRIMARY KEY (employee\_id)

);

INSERT INTO employee VALUES(1, 8);

INSERT INTO employee VALUES(2, 8);

INSERT INTO employee VALUES(3, 8);

INSERT INTO employee VALUES(4, 7);

INSERT INTO employee VALUES(5, 9);

INSERT INTO employee VALUES(6, 9);

SELECT

employee\_id,

count(\*) OVER(PARTITION BY team\_id) AS team\_size

FROM

employee

;

--drop tables

DROP TABLE employee;

**Q-55**

**Ans-**

CREATE TABLE person

(

id INT,

name VARCHAR(25),

phone\_number VARCHAR(11),

CONSTRAINT pk\_person PRIMARY KEY (id)

);

CREATE TABLE country

(

name VARCHAR(25),

country\_code VARCHAR(3),

CONSTRAINT pk\_country PRIMARY KEY (country\_code)

);

CREATE TABLE calls

(

caller\_id INT,

callee\_id INT,

duration INT

);

INSERT INTO person VALUES(3, 'Jonathan', '051-1234567');

INSERT INTO person VALUES(12, 'Elvis', '051-7654321');

INSERT INTO person VALUES(1, 'Moncef', '212-1234567');

INSERT INTO person VALUES(2, 'Maroua', '212-6523651');

INSERT INTO person VALUES(7, 'Meir', '972-1234567');

INSERT INTO person VALUES(9, 'Rachel', '972-0011100');

INSERT INTO country VALUES('Peru', '51');

INSERT INTO country VALUES('Israel', '972');

INSERT INTO country VALUES('Morocco', '212');

INSERT INTO country VALUES('Germany', '49');

INSERT INTO country VALUES('Ethiopia', '251');

INSERT INTO calls VALUES(1, 9, 33);

INSERT INTO calls VALUES(2, 9, 4);

INSERT INTO calls VALUES(1, 2, 59);

INSERT INTO calls VALUES(3, 12, 102);

INSERT INTO calls VALUES(3, 12, 330);

INSERT INTO calls VALUES(12, 3, 5);

INSERT INTO calls VALUES(7, 9, 13);

INSERT INTO calls VALUES(7, 1, 3);

INSERT INTO calls VALUES(9, 7, 1);

INSERT INTO calls VALUES(1, 7, 7);

WITH receiver\_caller\_calls AS(

SELECT

caller\_id AS caller\_receiver\_id,

duration

FROM

calls

UNION ALL

SELECT

callee\_id AS caller\_receiver\_id,

duration

FROM

calls

),

call\_duration\_avg AS(

SELECT

DISTINCT cn.name,

avg(c.duration) OVER() as global\_average,

avg(c.duration) OVER(PARTITION BY cn.name) as country\_average

FROM

person p

JOIN country cn

ON CAST(SUBSTRING\_INDEX(p.phone\_number, '-', 1) AS UNSIGNED) = CAST(cn.country\_code AS UNSIGNED)

JOIN receiver\_caller\_calls c

ON c.caller\_receiver\_id = p.id

)

SELECT

name

FROM

call\_duration\_avg

WHERE

country\_average > global\_average

;

--drop tables

DROP TABLE person;

DROP TABLE country;

DROP TABLE calls;

**Q-56**

**Ans-**

CREATE TABLE activity

(

player\_id INT,

device\_id INT,

event\_date DATE,

games\_played INT

);

INSERT INTO activity VALUES(1, 2, '2016-03-01', 5);

INSERT INTO activity VALUES(1, 2, '2016-05-02', 6);

INSERT INTO activity VALUES(2, 3, '2017-06-25', 1);

INSERT INTO activity VALUES(3, 1, '2016-03-02', 0);

INSERT INTO activity VALUES(3, 4, '2018-07-03', 5);

WITH activity\_with\_device\_serial AS(

SELECT

player\_id,

device\_id,

DENSE\_RANK() OVER(PARTITION BY player\_id ORDER BY event\_date) as device\_serial

FROM

activity

)

SELECT

player\_id,

device\_id

FROM

activity\_with\_device\_serial

WHERE

device\_serial = 1

;

--drop tables

DROP TABLE activity;

**Q-57**

**Ans-**

CREATE TABLE orders

(

order\_number INT,

customer\_number INT,

CONSTRAINT pk\_orders PRIMARY KEY (order\_number)

);

INSERT INTO orders VALUES(1, 1);

INSERT INTO orders VALUES(2, 2);

INSERT INTO orders VALUES(3, 3);

INSERT INTO orders VALUES(4, 3);

WITH customer\_with\_no\_of\_order AS(

SELECT

customer\_number,

count(\*) AS no\_of\_order

FROM

orders

GROUP BY customer\_number

)

SELECT

customer\_number

FROM

customer\_with\_no\_of\_order

ORDER BY no\_of\_order DESC

LIMIT 1;

--Follow Up:

INSERT INTO orders VALUES(5, 2);

WITH customer\_with\_no\_of\_order AS(

SELECT

customer\_number,

count(\*) AS no\_of\_order

FROM

orders

GROUP BY customer\_number

)

SELECT

customer\_number

FROM

customer\_with\_no\_of\_order

WHERE

no\_of\_order IN (

SELECT

max(no\_of\_order) AS max\_no\_of\_order

FROM

customer\_with\_no\_of\_order

)

;

--drop tables

DROP TABLE orders;

**Q-58**

**Ans-**

CREATE TABLE cinema

(

seat\_id INT AUTO\_INCREMENT,

free BOOLEAN,

CONSTRAINT pk\_cinema PRIMARY KEY (seat\_id)

);

INSERT INTO cinema(free) VALUES(TRUE);

INSERT INTO cinema(free) VALUES(FALSE);

INSERT INTO cinema(free) VALUES(TRUE);

INSERT INTO cinema(free) VALUES(TRUE);

INSERT INTO cinema(free) VALUES(TRUE);

WITH seat\_id\_with\_diff\_with\_next\_val AS (

SELECT

seat\_id,

seat\_id - LEAD(seat\_id, 1) OVER(ORDER BY seat\_id) AS diff\_next\_val,

seat\_id - LAG(seat\_id, 1) OVER(ORDER BY seat\_id) AS diff\_prev\_val

FROM

cinema

WHERE

free <> 0

)

SELECT

seat\_id

FROM

seat\_id\_with\_diff\_with\_next\_val

WHERE

diff\_next\_val = -1

OR diff\_prev\_val = 1

;

--drop tables

DROP TABLE cinema;

**Q-59**

**Ans-**

CREATE TABLE sales\_person

(

sales\_id INT,

name VARCHAR(25),

salary INT,

commission\_rate INT,

hire\_rate DATE,

CONSTRAINT pk\_sales\_person PRIMARY KEY (sales\_id)

);

CREATE TABLE company

(

com\_id INT,

name VARCHAR(25),

city VARCHAR(25),

CONSTRAINT pk\_company PRIMARY KEY (com\_id)

);

CREATE TABLE orders

(

order\_id INT,

order\_date DATE,

com\_id INT,

sales\_id INT,

amount INT,

CONSTRAINT pk\_orders PRIMARY KEY (order\_id),

CONSTRAINT fk\_company FOREIGN KEY (com\_id)

REFERENCES company(com\_id),

CONSTRAINT fk\_sales\_person FOREIGN KEY (sales\_id)

REFERENCES sales\_person(sales\_id)

);

INSERT INTO sales\_person VALUES(1, 'John', 100000, 6, '2006-04-01');

INSERT INTO sales\_person VALUES(2, 'Amy', 12000, 5, '2010-05-01');

INSERT INTO sales\_person VALUES(3, 'Mark', 65000, 12, '2008-12-25');

INSERT INTO sales\_person VALUES(4, 'Pam', 25000, 25, '2005-01-01');

INSERT INTO sales\_person VALUES(5, 'Alex', 5000, 10, '2007-02-03');

INSERT INTO company VALUES(1, 'RED', 'Boston');

INSERT INTO company VALUES(2, 'ORANGE', 'New York');

INSERT INTO company VALUES(3, 'YELLOW', 'Boston');

INSERT INTO company VALUES(4, 'GREEN', 'Austin');

INSERT INTO orders VALUES(1, '2014-01-01', 3, 4, 10000);

INSERT INTO orders VALUES(2, '2014-02-01', 4, 5, 5000);

INSERT INTO orders VALUES(3, '2014-03-01', 1, 1, 50000);

INSERT INTO orders VALUES(4, '2014-04-01', 1, 4, 25000);

SELECT

name

FROM

sales\_person sp

WHERE

NOT EXISTS(

SELECT

\*

FROM

orders o

JOIN company c ON o.com\_id = c.com\_id

WHERE

sp.sales\_id = o.sales\_id

AND c.name = 'RED'

)

;

--drop tables

DROP TABLE orders;

DROP TABLE company;

DROP TABLE sales\_person;

**Q-60**

**Ans-**

CREATE TABLE triangle

(

x INT,

y INT,

z INT,

CONSTRAINT pk\_triangle PRIMARY KEY (x,y,z)

);

INSERT INTO triangle VALUES(13, 15, 30);

INSERT INTO triangle VALUES(10, 20, 15);

SELECT

\*,

CASE

WHEN x + y > z AND y + z > x AND z + x > y

THEN 'Yes'

ELSE 'No'

END as triangle

FROM

triangle

;

--drop tables

DROP TABLE triangle;

**Q-61**

**Ans-**

CREATE TABLE point

(

x INT,

CONSTRAINT pk\_point PRIMARY KEY (x)

);

INSERT INTO point VALUES(-1);

INSERT INTO point VALUES(0);

INSERT INTO point VALUES(2);

SELECT

MIN(ABS(p1.x - p2.x)) AS shortest

FROM

point p1

JOIN point p2 ON p1.x <> p2.x

;

--Follow Up:

WITH ordered\_point AS(

SELECT

x,

ROW\_NUMBER() OVER(ORDER BY x) AS ordered\_no

FROM

point

)

SELECT

MIN(ABS(p1.x - p2.x)) AS shortest

FROM

ordered\_point p1

JOIN ordered\_point p2 ON p2.ordered\_no > p1.ordered\_no

;

--drop tables

DROP TABLE point;

**Q-62**

**Ans-**

CREATE TABLE actor\_director

(

actor\_id INT,

director\_id INT,

timestamp INT,

CONSTRAINT pk\_actor\_director PRIMARY KEY (timestamp)

);

INSERT INTO actor\_director VALUES(1, 1, 0);

INSERT INTO actor\_director VALUES(1, 1, 1);

INSERT INTO actor\_director VALUES(1, 1, 2);

INSERT INTO actor\_director VALUES(1, 2, 3);

INSERT INTO actor\_director VALUES(1, 2, 4);

INSERT INTO actor\_director VALUES(2, 1, 5);

INSERT INTO actor\_director VALUES(2, 1, 6);

SELECT

actor\_id,

director\_id

FROM

actor\_director

GROUP BY

actor\_id,

director\_id

HAVING

count(\*) >= 3

;

--drop tables

DROP TABLE actor\_director;

--Q63:

CREATE TABLE product

(

product\_id INT,

product\_name VARCHAR(25),

CONSTRAINT pk\_product PRIMARY KEY (product\_id)

);

CREATE TABLE sales

(

sale\_id INT,

product\_id INT,

year INT,

quantity INT,

price INT,

CONSTRAINT pk\_sales PRIMARY KEY (sale\_id, year),

CONSTRAINT fk\_product FOREIGN KEY (product\_id)

REFERENCES product(product\_id)

);

INSERT INTO product VALUES(100, 'Nokia');

INSERT INTO product VALUES(200, 'Apple');

INSERT INTO product VALUES(300, 'Samsung');

INSERT INTO sales VALUES(1, 100, 2008, 10, 5000);

INSERT INTO sales VALUES(2, 100, 2009, 12, 5000);

INSERT INTO sales VALUES(7, 200, 2011, 15, 9000);

SELECT

product\_name,

year,

price

FROM

product p

JOIN sales s ON p.product\_id = s.product\_id

;

--drop tables

DROP TABLE sales;

DROP TABLE product;

**Q-64**

**Ans-**

CREATE TABLE employee

(

employee\_id INT,

name VARCHAR(25),

experience\_years INT,

CONSTRAINT pk\_employee PRIMARY KEY (employee\_id)

);

CREATE TABLE project

(

project\_id INT,

employee\_id INT,

CONSTRAINT pk\_project PRIMARY KEY (project\_id, employee\_id),

CONSTRAINT fk\_employee FOREIGN KEY (employee\_id)

REFERENCES employee(employee\_id)

);

INSERT INTO employee VALUES(1, 'Khaled', 3);

INSERT INTO employee VALUES(2, 'Ali', 2);

INSERT INTO employee VALUES(3, 'John', 1);

INSERT INTO employee VALUES(4, 'Doe', 2);

INSERT INTO project VALUES(1, 1);

INSERT INTO project VALUES(1, 2);

INSERT INTO project VALUES(1, 3);

INSERT INTO project VALUES(2, 1);

INSERT INTO project VALUES(2, 4);

SELECT

p.project\_id,

ROUND(avg(e.experience\_years),2) AS average\_years

FROM

project p

JOIN employee e ON p.employee\_id = e. employee\_id

GROUP BY p.project\_id

;

--drop tables

DROP TABLE project;

DROP TABLE employee;

**Q-65**

**Ans-**

CREATE TABLE product

(

product\_id INT,

product\_name VARCHAR(25),

unit\_price INT,

CONSTRAINT pk\_product PRIMARY KEY (product\_id)

);

CREATE TABLE sales

(

seller\_id INT,

product\_id INT,

buyer\_id INT,

sale\_date DATE,

quantity INT,

price INT,

CONSTRAINT fk\_product FOREIGN KEY (product\_id)

REFERENCES product(product\_id)

);

INSERT INTO product VALUES(1, 'S8', 1000);

INSERT INTO product VALUES(2, 'G4', 800);

INSERT INTO product VALUES(3, 'iPhone', 1400);

INSERT INTO sales VALUES(1, 1, 1, '2019-01-21', 2, 2000);

INSERT INTO sales VALUES(1, 2, 2, '2019-02-17', 1, 800);

INSERT INTO sales VALUES(2, 2, 3, '2019-06-02', 1, 800);

INSERT INTO sales VALUES(3, 3, 4, '2019-05-13', 2, 2800);

WITH sales\_rank AS(

SELECT

seller\_id,

sum(price) AS total\_sales,

DENSE\_RANK() OVER(ORDER BY sum(price) DESC) rank\_by\_sales

FROM

sales

GROUP BY

seller\_id

)

SELECT

seller\_id

FROM

sales\_rank

WHERE

rank\_by\_sales = 1

;

--drop tables

DROP TABLE sales;

DROP TABLE product;

**Q-66**

**Ans-**

CREATE TABLE product

(

product\_id INT,

product\_name VARCHAR(25),

unit\_price INT,

CONSTRAINT pk\_product PRIMARY KEY (product\_id)

);

CREATE TABLE sales

(

seller\_id INT,

product\_id INT,

buyer\_id INT,

sale\_date DATE,

quantity INT,

price INT,

CONSTRAINT fk\_product FOREIGN KEY (product\_id)

REFERENCES product(product\_id)

);

INSERT INTO product VALUES(1, 'S8', 1000);

INSERT INTO product VALUES(2, 'G4', 800);

INSERT INTO product VALUES(3, 'iPhone', 1400);

INSERT INTO sales VALUES(1, 1, 1, '2019-01-21', 2, 2000);

INSERT INTO sales VALUES(1, 2, 2, '2019-02-17', 1, 800);

INSERT INTO sales VALUES(2, 1, 3, '2019-06-02', 1, 800);

INSERT INTO sales VALUES(3, 3, 3, '2019-05-13', 2, 2800);

WITH s8\_iphone\_sales AS(

SELECT

s.buyer\_id,

p.product\_name

FROM

sales s

JOIN product p ON s.product\_id = p.product\_id

WHERE

p.product\_name = 'S8'

OR p.product\_name = 'iPhone'

)

SELECT

s1.buyer\_id

FROM

s8\_iphone\_sales s1

LEFT JOIN s8\_iphone\_sales s2

ON (s1.product\_name = 'S8'

OR s1.product\_name = 'iPhone')

AND s2.product\_name = 'iPhone'

AND s1.buyer\_id = s2.buyer\_id

WHERE

s2.buyer\_id is null

;

--drop tables

DROP TABLE sales;

DROP TABLE product;

**Q-67**

**Ans-**

CREATE TABLE customer

(

customer\_id INT,

name VARCHAR(25),

visited\_on DATE,

amount INT,

CONSTRAINT pk\_customer PRIMARY KEY (customer\_id, visited\_on)

);

INSERT INTO customer VALUES(1, 'Jhon', '2019-01-01', 100);

INSERT INTO customer VALUES(2, 'Daniel', '2019-01-02', 110);

INSERT INTO customer VALUES(3, 'Jade', '2019-01-03', 120);

INSERT INTO customer VALUES(4, 'Khaled', '2019-01-04', 130);

INSERT INTO customer VALUES(5, 'Winston', '2019-01-05', 110);

INSERT INTO customer VALUES(6, 'Elvis', '2019-01-06', 140);

INSERT INTO customer VALUES(7, 'Anna', '2019-01-07', 150);

INSERT INTO customer VALUES(8, 'Maria', '2019-01-08', 80);

INSERT INTO customer VALUES(9, 'Jaze', '2019-01-09', 110);

INSERT INTO customer VALUES(1, 'Jhon', '2019-01-10', 130);

INSERT INTO customer VALUES(3, 'Jade', '2019-01-10', 150);

WITH daily\_sales AS(

SELECT

visited\_on,

sum(amount) AS daily\_sales\_total\_amount

FROM

customer

GROUP BY visited\_on

)

SELECT

visited\_on,

sum(daily\_sales\_total\_amount)

OVER(ORDER BY visited\_on RANGE BETWEEN INTERVAL '6' day PRECEDING AND CURRENT ROW) AS amount,

ROUND(avg(daily\_sales\_total\_amount)

OVER(ORDER BY visited\_on RANGE BETWEEN INTERVAL '6' day PRECEDING AND CURRENT ROW), 2) AS average\_amount

FROM

daily\_sales

LIMIT 6,18446744073709551615

;

--drop tables

DROP TABLE customer;

**Q-68**

**Ans-**

CREATE TABLE scores

(

player\_name VARCHAR(25),

gender VARCHAR(1),

day DATE,

score\_points INT,

CONSTRAINT pk\_scores PRIMARY KEY (gender, day)

);

INSERT INTO scores VALUES('Aron', 'F', '2020-01-01', 17);

INSERT INTO scores VALUES('Alice', 'F', '2020-01-07', 23);

INSERT INTO scores VALUES('Bajrang', 'M', '2020-01-07', 7);

INSERT INTO scores VALUES('Khali' , 'M', '2019-12-25', 11);

INSERT INTO scores VALUES('Slaman', 'M', '2019-12-30', 13);

INSERT INTO scores VALUES('Joe', 'M', '2019-12-31', 3);

INSERT INTO scores VALUES('Jose', 'M', '2019-12-18', 2);

INSERT INTO scores VALUES('Priya', 'F', '2019-12-31', 23);

INSERT INTO scores VALUES('Priyanka', 'F', '2019-12-30', 17);

SELECT

gender,

day,

sum(score\_points) OVER(PARTITION BY gender ORDER BY day) AS total

FROM

scores

;

--drop tables

DROP TABLE scores;

**Q-69**

**Ans-**

CREATE TABLE logs

(

log\_id INT,

CONSTRAINT pk\_logs PRIMARY KEY (log\_id)

);

INSERT INTO logs VALUES(1),(2),(3),(7),(8),(10);

WITH log\_id\_with\_diff\_with\_next\_val AS (

SELECT

log\_id,

IFNULL(log\_id - LEAD(log\_id, 1) OVER(ORDER BY log\_id), 0) AS diff\_next\_val,

IFNULL(log\_id - LAG(log\_id, 1) OVER(ORDER BY log\_id), 0) AS diff\_prev\_val

FROM

logs

),

start\_end\_logs AS(

SELECT

CASE

WHEN ABS(diff\_next\_val) = 1

OR (ABS(diff\_next\_val) <> 1

AND ABS(diff\_prev\_val) <> 1)

THEN log\_id

END start\_id,

CASE

WHEN ABS(LEAD(diff\_prev\_val) OVER(ORDER BY log\_id)) = 1

THEN LEAD(log\_id) OVER(ORDER BY log\_id)

WHEN (ABS(diff\_next\_val) <> 1

AND ABS(diff\_prev\_val) <> 1)

THEN log\_id

END end\_id

FROM

log\_id\_with\_diff\_with\_next\_val

WHERE

NOT (

ABS(IFNULL(diff\_next\_val, 0)) = 1

AND ABS(IFNULL(diff\_prev\_val, 0)) = 1

)

)

SELECT

\*

FROM

start\_end\_logs

WHERE

start\_id IS NOT NULL

;

--drop tables

DROP TABLE logs;

**Q-70**

**Ans-**

CREATE TABLE students

(

student\_id INT,

student\_name VARCHAR(25),

CONSTRAINT pk\_students PRIMARY KEY (student\_id)

);

CREATE TABLE subjects

(

subject\_name VARCHAR(25),

CONSTRAINT pk\_students PRIMARY KEY (subject\_name)

);

CREATE TABLE examinations

(

student\_id INT,

subject\_name VARCHAR(25)

);

INSERT INTO students VALUES(1, 'Alice');

INSERT INTO students VALUES(2, 'Bob');

INSERT INTO students VALUES(13, 'John');

INSERT INTO students VALUES(6,'Alex');

INSERT INTO subjects VALUES('Math');

INSERT INTO subjects VALUES('Physics');

INSERT INTO subjects VALUES('Programming');

INSERT INTO examinations VALUES(1,'Math');

INSERT INTO examinations VALUES(1, 'Physics');

INSERT INTO examinations VALUES(1, 'Programming');

INSERT INTO examinations VALUES(2, 'Programming');

INSERT INTO examinations VALUES(1, 'Physics');

INSERT INTO examinations VALUES(1, 'Math');

INSERT INTO examinations VALUES(13, 'Math');

INSERT INTO examinations VALUES(13, 'Programming');

INSERT INTO examinations VALUES(13, 'Physics');

INSERT INTO examinations VALUES(2, 'Math');

INSERT INTO examinations VALUES(1, 'Math');

SELECT

st.student\_id,

st.student\_name,

sb.subject\_name,

sum(

CASE

WHEN ex.subject\_name IS NOT NULL

THEN 1

ELSE

0

END

) as attended\_exams

FROM students st

JOIN subjects sb

LEFT JOIN examinations ex ON ex.subject\_name = sb.subject\_name

AND st.student\_id = ex.student\_id

GROUP BY

st.student\_id,

st.student\_name,

sb.subject\_name

ORDER BY

st.student\_id,

st.student\_name

;

--drop tables

DROP TABLE students;

DROP TABLE subjects;

DROP TABLE examinations;

**Q-71**

**Ans-**

CREATE TABLE employees

(

employee\_id INT,

employee\_name VARCHAR(25),

manager\_id INT,

CONSTRAINT pk\_employees PRIMARY KEY(employee\_id)

);

INSERT INTO employees VALUES(1, 'Boss', 1);

INSERT INTO employees VALUES(3, 'Alice', 3);

INSERT INTO employees VALUES(2, 'Bob', 1);

INSERT INTO employees VALUES(4, 'Daniel', 2);

INSERT INTO employees VALUES(7, 'Luis', 4);

INSERT INTO employees VALUES(8, 'Jhon', 3);

INSERT INTO employees VALUES(9, 'Angela', 8);

INSERT INTO employees VALUES(77, 'Robert', 1);

WITH RECURSIVE emp\_hir AS

(

SELECT

employee\_id,

manager\_id,

employee\_name,

1 as lvl

FROM

employees

WHERE

employee\_name = 'Boss'

UNION

SELECT

em.employee\_id,

em.manager\_id,

em.employee\_name,

eh.lvl + 1 as lvl

FROM

emp\_hir eh

JOIN employees em ON eh.employee\_id = em.manager\_id

WHERE

em.employee\_name <> 'Boss'

)

SELECT

eh1.employee\_id

FROM

emp\_hir eh1

WHERE

eh1.employee\_name <> 'Boss'

;

--drop tables

DROP TABLE employees;

**Q-72**

**Ans-**

CREATE TABLE transactions

(

id INT,

country VARCHAR(25),

state VARCHAR(15),

amount INT,

trans\_date DATE,

CONSTRAINT pk\_trx PRIMARY KEY(id)

);

INSERT INTO transactions VALUES(121, 'US', 'approved', 1000, '2018-12-18');

INSERT INTO transactions VALUES(122, 'US', 'declined', 2000, '2018-12-19');

INSERT INTO transactions VALUES(123, 'US', 'approved', 2000, '2019-01-01');

INSERT INTO transactions VALUES(124, 'DE', 'approved', 2000, '2019-01-07');

SELECT

DATE\_FORMAT(trans\_date, '%Y-%m') AS month,

country,

COUNT(\*) AS trans\_count,

COUNT(

CASE

WHEN state = 'approved'

THEN id

END

) AS approved\_count,

SUM(amount) AS trans\_total\_amount,

SUM(

CASE

WHEN state = 'approved'

THEN amount

END

) AS approved\_total\_amount

FROM

transactions

GROUP BY

DATE\_FORMAT(trans\_date, '%Y-%m'),

country

;

--drop tables

DROP TABLE transactions;

**Q-73**

**Ans-**

CREATE TABLE actions

(

user\_id INT,

post\_id INT,

action\_date DATE,

action VARCHAR(15),

extra VARCHAR(25)

);

CREATE TABLE removals

(

post\_id INT,

remove\_date DATE

);

INSERT INTO actions VALUES(1, 1, '2019-07-01','view', null);

INSERT INTO actions VALUES(1, 1, '2019-07-01', 'like', null);

INSERT INTO actions VALUES(1, 1, '2019-07-01', 'share', null);

INSERT INTO actions VALUES(2, 2, '2019-07-04', 'view', null);

INSERT INTO actions VALUES(2, 2, '2019-07-04', 'report', 'spam');

INSERT INTO actions VALUES(3, 4, '2019-07-04', 'view', null);

INSERT INTO actions VALUES(3, 4, '2019-07-04', 'report', 'spam');

INSERT INTO actions VALUES(4, 3, '2019-07-02', 'view', null);

INSERT INTO actions VALUES(4, 3, '2019-07-02', 'report', 'spam');

INSERT INTO actions VALUES(5, 2, '2019-07-03', 'view', null);

INSERT INTO actions VALUES(5, 2, '2019-07-03', 'report', 'racism');

INSERT INTO actions VALUES(5, 5, '2019-07-03', 'view', null);

INSERT INTO actions VALUES(5, 5, '2019-07-03', 'report', 'racism');

INSERT INTO removals VALUES(2, '2019-07-20');

INSERT INTO removals VALUES(3, '2019-07-18');

WITH reported\_removed\_stat AS(

SELECT

a.action\_date,

COUNT(\*) AS total\_reported,

COUNT(

CASE

WHEN r.post\_id IS NOT NULL

THEN r.post\_id

END

) AS total\_removed

FROM

actions a

LEFT JOIN removals r ON r.post\_id = a.post\_id

WHERE

a.action = 'report'

AND a.extra = 'spam'

GROUP BY

a.action\_date

),

daily\_pct AS(

SELECT

action\_date,

(total\_removed)\*100.00/total\_reported as daily\_ratio

FROM

reported\_removed\_stat

GROUP BY

action\_date

)

SELECT

ROUND(AVG(daily\_ratio), 2) AS average\_daily\_percent

FROM

daily\_pct

;

--drop tables

DROP TABLE actions;

DROP TABLE removals;

**Q-74**

**Ans-**

CREATE TABLE activity

(

player\_id INT,

device\_id INT,

event\_date DATE,

games\_played INT,

CONSTRAINT pk\_activity PRIMARY KEY(player\_id, event\_date)

);

INSERT INTO activity VALUES(1, 2, '2016-03-01', 5);

INSERT INTO activity VALUES(1 ,2, '2016-03-02', 6);

INSERT INTO activity VALUES(2, 3, '2017-06-25', 1);

INSERT INTO activity VALUES(3, 1, '2016-03-02', 0);

INSERT INTO activity VALUES(3, 4, '2018-07-03', 5);

WITH logged\_in\_prev\_day AS(

SELECT

count(DISTINCT player\_id) AS player\_count

FROM

activity a1

WHERE

EXISTS(

SELECT

\*

FROM

activity a2

WHERE

a1.player\_id = a2.player\_id

AND a1.event\_date = DATE\_ADD(a2.event\_date, INTERVAL 1 DAY)

)

),

all\_player AS(

SELECT

count(DISTINCT player\_id) AS total\_player\_count

FROM

activity

)

SELECT

ROUND(

(SELECT

player\_count

FROM

logged\_in\_prev\_day)\*1.00/

(SELECT

total\_player\_count

FROM

all\_player)

,2) AS fraction

;

--drop tables

DROP TABLE activity;

**Q-75**

**Ans-**

CREATE TABLE activity

(

player\_id INT,

device\_id INT,

event\_date DATE,

games\_played INT,

CONSTRAINT pk\_activity PRIMARY KEY(player\_id, event\_date)

);

INSERT INTO activity VALUES(1, 2, '2016-03-01', 5);

INSERT INTO activity VALUES(1 ,2, '2016-03-02', 6);

INSERT INTO activity VALUES(2, 3, '2017-06-25', 1);

INSERT INTO activity VALUES(3, 1, '2016-03-02', 0);

INSERT INTO activity VALUES(3, 4, '2018-07-03', 5);

WITH logged\_in\_prev\_day AS(

SELECT

count(DISTINCT player\_id) AS player\_count

FROM

activity a1

WHERE

EXISTS(

SELECT

\*

FROM

activity a2

WHERE

a1.player\_id = a2.player\_id

AND a1.event\_date = DATE\_ADD(a2.event\_date, INTERVAL 1 DAY)

)

),

all\_player AS(

SELECT

count(DISTINCT player\_id) AS total\_player\_count

FROM

activity

)

SELECT

ROUND(

(SELECT

player\_count

FROM

logged\_in\_prev\_day)\*1.00/

(SELECT

total\_player\_count

FROM

all\_player)

,2) AS fraction

;

--drop tables

DROP TABLE activity;

**Q-76**

**Ans-**

CREATE TABLE salaries

(

company\_id INT,

employee\_id INT,

employee\_name VARCHAR(25),

salary INT,

CONSTRAINT pk\_salary PRIMARY KEY(company\_id, employee\_id)

);

INSERT INTO salaries VALUES(1, 1, 'Tony', 2000);

INSERT INTO salaries VALUES(1, 2, 'Pronub', 21300);

INSERT INTO salaries VALUES(1, 3, 'Tyrrox', 10800);

INSERT INTO salaries VALUES(2, 1, 'Pam', 300);

INSERT INTO salaries VALUES(2, 7, 'Bassem', 450);

INSERT INTO salaries VALUES(2, 9, 'Hermione', 700);

INSERT INTO salaries VALUES(3, 7, 'Bocaben', 100);

INSERT INTO salaries VALUES(3, 2, 'Ognjen', 2200);

INSERT INTO salaries VALUES(3, 13, 'Nyan Cat', 3300);

INSERT INTO salaries VALUES(3, 15, 'Morning Cat', 7777);

WITH salaries\_with\_max\_by\_company AS (

SELECT

\*,

MAX(salary) OVER(PARTITION BY company\_id) AS max\_company\_salary

FROM

salaries

)

SELECT

company\_id,

employee\_id,

employee\_name,

CASE

WHEN max\_company\_salary > 10000

THEN

CEILING(salary - (salary\*0.49))

WHEN max\_company\_salary >= 1000

THEN

CEILING(salary - (salary\*0.24))

ELSE

salary

END AS salary

FROM

salaries\_with\_max\_by\_company

;

--drop tables

DROP TABLE salaries;

**Q-77**

**Ans-**

CREATE TABLE variables

(

name VARCHAR(5),

value INT,

CONSTRAINT pk\_variables PRIMARY KEY(name)

);

CREATE TABLE expressions

(

left\_operand VARCHAR(5),

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

right\_operand VARCHAR(5),

CONSTRAINT pk\_expressions PRIMARY KEY(left\_operand, operator, right\_operand)

);

INSERT INTO variables VALUES('x', 66);

INSERT INTO variables VALUES('y', 77);

INSERT INTO expressions VALUES('x', '>', 'y');

INSERT INTO expressions VALUES('x', '<', 'y');

INSERT INTO expressions VALUES('x', '=', 'y');

INSERT INTO expressions VALUES('y', '>', 'x');

INSERT INTO expressions VALUES('y', '<', 'x');

INSERT INTO expressions VALUES('y', '=', 'x');

WITH variables\_with\_val AS(

SELECT

e.left\_operand,

e.right\_operand,

e.operator,

MAX(CASE

WHEN e.left\_operand = v.name

THEN v.value

END) as first\_var,

MAX(CASE

WHEN e.right\_operand = v.name

THEN v.value

END) as second\_var

FROM

expressions e

JOIN variables v

ON e.left\_operand = v.name

OR e.right\_operand = v.name

GROUP BY

left\_operand,

right\_operand,

operator

)

SELECT

left\_operand,

operator,

right\_operand,

CASE

WHEN operator = '<'

THEN

IF(first\_var < second\_var, 'true', 'false')

WHEN operator = '>'

THEN

IF(first\_var > second\_var, 'true', 'false')

WHEN operator = '='

THEN

IF(first\_var = second\_var, 'true', 'false')

END value

FROM

variables\_with\_val

;

--drop tables

DROP TABLE variables;

**Q-78**

**Ans-**

CREATE TABLE person

(

id INT,

name VARCHAR(25),

phone\_number VARCHAR(11),

CONSTRAINT pk\_person PRIMARY KEY (id)

);

CREATE TABLE country

(

name VARCHAR(25),

country\_code VARCHAR(3),

CONSTRAINT pk\_country PRIMARY KEY (country\_code)

);

CREATE TABLE calls

(

caller\_id INT,

callee\_id INT,

duration INT

);

INSERT INTO person VALUES(3, 'Jonathan', '051-1234567');

INSERT INTO person VALUES(12, 'Elvis', '051-7654321');

INSERT INTO person VALUES(1, 'Moncef', '212-1234567');

INSERT INTO person VALUES(2, 'Maroua', '212-6523651');

INSERT INTO person VALUES(7, 'Meir', '972-1234567');

INSERT INTO person VALUES(9, 'Rachel', '972-0011100');

INSERT INTO country VALUES('Peru', '51');

INSERT INTO country VALUES('Israel', '972');

INSERT INTO country VALUES('Morocco', '212');

INSERT INTO country VALUES('Germany', '49');

INSERT INTO country VALUES('Ethiopia', '251');

INSERT INTO calls VALUES(1, 9, 33);

INSERT INTO calls VALUES(2, 9, 4);

INSERT INTO calls VALUES(1, 2, 59);

INSERT INTO calls VALUES(3, 12, 102);

INSERT INTO calls VALUES(3, 12, 330);

INSERT INTO calls VALUES(12, 3, 5);

INSERT INTO calls VALUES(7, 9, 13);

INSERT INTO calls VALUES(7, 1, 3);

INSERT INTO calls VALUES(9, 7, 1);

INSERT INTO calls VALUES(1, 7, 7);

WITH receiver\_caller\_calls AS(

SELECT

caller\_id AS caller\_receiver\_id,

duration

FROM

calls

UNION ALL

SELECT

callee\_id AS caller\_receiver\_id,

duration

FROM

calls

),

call\_duration\_avg AS(

SELECT

DISTINCT cn.name,

avg(c.duration) OVER() as global\_average,

avg(c.duration) OVER(PARTITION BY cn.name) as country\_average

FROM

person p

JOIN country cn

ON CAST(SUBSTRING\_INDEX(p.phone\_number, '-', 1) AS UNSIGNED) = CAST(cn.country\_code AS UNSIGNED)

JOIN receiver\_caller\_calls c

ON c.caller\_receiver\_id = p.id

)

SELECT

name

FROM

call\_duration\_avg

WHERE

country\_average > global\_average

;

--drop tables

DROP TABLE person;

DROP TABLE country;

DROP TABLE calls;

**Q-79**

**Ans-**

CREATE TABLE employee

(

employee\_id INT,

name VARCHAR(25),

months INT,

salary INT,

CONSTRAINT pk\_employee PRIMARY KEY(employee\_id)

);

INSERT INTO employee VALUES(12228, 'Rose', 15, 1968);

INSERT INTO employee VALUES(33645, 'Angela', 1, 3443);

INSERT INTO employee VALUES(45692, 'Frank', 17, 1608);

INSERT INTO employee VALUES(56118, 'Patrick', 7, 1345);

INSERT INTO employee VALUES(59725, 'Lisa', 11, 2330);

INSERT INTO employee VALUES(74197, 'Kimberly', 16, 4372);

INSERT INTO employee VALUES(78454, 'Bonnie', 8, 1771);

INSERT INTO employee VALUES(83565, 'Michael', 6, 2017);

INSERT INTO employee VALUES(98607, 'Todd', 5, 3396);

INSERT INTO employee VALUES(99989, 'Joe', 9, 3573);

SELECT

name

FROM

employee

ORDER BY

name

;

--drop tables

DROP TABLE employee;

**Q-80**

**Ans-**

CREATE TABLE user\_transactions

(

transaction\_id INT,

product\_id INT,

spend DECIMAL(10,2),

transaction\_date DATE

);

INSERT INTO user\_transactions VALUES(1341, 123424, 1500.60, '2019-12-31');

INSERT INTO user\_transactions VALUES(1423, 123424, 1000.20, '2020-12-31');

INSERT INTO user\_transactions VALUES(1623, 123424, 1246.44, '2021-12-31');

INSERT INTO user\_transactions VALUES(1322, 123424, 2145.32, '2022-12-31');

SELECT

DATE\_FORMAT(transaction\_date,'%Y') AS year,

product\_id,

spend AS curr\_year\_spend,

LAG(spend) OVER(ORDER BY DATE\_FORMAT(transaction\_date,'%Y')) AS prev\_year\_spend,

ROUND((spend - LAG(spend) OVER(ORDER BY DATE\_FORMAT(transaction\_date,'%Y')))

\*100.00/LAG(spend) OVER(ORDER BY DATE\_FORMAT(transaction\_date,'%Y')),2) AS yoy\_rate

FROM

user\_transactions

ORDER BY

year

;

--drop tables

DROP TABLE user\_transactions;

**Q-81**

**Ans-**

CREATE TABLE inventory

(

item\_id INT,

item\_type VARCHAR(20),

item\_category VARCHAR(20),

square\_footage DECIMAL(10,2)

);

INSERT INTO inventory VALUES(1374, 'prime\_eligible' , 'mini refrigerator', 68.00);

INSERT INTO inventory VALUES(4245, 'not\_prime', 'standing lamp', 26.40);

INSERT INTO inventory VALUES(2452, 'prime\_eligible', 'television', 85.00);

INSERT INTO inventory VALUES(3255, 'not\_prime', 'side table', 22.60);

INSERT INTO inventory VALUES(1672, 'prime\_eligible', 'laptop', 8.50);

WITH product\_inventory\_summary AS

(

SELECT

item\_type,

SUM(square\_footage) as square\_footage\_required,

COUNT(item\_id) as unique\_item\_count,

500000 as total\_space,

FLOOR(500000/sum(square\_footage))\*sum(square\_footage) as space\_used,

FLOOR(500000/sum(square\_footage))\*COUNT(item\_id) as item\_count

FROM

inventory

GROUP BY

item\_type

)

SELECT

t1.item\_type,

CASE

WHEN t1.item\_type = 'prime\_eligible'

THEN t1.item\_count

ELSE

FLOOR((500000-t2.space\_used)/t1.square\_footage\_required)\*t1.unique\_item\_count

END AS item\_count

FROM

product\_inventory\_summary t1

JOIN product\_inventory\_summary t2 ON t1.item\_type <> t2.item\_type

ORDER BY t1.item\_type DESC

;

--drop tables

DROP TABLE inventory;

**Q-82**

**Ans-**

CREATE TABLE user\_actions

(

user\_id INT,

event\_id INT,

event\_type VARCHAR(20),

event\_date DATE

);

INSERT INTO user\_actions VALUES(445, 7765 , 'sign-in', '2022-05-31');

INSERT INTO user\_actions VALUES(742, 6458, 'sign-in', '2022-06-03');

INSERT INTO user\_actions VALUES(445, 3634, 'like', '2022-06-05');

INSERT INTO user\_actions VALUES(742, 1374, 'comment', '2022-06-05');

INSERT INTO user\_actions VALUES(648, 3124, 'like', '2022-06-18');

SELECT

CAST(DATE\_FORMAT(curr\_month\_ua.event\_date, '%m') AS UNSIGNED) AS month,

count(distinct curr\_month\_ua.user\_id) AS monthly\_active\_users

FROM

user\_actions curr\_month\_ua

WHERE

curr\_month\_ua.event\_type IN ('sign-in', 'like', 'comment')

AND DATE\_FORMAT(curr\_month\_ua.event\_date,'%Y-%m') = '2022-06'

AND EXISTS(

SELECT

\*

FROM

user\_actions last\_month\_ua

WHERE

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

AND last\_month\_ua.event\_type IN ('sign-in', 'like', 'comment')

AND DATE\_FORMAT(curr\_month\_ua.event\_date, '%Y-%m') =

DATE\_FORMAT(last\_month\_ua.event\_date + INTERVAL '1' MONTH, '%Y-%m')

)

GROUP BY

CAST(DATE\_FORMAT(curr\_month\_ua.event\_date, '%m') AS UNSIGNED)

;

--drop tables

DROP TABLE user\_actions;

**Q-83**

**Ans-**

CREATE TABLE search\_frequency

(

searches INT,

num\_users INT

);

INSERT INTO search\_frequency VALUES(1, 2);

INSERT INTO search\_frequency VALUES(2, 2);

INSERT INTO search\_frequency VALUES(3, 3);

INSERT INTO search\_frequency VALUES(4, 1);

WITH cumulative\_sum AS

(

SELECT

\*,

SUM(num\_users) OVER(ORDER BY searches) as cum\_sum,

ROW\_NUMBER() OVER(ORDER BY searches) as row\_num

FROM

search\_frequency

),

max\_cumulative\_sum AS

(

SELECT

MAX(cum\_sum) as max\_cum\_sum

FROM

cumulative\_sum

),

odd\_even\_sum AS

(

SELECT

MIN(row\_num) AS row1

FROM

cumulative\_sum

WHERE

cum\_sum >= (

SELECT

CEILING((max\_cum\_sum)\*0.5)

FROM

max\_cumulative\_sum

)

),

even\_sum AS

(

SELECT

MIN(row\_num) AS row2

FROM

cumulative\_sum

WHERE

cum\_sum >= (

SELECT

CEILING((max\_cum\_sum)\*0.5)+1

FROM

max\_cumulative\_sum

)

)

SELECT

ROUND(

CASE

WHEN (SELECT max\_cum\_sum FROM max\_cumulative\_sum) % 2 = 0

THEN (

(SELECT searches FROM cumulative\_sum WHERE row\_num = (SELECT row1 FROM odd\_even\_sum))

+

(SELECT searches FROM cumulative\_sum WHERE row\_num = (SELECT row2 FROM even\_sum))

)/2.0

ELSE (SELECT searches FROM cumulative\_sum WHERE row\_num = (SELECT row1 FROM odd\_even\_sum))

END

,1) as median

;

--drop tables

DROP TABLE search\_frequency;

**Q-84**

**Ans-**

CREATE TABLE advertiser

(

user\_id VARCHAR(15),

status VARCHAR(15)

);

CREATE TABLE daily\_pay

(

user\_id VARCHAR(15),

paid DECIMAL(10,2)

);

INSERT INTO advertiser VALUES('bing', 'NEW');

INSERT INTO advertiser VALUES('yahoo', 'NEW');

INSERT INTO advertiser VALUES('alibaba', 'EXISTING');

--resuccernt test

INSERT INTO advertiser VALUES('oracle', 'CHURN');

INSERT INTO daily\_pay VALUES('yahoo', '45.00');

INSERT INTO daily\_pay VALUES('alibaba', '100.00');

INSERT INTO daily\_pay VALUES('target', '13.00');

--resuccernt test

INSERT INTO daily\_pay VALUES('oracle', '13.00');

WITH full\_outer\_join\_table AS(

SELECT

dp.user\_id AS user\_id\_dp,

dp.paid,

a.user\_id AS user\_id\_advertiser,

a.status

FROM daily\_pay dp

LEFT OUTER JOIN advertiser a ON dp.user\_id = a.user\_id

UNION

SELECT

dp.user\_id AS user\_id\_dp,

dp.paid,

a.user\_id AS user\_id\_advertiser,

a.status

FROM daily\_pay dp

RIGHT OUTER JOIN advertiser a ON dp.user\_id = a.user\_id

)

SELECT

CASE

WHEN user\_id\_dp IS NULL

THEN user\_id\_advertiser

ELSE user\_id\_dp

END AS user\_id,

CASE

WHEN user\_id\_dp is NULL

THEN

'CHURN'

ELSE

CASE

WHEN status is null

THEN 'NEW'

WHEN status = 'CHURN'

THEN 'RESURRECT'

ELSE 'EXISTING'

END

END AS new\_status

FROM full\_outer\_join\_table full\_table

;

--drop tables

DROP TABLE advertiser;

DROP TABLE daily\_pay;

**Q-85**

**Ans-**

CREATE TABLE server\_utilization

(

server\_id INT,

status\_time TIMESTAMP,

session\_status VARCHAR(10)

);

INSERT INTO server\_utilization VALUES(1, '2022-08-02 10:00:00', 'start');

INSERT INTO server\_utilization VALUES(1, '2022-08-04 10:00:00', 'stop');

INSERT INTO server\_utilization VALUES(2, '2022-08-17 10:00:00', 'start');

INSERT INTO server\_utilization VALUES(2, '2022-08-24 10:00:00', 'stop');

WITH up\_time\_by\_server AS

(

SELECT

server\_id,

session\_status,

status\_time,

CASE

WHEN session\_status = 'stop'

THEN

TIMESTAMPDIFF(SECOND, LAG(status\_time) OVER(PARTITION BY server\_id ORDER BY status\_time), status\_time)/3600

END as up\_time

FROM server\_utilization

)

SELECT

ROUND(sum(up\_time)/24)

FROM

up\_time\_by\_server

WHERE

up\_time is not null

;

--drop tables

DROP TABLE server\_utilization;

**Q-86**

**Ans-**

CREATE TABLE transactions

(

transaction\_id INT,

merchant\_id INT,

credit\_card\_id INT,

amount INT,

transaction\_timestamp TIMESTAMP

);

INSERT INTO transactions VALUES(1, 101, 1, 100, '2022-09-25 12:00:00');

INSERT INTO transactions VALUES(2, 101, 1, 100, '2022-09-25 12:08:00');

INSERT INTO transactions VALUES(3, 101, 1, 100, '2022-09-25 12:28:00');

INSERT INTO transactions VALUES(4, 102, 2, 300, '2022-09-25 12:00:00');

INSERT INTO transactions VALUES(5, 102, 2, 400, '2022-09-25 14:00:00');

WITH trx\_with\_repeadted AS

(

SELECT

credit\_card\_id,

amount,

transaction\_timestamp,

count(\*) OVER(

PARTITION BY credit\_card\_id,amount

ORDER BY transaction\_timestamp

RANGE BETWEEN INTERVAL '10' MINUTE PRECEDING AND CURRENT ROW

) AS moving\_count

FROM

transactions

)

SELECT

COUNT(\*) as payment\_count

FROM trx\_with\_repeadted

WHERE

moving\_count > 1

;

--drop tables

DROP TABLE transactions;

**Q-87**

**Ans-**

CREATE TABLE orders

(

order\_id INT,

customer\_id INT,

trip\_id INT,

status VARCHAR(30),

order\_timestamp TIMESTAMP

);

CREATE TABLE trips

(

dasher\_id INT,

trip\_id INT,

estimated\_delivery\_timestamp TIMESTAMP,

actual\_delivery\_timestamp TIMESTAMP

);

CREATE TABLE customers

(

customer\_id INT,

signup\_timestamp TIMESTAMP

);

INSERT INTO orders VALUES(727424,8472, 100463, 'completed successfully', '2022-06-05 09:12:00');

INSERT INTO orders VALUES(242513, 2341, 100482, 'completed incorrectly', '2022-06-05 14:40:00');

INSERT INTO orders VALUES(141367, 1314, 100362, 'completed incorrectly', '2022-06-07 15:03:00');

INSERT INTO orders VALUES(582193, 5421, 100657, 'never\_received', '2022-07-07 15:22:00');

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

INSERT INTO trips VALUES(101, 100463, '2022-06-05 09:42:00', '2022-06-05 09:38:00');

INSERT INTO trips VALUES(102, 100482, '2022-06-05 15:10:00', '2022-06-05 15:46:00');

INSERT INTO trips VALUES(101, 100362, '2022-06-07 15:33:00', '2022-06-07 16:45:00');

INSERT INTO trips VALUES(102, 100657, '2022-07-07 15:52:00',null);

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

INSERT INTO customers VALUES(8472, '2022-05-30 00:00:00');

INSERT INTO customers VALUES(2341, '2022-06-01 00:00:00');

INSERT INTO customers VALUES(1314, '2022-06-03 00:00:00');

INSERT INTO customers VALUES(1435, '2022-06-05 00:00:00');

INSERT INTO customers VALUES(5421, '2022-06-07 00:00:00');

SELECT

ROUND((COUNT(

CASE

WHEN lower(o.status) <> 'completed successfully'

THEN o.order\_id

END

)\*100.00/COUNT(o.order\_id)),2) as bad\_experience\_pct

FROM orders o

JOIN customers c ON c.customer\_id = o.customer\_id

WHERE

TIMESTAMPDIFF(DAY,o.order\_timestamp,c.signup\_timestamp) < 14

AND DATE\_FORMAT(c.signup\_timestamp,'%Y-%m') = '2022-06'

;

--drop tables

DROP TABLE orders;

DROP TABLE customers;

**Q-88**

**Ans-**

CREATE TABLE scores

(

player\_name VARCHAR(25),

gender VARCHAR(1),

day DATE,

score\_points INT,

CONSTRAINT pk\_scores PRIMARY KEY (gender, day)

);

INSERT INTO scores VALUES('Aron', 'F', '2020-01-01', 17);

INSERT INTO scores VALUES('Alice', 'F', '2020-01-07', 23);

INSERT INTO scores VALUES('Bajrang', 'M', '2020-01-07', 7);

INSERT INTO scores VALUES('Khali' , 'M', '2019-12-25', 11);

INSERT INTO scores VALUES('Slaman', 'M', '2019-12-30', 13);

INSERT INTO scores VALUES('Joe', 'M', '2019-12-31', 3);

INSERT INTO scores VALUES('Jose', 'M', '2019-12-18', 2);

INSERT INTO scores VALUES('Priya', 'F', '2019-12-31', 23);

INSERT INTO scores VALUES('Priyanka', 'F', '2019-12-30', 17);

SELECT

gender,

day,

sum(score\_points) OVER(PARTITION BY gender ORDER BY day) AS total

FROM

scores

;

--drop tables

DROP TABLE scores;

**Q-89**

**Ans-**

CREATE TABLE person

(

id INT,

name VARCHAR(25),

phone\_number VARCHAR(11),

CONSTRAINT pk\_person PRIMARY KEY (id)

);

CREATE TABLE country

(

name VARCHAR(25),

country\_code VARCHAR(3),

CONSTRAINT pk\_country PRIMARY KEY (country\_code)

);

CREATE TABLE calls

(

caller\_id INT,

callee\_id INT,

duration INT

);

INSERT INTO person VALUES(3, 'Jonathan', '051-1234567');

INSERT INTO person VALUES(12, 'Elvis', '051-7654321');

INSERT INTO person VALUES(1, 'Moncef', '212-1234567');

INSERT INTO person VALUES(2, 'Maroua', '212-6523651');

INSERT INTO person VALUES(7, 'Meir', '972-1234567');

INSERT INTO person VALUES(9, 'Rachel', '972-0011100');

INSERT INTO country VALUES('Peru', '51');

INSERT INTO country VALUES('Israel', '972');

INSERT INTO country VALUES('Morocco', '212');

INSERT INTO country VALUES('Germany', '49');

INSERT INTO country VALUES('Ethiopia', '251');

INSERT INTO calls VALUES(1, 9, 33);

INSERT INTO calls VALUES(2, 9, 4);

INSERT INTO calls VALUES(1, 2, 59);

INSERT INTO calls VALUES(3, 12, 102);

INSERT INTO calls VALUES(3, 12, 330);

INSERT INTO calls VALUES(12, 3, 5);

INSERT INTO calls VALUES(7, 9, 13);

INSERT INTO calls VALUES(7, 1, 3);

INSERT INTO calls VALUES(9, 7, 1);

INSERT INTO calls VALUES(1, 7, 7);

WITH receiver\_caller\_calls AS(

SELECT

caller\_id AS caller\_receiver\_id,

duration

FROM

calls

UNION ALL

SELECT

callee\_id AS caller\_receiver\_id,

duration

FROM

calls

),

call\_duration\_avg AS(

SELECT

DISTINCT cn.name,

avg(c.duration) OVER() as global\_average,

avg(c.duration) OVER(PARTITION BY cn.name) as country\_average

FROM

person p

JOIN country cn

ON CAST(SUBSTRING\_INDEX(p.phone\_number, '-', 1) AS UNSIGNED) = CAST(cn.country\_code AS UNSIGNED)

JOIN receiver\_caller\_calls c

ON c.caller\_receiver\_id = p.id

)

SELECT

name

FROM

call\_duration\_avg

WHERE

country\_average > global\_average

;

--drop tables

DROP TABLE person;

DROP TABLE country;

DROP TABLE calls;

**Q-90**

**Ans-**

CREATE TABLE numbers

(

num INT,

frequency INT

);

INSERT INTO numbers VALUES(0, 7);

INSERT INTO numbers VALUES(1, 1);

INSERT INTO numbers VALUES(2, 3);

INSERT INTO numbers VALUES(3, 1);

WITH cumulative\_sum AS

(

SELECT

\*,

SUM(frequency) OVER(ORDER BY num) as cum\_sum,

ROW\_NUMBER() OVER(ORDER BY num) as row\_num

FROM

numbers

),

max\_cumulative\_sum AS

(

SELECT

MAX(cum\_sum) as max\_cum\_sum

FROM

cumulative\_sum

),

odd\_even\_sum AS

(

SELECT

MIN(row\_num) AS row1

FROM

cumulative\_sum

WHERE

cum\_sum >= (

SELECT

CEILING((max\_cum\_sum)\*0.5)

FROM

max\_cumulative\_sum

)

),

even\_sum AS

(

SELECT

MIN(row\_num) AS row2

FROM

cumulative\_sum

WHERE

cum\_sum >= (

SELECT

CEILING((max\_cum\_sum)\*0.5)+1

FROM

max\_cumulative\_sum

)

)

SELECT

ROUND(

CASE

WHEN (SELECT max\_cum\_sum FROM max\_cumulative\_sum) % 2 = 0

THEN (

(SELECT num FROM cumulative\_sum WHERE row\_num = (SELECT row1 FROM odd\_even\_sum))

+

(SELECT num FROM cumulative\_sum WHERE row\_num = (SELECT row2 FROM even\_sum ))

)/2.0

ELSE(

SELECT num FROM cumulative\_sum WHERE row\_num = (SELECT row1 FROM odd\_even\_sum)

)

END

,1) as median

;

--drop tables

DROP TABLE numbers;

**Q-91**

**Ans-**

CREATE TABLE employee

(

employee\_id INT,

department\_id INT,

CONSTRAINT pk\_employee PRIMARY KEY(employee\_id)

);

CREATE TABLE salary

(

id INT,

employee\_id INT,

amount INT,

pay\_date DATE,

CONSTRAINT pk\_salary PRIMARY KEY(id),

CONSTRAINT fk\_employee FOREIGN KEY(employee\_id)

REFERENCES employee(employee\_id)

);

INSERT INTO employee VALUES(1, 1);

INSERT INTO employee VALUES(2, 2);

INSERT INTO employee VALUES(3, 2);

INSERT INTO salary VALUES(1, 1, 9000, '2017-03-31');

INSERT INTO salary VALUES(2, 2, 6000, '2017-03-31');

INSERT INTO salary VALUES(3, 3, 10000, '2017-03-31');

INSERT INTO salary VALUES(4, 1, 7000, '2017-02-28');

INSERT INTO salary VALUES(5, 2, 6000, '2017-02-28');

INSERT INTO salary VALUES(6, 3, 8000, '2017-02-28');

WITH department\_company\_avg\_monthly AS(

SELECT

DISTINCT DATE\_FORMAT(s.pay\_date, '%Y-%m') AS pay\_month,

department\_id,

AVG(amount) OVER(PARTITION BY DATE\_FORMAT(s.pay\_date, '%Y-%m')) as company\_avg,

AVG(amount) OVER(PARTITION BY DATE\_FORMAT(s.pay\_date, '%Y-%m'), department\_id) as department\_avg

FROM

salary s

JOIN employee e ON s.employee\_id = e.employee\_id

)

SELECT

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

department\_company\_avg\_monthly

ORDER BY

department\_id

;

--drop tables

DROP TABLE salary;

DROP TABLE employee;

**Q-92**

**Ans-**

CREATE TABLE activity

(

player\_id INT,

device\_id INT,

event\_date DATE,

games\_played INT,

CONSTRAINT pk\_activity PRIMARY KEY(player\_id, event\_date)

);

INSERT INTO activity VALUES(1, 2, '2016-03-01', 5);

INSERT INTO activity VALUES(1, 2, '2016-03-02', 6);

INSERT INTO activity VALUES(2, 3, '2017-06-25', 1);

INSERT INTO activity VALUES(3, 1, '2016-03-01', 0);

INSERT INTO activity VALUES(3, 4, '2016-07-03', 5);

WITH retention\_data AS(

SELECT

curr\_day.event\_date,

COUNT(DISTINCT curr\_day.player\_id) AS retention\_player\_count

FROM

activity curr\_day

WHERE

EXISTS(

SELECT

\*

FROM

activity next\_day

WHERE

curr\_day.player\_id = next\_day.player\_id

AND next\_day.event\_date = curr\_day.event\_date + INTERVAL '1' DAY

)

GROUP BY

curr\_day.event\_date

),

player\_signup\_data AS(

SELECT

DISTINCT player\_id,

FIRST\_VALUE(event\_date) OVER(PARTITION BY player\_id ORDER BY event\_date) as signup\_date

FROM

activity

),

daily\_player\_data AS(

SELECT

signup\_date,

COUNT(DISTINCT player\_id) AS daily\_player\_count

FROM

player\_signup\_data

GROUP BY

signup\_date

)

SELECT

dpd.signup\_date AS install\_dt,

dpd.daily\_player\_count AS installs,

ROUND(IFNULL(rd.retention\_player\_count,0)/dpd.daily\_player\_count,2) AS Day1\_retention

FROM

daily\_player\_data dpd

LEFT JOIN retention\_data rd ON dpd.signup\_date = rd.event\_date

;

--drop tables

DROP TABLE activity;

**Q-93**

**Ans-**

CREATE TABLE players

(

player\_id INT,

group\_id INT,

CONSTRAINT pk\_players PRIMARY KEY(player\_id)

);

CREATE TABLE matches

(

match\_id INT,

first\_player INT,

second\_player INT,

first\_score INT,

second\_score INT,

CONSTRAINT pk\_matches PRIMARY KEY(match\_id)

);

INSERT INTO players VALUES(15, 1);

INSERT INTO players VALUES(25, 1);

INSERT INTO players VALUES(30, 1);

INSERT INTO players VALUES(45, 1);

INSERT INTO players VALUES(10, 2);

INSERT INTO players VALUES(35, 2);

INSERT INTO players VALUES(50, 2);

INSERT INTO players VALUES(20, 3);

INSERT INTO players VALUES(40, 3);

INSERT INTO matches VALUES(1, 15, 45, 3, 0);

INSERT INTO matches VALUES(2, 30, 25, 1, 2);

INSERT INTO matches VALUES(3, 30, 15, 2, 0);

INSERT INTO matches VALUES(4, 40, 20, 5, 2);

INSERT INTO matches VALUES(5, 35, 50, 1, 1);

WITH player\_score AS(

SELECT

p.group\_id,

p.player\_id,

SUM(CASE

WHEN p.player\_id = m.first\_player

THEN m.first\_score

WHEN p.player\_id = m.second\_player

THEN m.second\_score

END) AS score

FROM

players p

JOIN matches m ON p.player\_id = m.first\_player OR p.player\_id = m.second\_player

GROUP BY

p.group\_id,

p.player\_id

),

ranked\_player AS(

SELECT

group\_id,

player\_id,

score,

DENSE\_RANK() OVER (PARTITION BY group\_id ORDER BY score DESC,player\_id) AS player\_rank

FROM

player\_score

)

SELECT

group\_id,

player\_id

FROM

ranked\_player

WHERE

player\_rank = 1

;

--drop tables

DROP TABLE players;

DROP TABLE matches;

**Q-94**

**Ans-**

CREATE TABLE student

(

student\_id INT,

student\_name VARCHAR(25),

CONSTRAINT pk\_student PRIMARY KEY(student\_id)

);

CREATE TABLE exam

(

exam\_id INT,

student\_id INT,

score INT,

CONSTRAINT pk\_exam PRIMARY KEY(exam\_id, student\_id)

);

INSERT INTO student VALUES(1, 'Daniel');

INSERT INTO student VALUES(2, 'Jade');

INSERT INTO student VALUES(3, 'Stella');

INSERT INTO student VALUES(4, 'Jonathan');

INSERT INTO student VALUES(5, 'Will');

INSERT INTO exam VALUES(10, 1, 70);

INSERT INTO exam VALUES(10, 2, 80);

INSERT INTO exam VALUES(10, 3, 90);

INSERT INTO exam VALUES(20, 1, 80);

INSERT INTO exam VALUES(30, 1, 70);

INSERT INTO exam VALUES(30, 3, 80);

INSERT INTO exam VALUES(30, 4, 90);

INSERT INTO exam VALUES(40, 1, 60);

INSERT INTO exam VALUES(40, 2, 70);

INSERT INTO exam VALUES(40, 4, 80);

WITH exam\_highest\_lowest AS(

SELECT

\*,

FIRST\_VALUE(score) OVER(PARTITION BY exam\_id ORDER BY score) as exam\_lowest,

FIRST\_VALUE(score) OVER(PARTITION BY exam\_id ORDER BY score DESC) as exam\_highest

FROM

exam

),

student\_highest\_lowest AS(

SELECT

DISTINCT student\_id

FROM

exam\_highest\_lowest

WHERE

score = exam\_lowest

OR score = exam\_highest

)

SELECT

student\_id,

student\_name

FROM

student s

WHERE

EXISTS(

SELECT

\*

FROM

exam e

WHERE

e.student\_id = s.student\_id

)

AND s.student\_id NOT IN(

SELECT

student\_id

FROM

student\_highest\_lowest

)

;

--drop tables

DROP TABLE student;

DROP TABLE exam;

**Q-95**

**Ans-**

CREATE TABLE student

(

student\_id INT,

student\_name VARCHAR(25),

CONSTRAINT pk\_student PRIMARY KEY(student\_id)

);

CREATE TABLE exam

(

exam\_id INT,

student\_id INT,

score INT,

CONSTRAINT pk\_exam PRIMARY KEY(exam\_id, student\_id)

);

INSERT INTO student VALUES(1, 'Daniel');

INSERT INTO student VALUES(2, 'Jade');

INSERT INTO student VALUES(3, 'Stella');

INSERT INTO student VALUES(4, 'Jonathan');

INSERT INTO student VALUES(5, 'Will');

INSERT INTO exam VALUES(10, 1, 70);

INSERT INTO exam VALUES(10, 2, 80);

INSERT INTO exam VALUES(10, 3, 90);

INSERT INTO exam VALUES(20, 1, 80);

INSERT INTO exam VALUES(30, 1, 70);

INSERT INTO exam VALUES(30, 3, 80);

INSERT INTO exam VALUES(30, 4, 90);

INSERT INTO exam VALUES(40, 1, 60);

INSERT INTO exam VALUES(40, 2, 70);

INSERT INTO exam VALUES(40, 4, 80);

WITH exam\_highest\_lowest AS(

SELECT

\*,

FIRST\_VALUE(score) OVER(PARTITION BY exam\_id ORDER BY score) as exam\_lowest,

FIRST\_VALUE(score) OVER(PARTITION BY exam\_id ORDER BY score DESC) as exam\_highest

FROM

exam

),

student\_highest\_lowest AS(

SELECT

DISTINCT student\_id

FROM

exam\_highest\_lowest

WHERE

score = exam\_lowest

OR score = exam\_highest

)

SELECT

student\_id,

student\_name

FROM

student s

WHERE

EXISTS(

SELECT

\*

FROM

exam e

WHERE

e.student\_id = s.student\_id

)

AND s.student\_id NOT IN(

SELECT

student\_id

FROM

student\_highest\_lowest

)

;

--drop tables

DROP TABLE student;

DROP TABLE exam;

**Q-96**

**Ans-**

CREATE TABLE songs\_history

(

history\_id INT,

user\_id INT,

song\_id INT,

song\_plays INT

);

CREATE TABLE songs\_weekly

(

user\_id INT,

song\_id INT,

listen\_time TIMESTAMP

);

INSERT INTO songs\_history VALUES(10011, 777, 1238, 11);

INSERT INTO songs\_history VALUES(12452, 695, 4520, 1);

INSERT INTO songs\_weekly VALUES(777, 1238, '2022-08-01 12:00:00');

INSERT INTO songs\_weekly VALUES(695, 4520, '2022-08-04 08:00:00');

INSERT INTO songs\_weekly VALUES(125, 9630, '2022-08-04 16:00:00');

INSERT INTO songs\_weekly VALUES(695, 9852, '2022-08-07 12:00:00');

WITH all\_song\_list AS(

SELECT

sw.user\_id,

sw.song\_id,

count(\*) as song\_plays

FROM

songs\_weekly sw

WHERE

STR\_TO\_DATE(listen\_time,'%Y-%m-%d') <= STR\_TO\_DATE('08/04/2022','%m/%d/%Y')

GROUP BY

sw.user\_id,

sw.song\_id

UNION ALL

SELECT

sh.user\_id,

sh.song\_id,

sh.song\_plays

FROM

songs\_history sh

)

SELECT

user\_id,

song\_id,

sum(song\_plays) as song\_plays

FROM

all\_song\_list

GROUP BY

user\_id,

song\_id

ORDER BY

song\_plays DESC

;

--drop tables

DROP TABLE songs\_history;

DROP TABLE songs\_weekly;

**Q-97**

**Ans-**

CREATE TABLE emails

(

email\_id INT,

user\_id INT,

signup\_date TIMESTAMP

);

CREATE TABLE texts

(

text\_id INT,

email\_id INT,

signup\_action VARCHAR(20)

);

INSERT INTO emails VALUES(125, 7771, STR\_TO\_DATE('06/14/2022 00:00:00', '%m/%d/%Y %H:%i:%s'));

INSERT INTO emails VALUES(236, 6950, STR\_TO\_DATE('07/01/2022 00:00:00', '%m/%d/%Y %H:%i:%s'));

INSERT INTO emails VALUES(433, 1052, STR\_TO\_DATE('07/09/2022 00:00:00', '%m/%d/%Y %H:%i:%s'));

INSERT INTO texts VALUES(6878, 125, 'Confirmed');

INSERT INTO texts VALUES(6920, 236, 'Not Confirmed');

INSERT INTO texts VALUES(6994, 236, 'Confirmed');

WITH email\_with\_action AS(

SELECT

DISTINCT e.email\_id,

t.signup\_action

FROM

emails e

LEFT JOIN texts t ON e.email\_id = t.email\_id

),

email\_confirm\_table AS(

SELECT

email\_id,

GROUP\_CONCAT(signup\_action ORDER BY signup\_action) AS action,

position('Confirmed' in GROUP\_CONCAT(signup\_action ORDER BY signup\_action)) as pos

FROM

email\_with\_action

GROUP BY

email\_id

)

SELECT

ROUND(

COUNT(

CASE

WHEN pos = 1

THEN email\_id

END

)\*1.00/

count(\*)

,2)as confirm\_rate

FROM email\_confirm\_table

;

--drop tables

DROP TABLE emails;

DROP TABLE texts;

**Q-98**

**Ans-**

CREATE TABLE tweets

(

tweet\_id INT,

user\_id INT,

tweet\_date TIMESTAMP

);

INSERT INTO tweets VALUES(214252, 111, STR\_TO\_DATE('06/01/2022 12:00:00', '%m/%d/%Y %H:%i:%s'));

INSERT INTO tweets VALUES(739252, 111, STR\_TO\_DATE('06/01/2022 12:00:00', '%m/%d/%Y %H:%i:%s'));

INSERT INTO tweets VALUES(846402, 111, STR\_TO\_DATE('06/02/2022 12:00:00', '%m/%d/%Y %H:%i:%s'));

INSERT INTO tweets VALUES(241425, 254, STR\_TO\_DATE('06/02/2022 12:00:00', '%m/%d/%Y %H:%i:%s'));

INSERT INTO tweets VALUES(137374, 111, STR\_TO\_DATE('06/04/2022 12:00:00', '%m/%d/%Y %H:%i:%s'));

WITH tweet\_per\_day\_by\_user AS

(

SELECT

user\_id,

tweet\_date,

COUNT(\*) as tweet\_count

FROM

tweets

group by

user\_id,

tweet\_date

)

SELECT

user\_id,

tweet\_date,

ROUND(

AVG(tweet\_count) OVER( PARTITION BY user\_id ORDER BY tweet\_date

ROWS BETWEEN 2 PRECEDING and CURRENT ROW )

,2) as rolling\_avg\_3d

FROM tweet\_per\_day\_by\_user;

--drop tables

DROP TABLE tweets;

**Q-99**

**Ans-**

CREATE TABLE activities

(

activity\_id INT,

user\_id INT,

activity\_type VARCHAR(10),

time\_spent DECIMAL(5,2),

activity\_date TIMESTAMP

);

CREATE TABLE age\_breakdown

(

user\_id INT,

age\_bucket VARCHAR(10)

);

INSERT INTO activities VALUES(7274, 123, 'open', 4.50, STR\_TO\_DATE('06/22/2022 12:00:00', '%m/%d/%Y %H:%i:%s'));

INSERT INTO activities VALUES(2425, 123, 'send', 3.50, STR\_TO\_DATE('06/22/2022 12:00:00', '%m/%d/%Y %H:%i:%s'));

INSERT INTO activities VALUES(1413, 456, 'send', 5.67, STR\_TO\_DATE('06/23/2022 12:00:00', '%m/%d/%Y %H:%i:%s'));

INSERT INTO activities VALUES(1414, 789, 'chat', 11.00, STR\_TO\_DATE('06/25/2022 12:00:00', '%m/%d/%Y %H:%i:%s'));

INSERT INTO activities VALUES(2536, 456, 'open', 3.00, STR\_TO\_DATE('06/25/2022 12:00:00', '%m/%d/%Y %H:%i:%s'));

INSERT INTO age\_breakdown VALUES(123, '31-35');

INSERT INTO age\_breakdown VALUES(456, '26-30');

INSERT INTO age\_breakdown VALUES(789, '21-25');

SELECT

ab.age\_bucket,

ROUND(

sum(

CASE

WHEN a.activity\_type = 'send'

THEN

a.time\_spent

END

)\*100.0/sum(

CASE

WHEN a.activity\_type in ('open','send')

THEN

a.time\_spent

END

)

,2) AS send\_perc,

ROUND(

sum(

CASE

WHEN a.activity\_type = 'open'

THEN

a.time\_spent

END

)\*100.0/sum(

CASE

WHEN a.activity\_type in ('open','send')

THEN

a.time\_spent

END

)

,2) AS open\_perc

FROM

activities a

JOIN age\_breakdown ab ON a.user\_id = ab.user\_id

WHERE

a.activity\_type in ('open','send')

GROUP BY

ab.age\_bucket

;

--drop tables

DROP TABLE activities;

DROP TABLE age\_breakdown;

**Q-100**

**Ans-**

CREATE TABLE personal\_profiles

(

profile\_id INT,

name VARCHAR(30),

followers INT

);

CREATE TABLE employee\_company

(

personal\_profile\_id INT,

company\_id INT

);

CREATE TABLE company\_pages

(

company\_id INT,

name VARCHAR(30),

followers INT

);

INSERT INTO personal\_profiles VALUES(1, 'Nick Singh', 92000);

INSERT INTO personal\_profiles VALUES(2, 'Zach Wilson', 199000);

INSERT INTO personal\_profiles VALUES(3, 'Daliana Liu', 171000);

INSERT INTO personal\_profiles VALUES(4, 'Ravit Jain', 107000);

INSERT INTO personal\_profiles VALUES(5, 'Vin Vashishta', 139000);

INSERT INTO personal\_profiles VALUES(6, 'Susan Wojcicki', 39000);

INSERT INTO employee\_company VALUES(1, 4);

INSERT INTO employee\_company VALUES(1, 9);

INSERT INTO employee\_company VALUES(2, 2);

INSERT INTO employee\_company VALUES(3, 1);

INSERT INTO employee\_company VALUES(4, 3);

INSERT INTO employee\_company VALUES(5, 6);

INSERT INTO employee\_company VALUES(6, 5);

INSERT INTO company\_pages VALUES(1 , 'The Data Science Podcast', 8000);

INSERT INTO company\_pages VALUES(2, 'Airbnb', 700000);

INSERT INTO company\_pages VALUES(3, 'The Ravit Show', 6000);

INSERT INTO company\_pages VALUES(4, 'DataLemur', 200);

INSERT INTO company\_pages VALUES(5, 'YouTube', 16000000);

INSERT INTO company\_pages VALUES(6, 'DataScience.Vin', 4500);

INSERT INTO company\_pages VALUES(9, 'Ace The Data Science Interview', 4479);

WITH profile\_with\_max\_company\_follower AS(

SELECT

pp.profile\_id,

pp.name,

pp.followers,

cp.name AS company\_name,

cp.followers AS company\_follower,

max(cp.followers) OVER(PARTITION BY pp.profile\_id) as max\_company\_follower

FROM

personal\_profiles pp

JOIN employee\_company ec ON pp.profile\_id = ec.personal\_profile\_id

JOIN company\_pages cp ON cp.company\_id = ec.company\_id

ORDER BY

pp.name

)

SELECT

DISTINCT profile\_id

FROM

profile\_with\_max\_company\_follower

WHERE

followers > max\_company\_follower

ORDER BY

profile\_id

;

--drop tables

DROP TABLE personal\_profiles;

DROP TABLE employee\_company;

DROP TABLE company\_pages;

**Q-101**

**Ans-**

create table if not exists UserActivity

(

username VARCHAR(50),

activity varchar(50),

startDate date,

endDate date

);

insert INTO UserActivity VALUES ('Alice','Travel','2020-02-12','2020-02-20'),('Alice','Dancing','2020-02-21','2020-02-23'),('Alice','Travel','2020-02-24','2020-02-28'),('Bob','Travel','2020-02-11','2020-02-18');

select \* from UserActivity;

--Write an SQL query to show the second most recent activity of each user. If the user only has one activity, return that one. A user cannot perform more than one activity at the same time. Return the result table in any order.

select distinct username, activity, startDate, endDate

from

(select u.\*,

rank() over (partition by username order by startDate desc) as rnk,

count(activity) over (partition by username) as num

from UserActivity u) t

where (num <> 1 and rnk = 2) or (num = 1 and rnk = 1);

**Q-102**

**Ans-**

Table: UserActivity

--Write an SQL query to show the second most recent activity of each user.If the user only has one activity, return that one. A user cannot perform more than one activity at the same time. Return the result table in any order.

select distinct username, activity, startDate, endDate

from

(select u.\*,

rank() over (partition by username order by startDate desc) as rnk,

count(activity) over (partition by username) as num

from UserActivity u) t

where (num <> 1 and rnk = 2) or (num = 1 and rnk = 1);

**Q-103**

**Ans-**

--STUDENTS table

create table if not exists STUDENTS

(

id int,

name VARCHAR(50),

marks int

);

insert into STUDENTS VALUES (1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),(3,'Belvet',84);

select \* from STUDENTS;

--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.

SELECT NAME FROM STUDENTS WHERE MARKS > 75 ORDER BY RIGHT(NAME,3), ID ASC;

**Q-104**

**Ans-**

--Employee table

create table if not exists Employee

(

employee\_id int,

name VARCHAR(50),

months int,

salary int

);

Insert into Employee 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 \* from Employee;

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

SELECT name FROM Employee ORDER BY name;

**Q-105**

**Ans-**

--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.

SELECT name FROM Employee WHERE salary > 2000 AND months < 10 ORDER BY employee\_id;

**Q-106**

**Ans-**

--TRIANGLES table

create table if not exists TRIANGLES

(

A int,

B int,

C int

);

insert into TRIANGLES VALUES (20,20,23),(20,20,20),(20,21,22),(13,14,30);

select \* from TRIANGLES;

--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.

SELECT CASE

WHEN A + B <= C OR A + C <= B OR B + C <= A THEN 'Not A Triangle'

WHEN A = B AND B = C THEN 'Equilateral'

WHEN A = B OR B = C OR A = C THEN 'Isosceles'

ELSE 'Scalene'

END

FROM TRIANGLES;

**Q-107**

**Ans-**

--EMPLOYEES table

create table if not exists EMPLOYEES

(

id int,

name VARCHAR(50),

salary int

);

insert into EMPLOYEES VALUES (1,'Kristeen',1420), (2,'Ashley',2006), (3,'Julia',2210), (4,'Maria',3000);

select \* from EMPLOYEES;

--Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries), and round it up to the next integer.

select ceil(avg(salary) - avg(replace(salary, '0', ''))) from EMPLOYEES;

**Q-108**

**Ans-**

--Employee table

select MAX(salary\*months), COUNT(\*) from Employee where (salary \* months) >= (select MAX(salary \* months) from Employee);

**Q-109**

**Ans-**

--OCCUPATIONS table

create table if not exists OCCUPATIONS

(

Name VARCHAR(50),

Occupation VARCHAR(50)

);

Insert into OCCUPATION VALUES ('Samantha','Doctor'),('Julia','Actor'),('Maria','Actor'),('Meera','Singer'),('Ashely','Professor'),('Ketty','Professor'),('Christeen','Professor'),('Jane','Actor'),('Jenny','Doctor'),('Priya','Singer');

select \* from OCCUPATIONS;

--Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order,

(

SELECT CONCAT(NAME, '(', SUBSTRING(Occupation, 1, 1), ')') as THETEXT, '1' as SELECTNUMBER

FROM OCCUPATIONS

)

UNION ALL

(

SELECT CONCAT('There are total ', COUNT(\*),' ', Occupation, (IF (COUNT(\*) > 1, 's',''))) as THETEXT, '2' as SELECTNUMBER

FROM OCCUPATIONS GROUP BY Occupation

)

ORDER BY SELECTNUMBER ASC, THETEXT ASC;

**Q-110**

**Ans-**

--Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

select

Doctor,

Professor,

Singer,

Actor

from (

select

NameOrder,

max(case Occupation when 'Doctor' then Name end) as Doctor,

max(case Occupation when 'Professor' then Name end) as Professor,

max(case Occupation when 'Singer' then Name end) as Singer,

max(case Occupation when 'Actor' then Name end) as Actor

from (

select

Occupation,

Name,

row\_number() over(partition by Occupation order by Name ASC) as NameOrder

from OCCUPATIONS

) as NameLists

group by NameOrder

) as Names;

**Q-111**

**Ans-**

--Table, BST,

create table if not exists BST

(

N int,

P int

);

insert into BST VALUES(1,2),(3,2),(6,8),(9,8),(2,5),(8,5),(5,null);

select \* from BST;

--Write a query to find the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:

--● Root: If node is root node.

--● Leaf: If node is leaf node.

--● Inner: If node is neither root nor leaf node.

SELECT N,

CASE

WHEN P IS NULL THEN 'Root'

WHEN N IN (SELECT P FROM BST) THEN 'Inner'

ELSE 'Leaf'

END

FROM BST

ORDER by N;

**Q-112**

**Ans-**

--Given the table schemas below, write a query to print the company\_code, founder name, total number of lead managers, total number of senior managers, total number of managers, and total number of employees. Order your output by ascending company\_code.

--Company

create table if not exists Company

(

company\_code VARCHAR(50),

founder VARCHAR(50)

);

insert into Company VALUES ('C1','Monika'),('C2','Samantha');

select \* from Company;

--Lead\_Manager

create table if not exists Lead\_Manager

(

lead\_manager\_code VARCHAR(50),

company\_code VARCHAR(50)

);

insert into Lead\_Manager VALUES ('LM1','C1'),('LM2','C2');

select \* from Lead\_Manager;

--Senior\_Manager

create table if not exists Senior\_Manager

(

senior\_manager\_code VARCHAR(50),

lead\_manager\_code VARCHAR(50),

company\_code VARCHAR(50)

);

insert into Senior\_Manager VALUES ('SM1','LM1','C1'),('SM2','LM1','C1'),('SM3','LM2','C2');

select \* from Senior\_Manager;

--Manager Table:

create table if not exists Manager

(

manager\_code VARCHAR(50),

senior\_manager\_code VARCHAR(50),

lead\_manager\_code VARCHAR(50),

company\_code VARCHAR(50)

);

insert into Manager VALUES ('M1','SM1','LM1','C1'),('M2','SM3','LM2','C2'),('M3','SM3','LM2','C2');

select \* from Manager;

--Employee:

create table if not exists Employee

(

employee\_code VARCHAR(50),

manager\_code VARCHAR(50),

senior\_manager\_code VARCHAR(50),

lead\_manager\_code VARCHAR(50),

company\_code VARCHAR(50)

);

Insert into Employee VALUES ('E1','M1','SM1','LM1','C1'), ('E2','M1','SM1','LM1','C1'), 'E3','M2','SM3','LM2','C2'), ('E4','M3','SM3','LM2','C2');

select \* from Employee;

SELECT c.company\_code, c.founder, COUNT(DISTINCT e.lead\_manager\_code), COUNT(DISTINCT e.senior\_manager\_code), COUNT(DISTINCT e.manager\_code), COUNT(DISTINCT e.employee\_code) FROM Company c

JOIN Employee e ON c.company\_code = e.company\_code GROUP BY c.company\_code, c.founder ORDER BY c.company\_code;

**Q-113**

**Ans-**

--Write a query to print all prime numbers less than or equal to 1000. Print your result on a single line,and use the ampersand () character as your separator (instead of a space).

select listagg(Prime\_Number,'&') within group(order by Prime\_Number)

from (select L Prime\_Number from

(select Level L

from Dual

connect by Level <= 1000),

(select Level M

from Dual

connect by Level <= 1000)

where M <= L

group by L

having count(case when L/M = trunc(L/M) then 'Y' end) = 2

order by L);

**Q-114**

**Ans-**

--P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

--\*

--\* \*

--\* \* \*

--\* \* \* \*

--\* \* \* \* \*

--Write a query to print the pattern P(20).

SELECT SYS\_CONNECT\_BY\_PATH(NULL, '\* ') FROM DUAL CONNECT BY ROWNUM <= 20 ORDER BY 1 DESC;

**Q-115**

--P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

--\* \* \* \* \*

--\* \* \* \*

--\* \* \*

--\* \*

--\*

--Write a query to print the pattern P(20).

SELECT SYS\_CONNECT\_BY\_PATH(NULL, '\* ') FROM DUAL CONNECT BY ROWNUM <= 20 ORDER BY 1 DESC;

SET @no\_of\_lines = 5 + 1;

SELECT REPEAT('\* ', @no\_of\_lines := @no\_of\_lines -1)

FROM INFORMATION\_SCHEMA.TABLES

WHERE @no\_of\_lines > 0;

**Q-116**

**Ans-**

--Functions,

create table if not exists Functions

(

X int,

Y int

);

insert into Functions VALUES (20,20),(20,20),(20,21),(23,22),(22,23),(21,20);

select \* from Functions;

--Write a query to output all such symmetric pairs in ascending order by the value of X. List the row such that X1 ≤ Y1.

SELECT f1.X, f1.Y FROM Functions AS f1

WHERE f1.X = f1.Y AND

(SELECT COUNT(\*) FROM Functions WHERE X = f1.X AND Y = f1.Y) > 1

UNION

SELECT f1.X, f1.Y from Functions AS f1

WHERE EXISTS(SELECT X, Y FROM Functions WHERE f1.X = Y AND f1.Y = X AND f1.X < X)

ORDER BY X;

**Q-117**

**Ans-**

--STUDENTS TABLE

--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 if not exists STUDENTS

(

id int,

name VARCHAR(50),

marks int

);

insert into STUDENTS VALUES (1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),(3,'Belvet',84);

select \* from STUDENTS;

--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.

SELECT NAME FROM STUDENTS WHERE MARKS > 75 ORDER BY RIGHT(NAME,3), ID ASC;

**Q-118**

**Ans-**

--Employee table

create table if not exists Employee

(

employee\_id int,

name VARCHAR(50),

months int,

salary int

);

insert into Employee 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 \* from Employee;

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

SELECT name FROM Employee ORDER BY name;

**Q-119**

**Ans-**

--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.

SELECT name FROM Employee WHERE salary > 2000 AND months < 10 ORDER BY employee\_id;

**Q-120**

**Ans-**

----TRIANGLES table

create table if not exists TRIANGLES

(

A int,

B int,

C int

);

insert into TRIANGLES VALUES (20,20,23),(20,20,20),(20,21,22),(13,14,30);

select \* from TRIANGLES;

--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.

SELECT CASE

WHEN A + B <= C OR A + C <= B OR B + C <= A THEN 'Not A Triangle'

WHEN A = B AND B = C THEN 'Equilateral'

WHEN A = B OR B = C OR A = C THEN 'Isosceles'

ELSE 'Scalene'

END

FROM TRIANGLES;

**Q-121**

**Ans-**

--user\_transactions Table:

create table if not exists user\_transactions

(

transaction\_id int,

product\_id int,

spend decimal,

transaction\_date DATETIME

);

insert into user\_transactions VALUES (1341,123424,1500.60,'12/31/2019 12:00:00'),(1423,123424,1000.20,'12/31/202012:00:00')(1623,123424,1246.44,'12/31/202112:00:00')(1322,123424,2145.32,'12/31/2022 12:00:00');

select \* from user\_transactions;

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

**Q-122**

**Ans-**

--inventory table:

create table if not exists inventory

(

item\_id int,

item\_type VARCHAR(50),

item\_category VARCHAR(50),

square\_footage DECIMAL

);

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);

select \* from inventory;

--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.

SELECT

item\_type,

SUM(square\_footage) AS total\_sqft,

COUNT(\*) AS item\_count

FROM inventory

GROUP BY item\_type;

**Q-123**

**Ans-**

--user\_actions Table:

create table if not exists 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','05/31/2022 12:00:00'),(742,6458,'sign-in','06/03/2022 12:00:00'),(445,3634,'like','06/05/202212:00:00'),(742,1374,'comment','06/05/202212:00:00'),(648,3124,'like','06/18/2022 12:00:00');

select \* from user\_actions;

--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).

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);

**Q-124**

**Ans-**

--search\_frequency Table:

create table if not exists search\_frequency

(

searches int,

num\_users int

);

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

select \* from search\_frequency;

--Write a query to report the median of searches made by a user. Round the median to one decimal point

WITH RECURSIVE cte AS (

SELECT searches, num\_users as NU FROM search\_frequency

UNION ALL

SELECT cte.searches,

cte.NU - 1

FROM cte WHERE NU > 0

)

select PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY searches) AS median FROM cte

WHERE nu > 0;

**Q-125**

**Ans-**

--advertiser Table:

create table if not exists advertiser

(

user\_id VARCHAR(50),

status VARCHAR(50)

);

insert into advertiser VALUES ('bing','NEW'),('yahoo','NEW'),('alibaba','EXISTING');

select \* from advertiser;

--daily\_pay Table:

create table if not exists daily\_pay

(

user\_id VARCHAR(50),

paid decimal

);

insert into daily\_pay VALUES ('yahoo',45.00),('alibaba',100.00),('target',13.00);

select \* from daily\_pay;

--Write a query to update the Facebook advertiser's status using the daily\_pay table. Advertiser is two-column table containing the user id and their payment status based on the last payment an daily\_pay table has current information about their payment. Only advertisers who paid will show up in this table.Output the user id and current payment status sorted by the user id.

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;

**Q-126**

**Ans-**

--server\_utilization Table:

create table if not exists server\_utilization

(

server\_id int,

status\_time TIMESTAMP,

session\_status VARCHAR(50)

);

insert into server\_utilization VALUES(1,'08/02/2022 10:00:00','start'),(1,'08/04/2022 10:00:00','stop '),(2,'08/17/2022 10:00:00','start'),(2,'08/24/2022 10:00:00','stop');

select \* from server\_utilization;

--Write a query that calculates the total time that the fleet of servers was running. The output should be in units of full days.

**Q-127**

**Ans-**

--transactions

create table if not exists transactions

(

transaction\_id int,

merchant\_id int,

credit\_card\_id INT,

amount int,

transaction\_timestamp datetime

);

insert into transactions values(1,101,1,100,'09/25/202212:00:00'),(2,101,1,100,'09/25/2022'), (3,101,1,100,'09/25/202212:28:00'), (4,102,2,300,'09/25/202212:00:00'),(6,102,2,400,'09/25/2022 14:00:00');

select \* from transactions;

--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.

WITH payments AS (

SELECT

merchant\_id,

EXTRACT(EPOCH FROM transaction\_timestamp -

LAG(transaction\_timestamp) OVER(

PARTITION BY merchant\_id, credit\_card\_id, amount

ORDER BY transaction\_timestamp)

)/60 AS minute\_difference

FROM transactions)

SELECT COUNT(merchant\_id) AS payment\_count

FROM payments

WHERE minute\_difference <= 10;

**Q-128**

**Ans-**

--orders Table:

create table if not exists orders

(

order\_id int,

customer\_id int,

trip\_id INT,

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

order\_timestamp timestamp

);

insert into orders VALUES (727424,8472,100463,'completed successfully','06/05/2022 09:12:00'),(242513,2341,100482,'completed incorrectly','06/05/202214:40:00'),(141367,1314,100362,'completd incorrectly','06/07/202215:03:00'),(582193,5421,100657,'never\_received','07/07/202215:22:00'),(253613,1314,100213,'completed successfully','06/12/2022 13:43:00');

select \* from orders;

--trips Table:

create table if not exists trips

(

dasher\_id int,

trip\_id int,

estimated\_delivery\_timestamp timestamp,

actual\_delivery\_timestamp timestamp

);

insert into trips VALUES (101,100463,'06/05/2022 09:42:00','06/05/2022 09:38:00'),(102,100482,'06/05/2022 15:10:00','06/05/202215:46:00'),(101,100362,'06/07/202215:33:00','06/07/202216:45:00'),(102,100657,'07/07/2022 15:52:00','-'),(103,100213,'06/12/2022 14:13:00','06/12/2022 14:10:00');

select \* from trips;

--customers Table:

create table if not exists customers

(

customer\_id int,

signup\_timestamp timestamp

);

insert into customers VALUES (8472,'05/30/2022 00:00:00'),(2341,'06/01/2022 00:00:00'),(1314,'06/03/2022 00:00:00'),(1435,'06/05/2022 00:00:00'),(5421,'06/07/2022 00:00:00');

select \* from customers;

--Write a query to find the bad experience rate in the first 14 days for new users who signed up in June 2022. Output the percentage of bad experience rounded to 2 decimal places.

**Q-129**

**Ans-**

-- Scores

create table if not exists Scores

(

player\_name VARCHAR(50),

gender VARCHAR(50),

day DATE,

score\_points int,

constraint pk 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);

--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. The query result format is in the following example.

select s1.gender, s1.day, sum(s2.score\_points) as total from Scores s1, Scores s2

where s1.gender = s2.gender and s1.day >= s2.day

group by s1.gender, s1.day

order by s1.gender, s1.day;

**Q-130.**

**Ans-**

--Table Person:

create table if not exists Person

(

id int,

name VARCHAR(50),

phone\_number VARCHAR(50),

constraint pk PRIMARY KEY (id)

);

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');

select \* from Person;

--Country table:

create table if not exists Country

(

name VARCHAR(50),

country\_code VARCHAR(50),

constraint pk PRIMARY KEY (country\_code)

);

insert into Country VALUES ('Peru',51),('Israel',972),('Morocco',212),('Germany',49),('Ethiopia',251);

select \* from Country;

--Table Calls:

create table if not exists Calls

(

caller\_id int,

callee\_id int,

duration int

);

insert into Calls 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);

select \* from Calls;

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

--Return the result table in any order.

SELECT

co.name AS country

FROM

Person p

JOIN

Country co

ON SUBSTRING(phone\_number,1,3) = country\_code

JOIN

Calls c

ON p.id IN (c.caller\_id, c.callee\_id)

GROUP BY

co.name

HAVING

AVG(duration) > (SELECT AVG(duration) FROM Calls);

**Q-131**

**Ans-**

--Table: Numbers

create table if not exists Numbers

(

num int,

frequency int,

constraint pk PRIMARY KEY (num)

);

drop table Numbers;

insert into Numbers VALUES (0,7),(1,1),(2,3),(3,1);

select \* from Numbers;

--Write an SQL query to report the median of all the numbers in the database after decompressing the Numbers table. Round the median to one decimal point.

with recursive rec\_cte as

(

select num,frequency,1 asc cnt

from Numbers

UNION

select num,frequency,cnt+1 as cnt

from rec\_cte

where cnt < frequency

),

med\_cte as

(

SELECT num,frequency,cnt,

row\_number() over (order by num) row\_num,

count(\*) over () tot\_count

from rec\_cte

)

select case when MOD(tot\_count,2) = 0 then round(avg(num),1)

else round(num,1) end as median

from med\_cte

where row\_num BETWEEN

tot\_count/2 and tot\_count/2+1;

**Q-132**

**Ans-**

--Table: Salary

create table if not exists Salary

(

id int,

employee\_id int,

amount int,

pay\_date date,

constraint pk PRIMARY KEY (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');

select \* from Salary;

--Employee table:

create table if not exists Employee

(

employee\_id int,

department\_id int,

constraint pk PRIMARY KEY (employee\_id)

);

insert into Employee VALUES (1,1),(2,2),(3,2);

select \* from Employee;

--Write an SQL query to report the comparison result (higher/lower/same) of the average salary of employees in a department to the company's average salary. Return the result table in any order.

select

pay\_month,

department\_id,

case when dept\_avg > comp\_avg then 'higher' when dept\_avg < comp\_avg then 'lower' else 'same' end comparison

from (

select date\_format(b.pay\_date, '%Y-%m') pay\_month, a.department\_id, avg(b.amount) dept\_avg, d.comp\_avg

from Employee a

inner join Salary b

on (a.employee\_id = b.employee\_id)

inner join (select date\_format(c.pay\_date, '%Y-%m') pay\_month, avg(c.amount) comp\_avg

from Salary c

group by date\_format(c.pay\_date, '%Y-%m')) d

on ( date\_format(b.pay\_date, '%Y-%m') = d.pay\_month)

group by date\_format(b.pay\_date, '%Y-%m'), department\_id, d.comp\_avg) final;

**Q-133**

**Ans-**

--Table: Activity

create table if not exists Activity

(

player\_id int,

device\_id int,

event\_date date,

games\_played int,

constraint pk PRIMARY KEY (player\_id, event\_date)

);

insert INTO Activity 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 \* from Activity;

--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.Return the result table in any order.

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 a1 left join Activity a2

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

left join Activity 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;

**Q-134**

**Ans-**

--Table: Players

create table if not exists Players

(

player\_id int,

group\_id int,

constraint pk PRIMARY KEY (player\_id)

);

insert into Players VALUES (15,1), (25,1), (30,1), (45,1), (10,2), (35,2), (50,2), (20,3), (40,3);

select \* from Players;

--Table: Matches

create table if not exists Matches

(

match\_id int,

first\_player int,

second\_player int,

first\_score int,

second\_score int,

constraint pk PRIMARY KEY (match\_id)

);

insert into Matches VALUES (1,15,45,3,0),(2,30,25,1,2),(3,30,15,2,0),(4,40,20,5,2),(5,35,50,1,1);

select \* from Matches;

--Write an SQL query to find the winner in each group.

--Return the result table in any order.

select group\_id, player\_id from (

select p.group\_id, ps.player\_id, sum(ps.score) as score

from Players p,

(

select first\_player as player\_id, first\_score as score

from Matches

union all

select second\_player, second\_score

from Matches

) ps

where p.player\_id = ps.player\_id

group by ps.player\_id

order by group\_id, score desc, player\_id

-- limit 1 -- by default, groupby will pick the first one i.e. max score player here

) top\_scores

group by group\_id;

**Q-135**

**Ans-**

--Table: Student

create table if not exists Student

(

student\_id int,

student\_name VARCHAR(50),

constraint pk PRIMARY KEY (student\_id)

);

insert into Student VALUES (1,'Daniel'),(2,'Jade'),(3,'Stella'),(4,'Jonathan'),(5,'Will');

select \* from Student;

--Table: Exam

create table if not exists Exam

(

exam\_id int,

student\_id int,

score int,

constraint pk PRIMARY KEY (exam\_id, student\_id)

);

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 \* from Exam;

--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.Return the result table ordered by student\_id.

select

Student.\*

from Exam

inner join Student on Student.student\_id=Exam.student\_id

group by student\_id

having max(score) not in (select max(score) from Exam)

and min(score) not in (select min(score) from Exam);

**Q-136**

**Ans-**

--Table: Student

--Exam table:

--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.Return the result table ordered by student\_id.

select

Student.\*

from Exam

inner join Student on Student.student\_id=Exam.student\_id

group by student\_id

having max(score) not in (select max(score) from Exam)

and min(score) not in (select min(score) from Exam);

**Q-137**

**Ans-**

--Table: UserActivity

create table if not exists UserActivity

(

username VARCHAR(50),

activity varchar(50),

startDate date,

endDate date

);

insert INTO UserActivity VALUES ('Alice','Travel','2020-02-12','2020-02-20'), ('Alice','Dancing','2020-02-21','2020-02-23'), ('Alice','Travel','2020-02-24','2020-02-28'),('Bob','Travel','2020-02-11','2020-02-18');

select \* from UserActivity;

--Write an SQL query to show the second most recent activity of each user.If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.Return the result table in any order.

select distinct username, activity, startDate, endDate

from

(select u.\*,

rank() over (partition by username order by startDate desc) as rnk,

count(activity) over (partition by username) as num

from UserActivity u) t

where (num <> 1 and rnk = 2) or (num = 1 and rnk = 1);

**Q-138**

**Ans-**

--Table: UserActivity

--Write an SQL query to show the second most recent activity of each user.If the user only has one activity, return that one. A user cannot perform more than one activity at the same time. Return the result table in any order.

select distinct username, activity, startDate, endDate

from

(select u.\*,

rank() over (partition by username order by startDate desc) as rnk,

count(activity) over (partition by username) as num

from UserActivity u) t

where (num <> 1 and rnk = 2) or (num = 1 and rnk = 1);

**Q-139**

**Ans-**

--STUDENTS table

create table if not exists STUDENTS

(

id int,

name VARCHAR(50),

marks int

);

insert into STUDENTS VALUES (1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),(3,'Belvet',84);

select \* from STUDENTS;

--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.

SELECT NAME FROM STUDENTS WHERE MARKS > 75 ORDER BY RIGHT(NAME,3), ID ASC;

**Q-140**

**Ans-**

--Employee table

create table if not exists Employee

(

employee\_id int,

name VARCHAR(50),

months int,

salary int

);

insert into Employee 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 \* from Employee;

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

SELECT name FROM Employee ORDER BY name;

**Q-141**

**Ans-**

--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.

SELECT name FROM Employee WHERE salary > 2000 AND months < 10 ORDER BY employee\_id;

**Q-142**

**Ans-**

--TRIANGLES table

create table if not exists TRIANGLES

(

A int,

B int,

C int

);

insert into TRIANGLES VALUES (20,20,23),(20,20,20),(20,21,22),(13,14,30);

select \* from TRIANGLES;

--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.

SELECT CASE

WHEN A + B <= C OR A + C <= B OR B + C <= A THEN 'Not A Triangle'

WHEN A = B AND B = C THEN 'Equilateral'

WHEN A = B OR B = C OR A = C THEN 'Isosceles'

ELSE 'Scalene'

END

FROM TRIANGLES;

**Q-143**

**Ans-**

--EMPLOYEES table

create table if not exists EMPLOYEES

(

id int,

name VARCHAR(50),

salary int

);

insert into EMPLOYEES VALUES (1,'Kristeen',1420), (2,'Ashley',2006), (3,'Julia',2210), (4,'Maria',3000);

select \* from EMPLOYEES;

--Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries), and round it up to the next integer.

select ceil(avg(salary) - avg(replace(salary, '0', ''))) from EMPLOYEES;

**Q-144**

**Ans-**

--Employee table

create table if not exists Employee

(

employee\_id int,

name VARCHAR(50),

months int,

salary int

);

Insert into Employee 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 \* from Employee;

--Write a query to find the maximum total earnings for all employees as well as the total number ofemployees who have maximum total earnings. Then print these values as 2 space-separated integers.

select MAX(salary\*months), COUNT(\*) from Employee where (salary \* months) >= (select MAX(salary \* months) from Employee);

**Q-145**

**Ans-**

--OCCUPATIONS table

create table if not exists OCCUPATIONS

(

Name VARCHAR(50),

Occupation VARCHAR(50)

);

insert into OCCUPATIONS VALUES ('Samantha','Doctor'), ('Julia','Actor'), ('Maria','Actor'), ('Meera','Singer'), ('Ashely','Professor'),('Ketty','Professor'),('Christeen','Professor'),('Jane','Actor'),('Jenny','Doctor'),('Priya','Singer');

select \* from OCCUPATIONS;

--Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order,

(

SELECT CONCAT(NAME, '(', SUBSTRING(Occupation, 1, 1), ')') as THETEXT, '1' as SELECTNUMBER

FROM OCCUPATIONS

)

UNION ALL

(

SELECT CONCAT('There are total ', COUNT(\*),' ', Occupation, (IF (COUNT(\*) > 1, 's',''))) as THETEXT, '2' as SELECTNUMBER

FROM OCCUPATIONS GROUP BY Occupation

)

ORDER BY SELECTNUMBER ASC, THETEXT ASC;

**Q-146**

**Ans-**

--Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

select

Doctor,

Professor,

Singer,

Actor

from (

select

NameOrder,

max(case Occupation when 'Doctor' then Name end) as Doctor,

max(case Occupation when 'Professor' then Name end) as Professor,

max(case Occupation when 'Singer' then Name end) as Singer,

max(case Occupation when 'Actor' then Name end) as Actor

from (

select

Occupation,

Name,

row\_number() over(partition by Occupation order by Name ASC) as NameOrder

from OCCUPATIONS

) as NameLists

group by NameOrder

) as Names;

**Q-147**

**Ans-**

--BST TABLE

create table if not exists BST

(

N int,

P int

);

insert into BST VALUES(1,2),(3,2),(6,8),(9,8),(2,5),(8,5),(5,null);

select \* from BST;

--Write a query to find the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:

--● Root: If node is root node.

--● Leaf: If node is leaf node.

--● Inner: If node is neither root nor leaf node.

SELECT N,

CASE

WHEN P IS NULL THEN 'Root'

WHEN N IN (SELECT P FROM BST) THEN 'Inner'

ELSE 'Leaf'

END

FROM BST

ORDER by N;

**Q-148**

**Ans-**

--Given the table schemas below, write a query to print the company\_code, founder name, total number of lead managers, total number of senior managers, total number of managers, and total number of employees. Order your output by ascending company\_code.

--Company

create table if not exists Company

(

company\_code VARCHAR(50),

founder VARCHAR(50)

);

insert into Company VALUES ('C1','Monika'),('C2','Samantha');

select \* from Company;

--Lead\_Manager

create table if not exists Lead\_Manager

(

lead\_manager\_code VARCHAR(50),

company\_code VARCHAR(50)

);

insert into Lead\_Manager VALUES ('LM1','C1'),('LM2','C2');

select \* from Lead\_Manager;

--Senior\_Manager

create table if not exists Senior\_Manager

(

senior\_manager\_code VARCHAR(50),

lead\_manager\_code VARCHAR(50),

company\_code VARCHAR(50)

);

insert into Senior\_Manager VALUES ('SM1','LM1','C1'),('SM2','LM1','C1'),('SM3','LM2','C2');

select \* from Senior\_Manager;

--Manager Table:

create table if not exists Manager

(

manager\_code VARCHAR(50),

senior\_manager\_code VARCHAR(50),

lead\_manager\_code VARCHAR(50),

company\_code VARCHAR(50)

);

insert into Manager VALUES ('M1','SM1','LM1','C1'),('M2','SM3','LM2','C2'),('M3','SM3','LM2','C2');

select \* from Manager;

--Employee:

create table if not exists Employee

(

employee\_code VARCHAR(50),

manager\_code VARCHAR(50),

senior\_manager\_code VARCHAR(50),

lead\_manager\_code VARCHAR(50),

company\_code VARCHAR(50)

);

insert into Employee VALUES ('E1','M1','SM1','LM1','C1'),('E2','M1','SM1','LM1','C1'), ('E3','M2','SM3','LM2','C2'), ('E4','M3','SM3','LM2','C2');

select \* from Employee;

SELECT c.company\_code, c.founder, COUNT(DISTINCT e.lead\_manager\_code), COUNT(DISTINCT e.senior\_manager\_code), COUNT(DISTINCT e.manager\_code), COUNT(DISTINCT e.employee\_code) FROM Company c

JOIN Employee e ON c.company\_code = e.company\_code GROUP BY c.company\_code, c.founder ORDER BY c.company\_code;

**Q-149**

**Ans-**

--Functions table

create table if not exists Functions

(

X int,

Y int

);

insert into Functions VALUES (20,20),(20,20),(20,21),(23,22),(22,23),(21,20);

select \* from Functions;

--Write a query to output all such symmetric pairs in ascending order by the value of X. List the row such that X1 ≤ Y1.

SELECT f1.X, f1.Y FROM Functions AS f1

WHERE f1.X = f1.Y AND

(SELECT COUNT(\*) FROM Functions WHERE X = f1.X AND Y = f1.Y) > 1

UNION

SELECT f1.X, f1.Y from Functions AS f1

WHERE EXISTS(SELECT X, Y FROM Functions WHERE f1.X = Y AND f1.Y = X AND f1.X < X)

ORDER BY X;

**Q-150**

**Ans-**

--Students TABLE

create table if not exists Students

(

id int,

name VARCHAR(50)

);

insert into Students VALUES (1,'Ashley'),(2,'Samantha'),(3,'Julia'),(4,'Scarlet');

select \* from Students;

--Friends TABLE

create table if not exists Friends

(

id int,

friend\_id int

);

insert into Friends VALUES (1,2),(2,3),(3,4),(4,1);

select \* from Friends;

--Packages. TABLE

create table if not exists Packages

(

id int,

salary float

);

insert into Packages VALUES (1,15.20),(2,10.06),(3,11.55),(4,12.12);

select \* from Packages;

--Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students get the same salary offer.

select S1.name

from Students s1

inner join Packages p1 on s1.id = p1.id

inner join Friends f on s1.id = f.id

inner join Students s2 on f.friend\_id = s2.id

inner join Packages p2 on s2.id = p2.id

where p1.salary < p2.salary

order by p2.salary;

**Q-151**

**Ans-**

--Hackers table:

create table if not exists Hackers

(

hacker\_id int,

name VARCHAR(50)

);

insert into Hackers VALUES(5580,'Rose'),(8439,'Angela'),(27205,'Frank'),(52243,'Patrick'), (52348,'Lisa'),(57645,'Kimberly'),(77726,'Bonnie'),(83082,'Michael'),(86870,'Todd'),(90411,'Joe');

select \* from Hackers;

--Difficulty: table:

create table if not exists Difficulty

(

difficulty\_level int,

score int

);

insert into Difficulty VALUES (1,20),(2,30),(3,40),(4,60),(5,80),(6,100),(7,120);

select \* from Difficulty;

--Challenges table:

create table if not exists Challenges

(

challenge\_id int,

hacker\_id int,

difficulty\_level int

);

insert into Challenges VALUES (4810,77726,4), (21089,27205,1), (36566,5580,7), (66730,52243,6), (71055,52243,2);

select \* from Challenges;

--Submissions table:

create table if not exists Submissions

(

submission\_id int,

hacker\_id int,

challenge\_id int,

score int

);

insert into Submissions VALUES (68628,77726,36566,30), (65300,77726,21089,10), (40326,52243,36566,77), (8941,27205,4810,4),(83554,77726,66730,30),(97397,90411,4810,40),(84162,83082,4810,40),(97431,90411,71055,30);

select \* from Submissions;

--Write a query to print the respective hacker\_id and name of hackers who achieved full scores for more than one challenge. Order your output in descending order by the total number of challenges in which the hacker earned a full score. If more than one hacker received full scores in the same number of challenges, then sort them by ascending hacker\_id.

SELECT S.hacker\_id, name

FROM Submissions AS S

JOIN Hackers AS H ON S.hacker\_id = H.hacker\_id

JOIN Challenges AS C ON S.challenge\_id = C.challenge\_id

JOIN Difficulty AS D ON C.difficulty\_level = D.difficulty\_level

WHERE S.score = D.score

GROUP BY name, S.hacker\_id

HAVING count(S.challenge\_id) > 1

ORDER BY count(S.challenge\_id) DESC, S.hacker\_id;

**Q-152**

**Ans-**

--table Projects

create table if not exists Projects

(

Task\_ID int,

Start\_Date date,

End\_Date date

);

insert into Projects VALUES (1,'2015-10-01','2015-10-02'),(2,'2015-10-02','2015-10-03'),(3,'2015-10-03','2015-10-04'),(4,'2015-10-13','2015-10-14'),(5,'2015-10-14','2015-10-15'),(6,'2015-10-28','2015-10-29'),(7,'2015-10-30','2015-10-31');

--Write a query to output the start and end dates of projects listed by the number of days it took to complete the project in ascending order. If there is more than one project that have the same number of completion days, then order by the start date of the project.

Select Start\_Date, MIN(End\_Date)

From

(Select b.Start\_Date

From Projects as a

RIGHT Join Projects as b

ON b.Start\_Date = a.End\_Date

WHERE a.Start\_Date IS NULL

) sd,

(Select a.End\_Date

From Projects as a

Left Join Projects as b

ON b.Start\_Date = a.End\_Date

WHERE b.End\_Date IS NULL

) ed

Where Start\_Date < End\_Date

GROUP BY Start\_Date

ORDER BY datediff(MIN(End\_Date), Start\_Date), Start\_Date;

**Q-153**

**Ans-**

--transactions Table:

create table if not exists transactions

(

user\_id int,

amount float,

transaction\_date TIMESTAMP

);

insert into transactionsVALUES(1,9.99,'08/01/202210:00:00'),(1,55,'08/17/202210:00:00'), (2,149.5,'08/05/2022 10:00:00'), (2,4.89,'08/06/2022 10:00:00'), (2,34,'08/07/2022 10:00:00');

select \* from transactions;

--In an effort to identify high-value customers, Amazon asked for your help to obtain data about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more consecutive days. List the user IDs who have gone on at least 1 shopping spree in ascending order.

**Q-154**

**Ans-**

--payments Table:

--You are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A two-way unique relationship is established when two people send money back and forth. Write a query to find the number of two-way unique relationships in this data.Assumption:

--A payer can send money to the same recipient multiple times.

create table if not exists payments

(

payer\_id int,

recipient\_id int,

amount int

);

insert into payments VALUES (101,201,30), (201,101,10), (101,301,20), (301,101,80), (201,301,70);

select \* from payments;

WITH T1 AS

(SELECT

payer\_id,

recipient\_id

FROM payments

INTERSECT

SELECT

recipient\_id,

payer\_id

FROM payments)

SELECT

COUNT(payer\_id)/2 AS UNIQUE\_RELATIONSHIPS

FROM

T1;

**Q-155**

**Ans-**

--user\_logins Table:

create table if not exists user\_logins

(

user\_id int,

login\_date DATETIME

);

insert into user\_logins VALUES (725,'03/03/2022 12:00:00'), (245,'03/28/2022 12:00:00'), (112,'03/05/2022 12:00:00'), (245,'04/29/2022 12:00:00'), (112,'04/05/2022 12:00:00');

select \* from user\_logins;

--Assume you are given the table below containing information on Facebook user logins. Write a query to obtain the number of reactivated users (which are dormant users who did not log in the previous month, then logged in during the current month).Output the current month (in numerical) and number of reactivated users.

**Q-156**

**Ans-**

--user\_transactions Table:

create table if not exists user\_transactions

(

transaction\_id int,

product\_id int,

spend decimal (5,2),

transaction\_date DATETIME

);

Insertintouser\_transactionsVALUES(759274,111,49.50,'02/03/202200:00:00'),(850371,111,51.00,'03/15/202200:00:00'),(615348,145,36.30,'03/22/202200:00:00'),(137424,156,151.00,'04/04/202200:00:00'),(248475,156,87.00,'04/16/2022 00:00:00');

select \* from user\_transactions;

--Write a query to obtain the list of customers whose first transaction was valued at $50 or more. Output the number of users.

**Q-157**

**Ans-**

--measurements Table:

create table if not exists measurements

(

measurement\_id int,

measurement\_value DECIMAL,

measurement\_time datetime

);

insert into measurements VALUES (131233,1109.51,'07/10/2022 09:00:00'), (135211,1662.74,'07/10/2022 11:00:00'),(523542,1246.24,'07/10/2022.13:15:00'),(143562,1124.50,'07/11/202215:00:00'),(346462,1234.14,'07/11/2022 16:45:00');

select \* from measurements;

--Write a query to obtain the sum of the odd-numbered and even-numbered measurements on a particular day, in two different columns.

WITH ranked\_measurements AS (

SELECT

CAST(measurement\_time AS DATE) AS measurement\_day,

measurement\_value,

ROW\_NUMBER() OVER (

PARTITION BY CAST(measurement\_time AS DATE)

ORDER BY measurement\_time) AS measurement\_num

FROM measurements

)

SELECT

measurement\_day,

SUM(

CASE WHEN measurement\_num % 2 != 0 THEN measurement\_value

ELSE 0 END) AS odd\_sum,

SUM(

CASE WHEN measurement\_num % 2 = 0 THEN measurement\_value

ELSE 0 END) AS even\_sum

FROM ranked\_measurements

GROUP BY measurement\_day;

**Q-158**

**Ans-**

--transactions Table:

--transactions Table:

create table if not exists transactions

(

user\_id int,

amount float,

transaction\_date TIMESTAMP

);

insert into transactionsVALUES(1,9.99,'08/01/202210:00:00'),(1,55,'08/17/2022 10:00:00'),(2,149.5,'08/05/2022 10:00:00'),(2,4.89,'08/06/2022 10:00:00'),(2,34,'08/07/2022 10:00:00');

select \* from transactions;

--In an effort to identify high-value customers, Amazon asked for your help to obtain data about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more consecutive days. List the user IDs who have gone on at least 1 shopping spree in ascending order.

**Q-159**

**Ans-**

--rental\_amenities Table:

create table if not exists rental\_amenities

(

rental\_id int,

amenity VARCHAR(50)

);

insert into rental\_amenities VALUES (123,'pool'), (123,'kitchen'), (234,'hot tub'), (234,'fireplace'), (345,'kitchen'), (345,'pool'), (456,'pool');

select \* from rental\_amenities;

--write a query to find the unique combination of two Airbnb rentals with the same exact amenities offered.

**Q-160**

**Ans-**

--ad\_campaigns Table:

create table if not exists ad\_campaigns

(

campaign\_id int,

spend int,

revenue FLOAT,

advertiser\_id int

);

insertintoad\_campaignsVALUES(1,500,7500,3),(2,1000,900,1),(3,3000,12000,2),(4,500,2000,4),(5,100,400,4);

select \* from ad\_campaigns;

--Write a query to calculate the return on ad spend (ROAS) for each advertiser across all ad campaigns. Round your answer to 2 decimal places, and order your output by the advertiser\_id.

select advertiser\_id, round(cast(sum(revenue)/sum(spend) as numeric),2) as ROAS

from ad\_campaigns

group by advertiser\_id

order by advertiser\_id;

**Q-161**

**Ans-**

--employee\_pay Table:

create table if not exists employee\_pay

(

employee\_id int,

salary int,

title VARCHAR(50)

);

insert into employee\_pay VALUES (101,80000,'Data Analyst'), (102,90000,'Data Analyst'),(103,100000,'Data Analyst'),(104,30000,'DataAnalyst'),(105,120000,'DataScientist'),(106,100000,'DataScientist'),(107,80000,'Data Scientist'),(108,310000,'Data Scientist');

select \* from employee\_pay;

--Write a query that shows the following data for each compensation outlier: employee ID, salary, and whether they are potentially overpaid or potentially underpaid (refer to Example Output below).

**Q-162**

**Ans-**

--payments table

create table if not exists payments

(

payer\_id int,

recipient\_id int,

amount int

);

insert into payments VALUES (101,201,30), (201,101,10), (101,301,20), (301,101,80), (201,301,70);

select \* from payments;

WITH T1 AS

(SELECT

payer\_id,

recipient\_id

FROM payments

INTERSECT

SELECT

recipient\_id,

payer\_id

FROM payments)

SELECT

COUNT(payer\_id)/2 AS UNIQUE\_RELATIONSHIPS

FROM

T1;

**Q-163**

**Ans-**

--purchases Table:

create table if not exists purchases

(

user\_id int,

product\_id int,

quantity int,

purchase\_date DATETIME

);

insert into purchases VALUES (536,3223,6,'01/11/2022 12:33:44'),(827,3585,35,'02/20/2022 14:05:26'), (536,3223,5,'03/02/2022 09:33:28'), (536,1435,10,'03/02/2022 08:40:00'), (827,2452,45,'04/09/2022 00:00:00');

select \* from purchases;

--Write a query to obtain the number of users who purchased the same product on two or more different days. Output the number of unique users.

**Q-164**

**Ans-**

--search\_category Table:

create table if not exists search\_category

(

country VARCHAR(50),

search\_cat VARCHAR(50),

num\_search int,

invalid\_result\_pct DECIMAL

);

insert into search\_category VALUES ('UK','home',null,null),('UK','tax',98000,1.00),('UK','travel',100000,3.25);

select \* from search\_category;

--Write a query to obtain the percentage of invalid searches.Output the country in ascending order, total searches and overall percentage of invalid searches rounded to 2 decimal places.

**Q-165**

**Ans**

--transactions Table:

create table if not exists transactions

(

transaction\_id VARCHAR(50),

type enum('deposit','withdrawal'),

amount DECIMAL,

transaction\_date DATETIME

);

insertintotransactionsVALUES(19153,'deposit',65.90,'07/10/202210:00:00'),(53151,'deposit',178.55,'07/08/2022 10:00:00'), (29776,'withdrawal',25.90,'07/08/202210:00:00'), (16461,'withdrawal',45.99,'07/08/2022 10:00:00'), (77134,'deposit',32.60,'07/10/2022 10:00:00');

select \* from transactions;

--Write a query to print the cumulative balance of the merchant account at the end of each day, with the total balance reset back to zero at the end of the month. Output the transaction date and cumulative balance.

**Q-166**

**Ans-**

--product\_spend Table:

create table if not exists product\_spend

(

category VARCHAR(50),

product VARCHAR(50),

user\_id int,

spend int,

transaction\_date TIMESTAMP

);

insert into product\_spend VALUES ('appliance','refrigerator',165,246.00,'12/26/2021 12:00:00'), ('appliance','refrigerator',123,299.99,'03/02/202212:00:00'),('appliance','washingmachine',123,219.80,'03/02/202212:00:00'),('electronics','vacuum',178,152.00,'04/05/202212:00:00'),('electronics','wirelessheadset',156,249.90,'07/08/2022 12:00:00'), ('electronics','vacuum',145,189.00,'07/15/2022 12:00:00');

select \* from product\_spend;

--Identify the top two highest-grossing products within each category in 2022. Output the category, product, and total spend.

SELECT

category,

product,

total\_spend

FROM (

SELECT

\*,

RANK() OVER (

PARTITION BY category

ORDER BY total\_spend DESC) AS ranking

FROM (

SELECT

category,

product,

SUM(spend) AS total\_spend

FROM product\_spend

WHERE transaction\_date >= '2022-01-01'

AND transaction\_date <= '2022-12-31'

GROUP BY category, product) AS total\_spend

) AS top\_spend

WHERE ranking <= 2

ORDER BY category, ranking;

**Q-167**

**Ans-**

--users Table:

create table if not exists users

(

user\_id int,

signup\_date DATETIME,

last\_login DATETIME

);

insert into users VALUES (1001,'06/01/2022 12:00:00','07/05/2022 12:00:00'),(1002,'06/03/2022 12:00:00','06/15/2022 12:00:00'),(1004,'06/02/2022 12:00:00','06/15/2022 12:00:00'),(1006,'06/15/2022 12:00:00','07/05/2022 12:00:00'),(1012,'06/16/2022 12:00:00','07/22/2022 12:00:00');

select \* from users;

--Write a query to generate the churn rate by week in June 2022. Output the week number (1, 2, 3, 4, ...) and the corresponding churn rate rounded to 2 decimal places.

**Q-168**

**Ans-**

--songs\_history Table:

create table if not exists songs\_history

(

history\_id int,

user\_id int,

song\_id int,

song\_plays int

);

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

select \* from songs\_history;

--songs\_weekly Table:

create table if not exists songs\_weekly

(

user\_id int,

song\_id int,

listen\_time DATETIME

);

insert into songs\_weekly VALUES (777,1238,'08/01/2022 12:00:00'), (695,4520,'08/04/2022 08:00:00'), (125,9630,'08/04/2022 16:00:00'), (695,9852,'08/07/2022 12:00:00');

select \* from songs\_weekly;

--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.

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

FROM (

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

) AS report

GROUP BY user\_id, song\_id

ORDER BY song\_count DESC;

**Q-169**

**Ans-**

--emails Table:

create table if not exists emails

(

email\_id int,

user\_id int,

signup\_date DATETIME

);

insert into emails VALUES (125,7771,'06/14/2022 00:00:00'), (236,6950,'07/01/2022 00:00:00'), (433,1052,'07/09/2022 00:00:00');

select \* from emails;

--texts Table:

create table if not exists texts

(

text\_id int,

email\_id int,

signup\_action VARCHAR(50)

);

insert into texts VALUES (6878,125,'Confirmed'),(6920,236,'Not Confirmed'),(6994,236,'Confirmed');

select \* from texts;

--Write a query to find the confirmation rate of users who confirmed their signups with text messages. Round the result to 2 decimal places.

SELECT

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

FROM (

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'

) AS rate;

**Q-170**

**Ans-**

--tweets Table:

create table if not exists tweets

(

tweet\_id int,

user\_id int,

tweet\_date timestamp

);

InsertintotweetsVALUES(214252,111,'06/01/202212:00:00'),(739252,111,'06/01/202212:00:00'),(846402,111,'06/02/2022 12:00:00'), (241425,254,'06/02/2022 12:00:00'), (137374,111,'06/04/2022 12:00:00');

select \* from tweets;

--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.

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;

**Q-171**

**Ans-**

--activities Table:

create table if not exists activities

(

activity\_id int,

user\_id int,

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

time\_spent FLOAT,

activity\_date DATETIME

);

insert into activities VALUES (7274,123,'open',4.50,'06/22/2022 12:00:00'), (2425,123,'send',3.50,'06/22/2022 12:00:00'),(1413,456,'send',5.67,'06/23/202212:00:00'),(1414,789,'chat',11.00,'06/25/202212:00:00'),(2536,456,'open',3.00,'06/25/2022 12:00:00');

select \* from activities;

--age\_breakdown Table:

create table if not exists age\_breakdown

(

user\_id int,

age\_bucket varchar(50)

);

insert into age\_breakdown VALUES (123,'31-35'),(456,'26-30'),(789,'21-25');

select \* from age\_breakdown;

--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.

**Q-172**

**Ans-**

--personal\_profiles Table:

create table if not exists personal\_profiles

(

profile\_id int,

name varchar(50),

followers int

);

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);

select \* from personal\_profiles;

--employee\_company Table:

create table if not exists employee\_company

(

personal\_profile\_id int,

company\_id int

);

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

select \* from employee\_company;

--company\_pages Table:

create table if not exists company\_pages

(

company\_id int,

name VARCHAR(50),

followers int

);

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

select \* from company\_pages;

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