```
create database gds_mysql_assignment_db3;

use gds_mysql_assignment_db3;

--Q 101.
--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','2020-0
2-21','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);

--Q102.
--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);
```

```
--Q103.
--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;


--Q104.
--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',16,43
72),(78454,'Bonnie',8,1771),(83565,'Michael',6,2017),(98607,'Todd',5,33
96),(99989,'Joe',9,3573);

select * from Employee;

--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;
```

```
--Q105
--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;



--Q106.
--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;


--Q107.
--Employee table
 select MAX(salary*months), COUNT(*) from Employee where (salary *
months) >= (select MAX(salary * months) from Employee);



--Q108.
--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','Sin
ger'),('Ashely','Professor'),('Ketty','Professor'),('Christeen','Profes
sor'),('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;


--Q109.
--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;


--Q110.
--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;


--Q111 .


--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','S
M3','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;


--Q112.
--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);
```

```
--Q113.
--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;
```

```
--Q114.
--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;
```

```
--Q115.
--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;


--Q116.
--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',16,43
72),(78454,'Bonnie',8,1771),(83565,'Michael',6,2017),(98607,'Todd',5,33
96),(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;


--Q117..
--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;

--Q118.
----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;



--Q119.
--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,'12/31/2020
12:00:00')(1623,123424,1246.44,'12/31/2021
12:00:00')(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.
```

```
--Q120.
--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_prim
e','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;



--Q121.
--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','06/05/2022
12:00:00'),(742,1374,'comment','06/05/2022
12:00:00'),(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);



--Q122.
--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;
```

```
--Q123.
--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;


--Q124.

--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.



--Q125.
--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,'09/25/2022
12:00:00'),(2,101,1,100,'09/25/2022'),(3,101,1,100,'09/25/2022
```

```
12:28:00'),(4,102,2,300,'09/25/2022 12:00:00'),(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;


--Q126.
--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','06/05/2022 14:40:00'),(141367,1314,100362,'completed
incorrectly','06/07/2022
15:03:00'),(582193,5421,100657,'never_received','07/07/2022
15:22:00'),(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','06/05/2022
15:46:00'),(101,100362,'06/07/2022 15:33:00','06/07/2022
16:45:00'),(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.



--Q127.
-- 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;


--Q128.
--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','21
2-1234567'),(2,'Maroua','212-6523651'),(7,'Meir','972-1234567'),(9,'Rac
hel','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',2
51);

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);



--Q129.
--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;


--Q130.
--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;


--Q131.
--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,'20
16-03-01',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;
```

```
--Q132.
--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;
```

```
--Q133.
--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);


--Q134.
--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);


--Q135.
--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','2020-0
2-21','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);


--Q136.
--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);


--Q137.
--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;


--Q138.
--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',16,43
72),(78454,'Bonnie',8,1771),(83565,'Michael',6,2017),(98607,'Todd',5,33
96),(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);
```

```
--Q139.
--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','Sin
ger'),('Ashely','Professor'),('Ketty','Professor'),('Christeen','Profes
sor'),('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;


--Q140 .
--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;



--Q141.
--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;



--Q142 .

--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','S
M3','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;



--Q143.
--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;



--Q144.
```

```
--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;


--Q145.
```

```
--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,'T
odd'),(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),(71055,52
243,2);

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),(8
941,27205,4810,4),(83554,77726,66730,30),(97397,90411,4810,40),(84162,8
3082,4810,40),(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;



--Q146.
--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;
```

```
--Q147.
--transactions Table:
create table if not exists transactions
(
    user_id int,
    amount float,
    transaction_date TIMESTAMP
);

insert into transactions VALUES (1,9.99,'08/01/2022
10:00:00'),(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.

```
--Q148 .
--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;



--Q149.
--user_transactions Table:
create table if not exists user_transactions
(
    transaction_id int,
    product_id int,
    spend decimal (5,2),
    transaction_date DATETIME
);

insert into user_transactions VALUES (759274,111,49.50,'02/03/2022
00:00:00'),(850371,111,51.00,'03/15/2022
00:00:00'),(615348,145,36.30,'03/22/2022
00:00:00'),(137424,156,151.00,'04/04/2022
00:00:00'),(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.



--Q150.
--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,'07/11/2022 15:00:00'),
(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;



--Q151.
--transactions Table:
--transactions Table:
```

```
create table if not exists transactions
(
    user_id int,
    amount float,
    transaction_date TIMESTAMP
);

insert into transactions VALUES (1,9.99,'08/01/2022
10:00:00'),(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.


--Q152.
--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.



--Q153.
--ad_campaigns Table:
create table if not exists ad_campaigns
(
    campaign_id int,
    spend int,
    revenue FLOAT,
    advertiser_id int
);

insert into ad_campaigns VALUES (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;


--Q154.
--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,'Data
Analyst'),(105,120000,'Data Scientist'),(106,100000,'Data
Scientist'),(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).




--Q155.
--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;


--Q156.
--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.



--Q157.
--transactions Table:
create table if not exists transactions
(
    transaction_id VARCHAR(50),
    type enum('deposit','withdrawal'),
    amount DECIMAL,
    transaction_date DATETIME
);
```

```
insert into transactions VALUES (19153,'deposit',65.90,'07/10/2022
10:00:00'),(53151,'deposit',178.55,'07/08/2022
10:00:00'),(29776,'withdrawal',25.90,'07/08/2022
10:00:00'),(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.




--Q158.
--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,'03/02/2022
12:00:00'),('appliance','washing machine',123,219.80,'03/02/2022
12:00:00'),('electronics','vacuum',178,152.00,'04/05/2022
12:00:00'),('electronics','wireless headset',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;


--Q159.
--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.
```