```
create database gds_mysql_assignment_db2;

use gds_mysql_assignment_db2;

--Q51.
--World table:
create table if not exists World
(
    name VARCHAR(50),
    continent varchar(50),
    area int,
    population bigint,
    gdp bigint,
    constraint pk PRIMARY KEY (name)
);

insert into World VALUES
('Afghanistan','Asia',652230,25500100,20343000000),('Albania','Europe',
28748,2831741,12960000000),('Algeria','Africa',2381741,37100000,1886810
00000),('Andorra','Europe',468,78115,3712000000),('Angola','Africa',124
6700,20609294,100990000000);

select * from World;

--Write an SQL query to report the name, population, and area of the
big countries.
--Return the result table in any order.
SELECT name,population,area
FROM World
WHERE area > 3000000 or population > 25000000;


--Q52.
--Table: Customer
create table if not exists Customer
(
    id int,
    name varchar(50),
    referee_id int,
    constraint pk PRIMARY KEY (id)
);

insert into Customer VALUES
(1,'Will',null),(2,'Jane',null),(3,'Alex',2),(4,'Bill',null),(5,'Zack',
1),(6,'Mark',2);
select * from Customer;

--Write an SQL query to report the names of the customer that are not
referred by the customer with id= 2.
--Return the result table in any order.
select name from Customer where referee_id != 2 or referee_id is null;
```

```
--Q53.
--Table: Customers
create table if not exists Customers
(
    id int,
    name varchar(50),
    constraint pk PRIMARY KEY (id)
);

insert into Customers VALUES (1,'Joe'),(2,'Henry'),(3,'Sam'),(4,'Max');

select * from Customers;

--Table: Orders
create table if not exists Orders
(
    id int,
    customerId int,
    constraint pk PRIMARY KEY (id),
    constraint fk FOREIGN KEY (customerId) REFERENCES Customers(id)
);

insert into Orders VALUES (1,3),(2,1);

select * from Orders;

--Write an SQL query to report all customers who never order anything.
--Return the result table in any order.
SELECT C.Name FROM Customers C LEFT JOIN Orders O ON C.Id =
O.CustomerId WHERE O.CustomerId is NULL;


--Q54.
--Table: Employee
create table if not exists Employee
(
    employee_id int,
    team_id int,
    constraint pk PRIMARY KEY (employee_id)
);

insert into Employee VALUES (1,8),(2,8),(3,8),(4,7),(5,9),(6,9);

select * from Employee;

--Write an SQL query to find the team size of each of the employees.
--Return result table in any order.
SELECT employee_id, COUNT(team_id) OVER (PARTITION BY team_id)
team_size
```

```
FROM Employee;


--Q55
--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);



--Q56.
--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,'2016-05-02',6),(2,3,'2017-06-25',1),(3,1,'20
16-03-02',0),(3,4,'2018-07-03',5);

select * from Activity;

--Write an SQL query to report the device that is first logged in for
each player.
--Return the result table in any order
select player_id, min(event_date) as first_login
from Activity
group by player_id;


--Q57.
--Table: Orders
create table if not exists Orders
(
    order_number int,
    customer_number int,
    constraint pk PRIMARY KEY (order_number)
);
```

```
insert into Orders VALUES (1,1),(2,2),(3,3),(4,3);

select * from Orders;

--Write an SQL query to find the customer_number for the customer who
has placed the largest number of orders.
--The test cases are generated so that exactly one customer will have
placed more orders than any other customer.
SELECT
    customer_number
FROM
    Orders
GROUP BY customer_number
ORDER BY COUNT(*) DESC
LIMIT 1
;


--Q58.
--Table: Cinema
create table if not exists Cinema
(
    seat_id int AUTO_INCREMENT,
    free bool,
    constraint pk PRIMARY KEY (seat_id)
);

insert into Cinema VALUES (1,1),(2,0),(3,1),(4,1),(5,1);

select * from Cinema;

--Write an SQL query to report all the consecutive available seats in
the cinema.
--Return the result table ordered by seat_id in ascending order.
SELECT
    DISTINCT t1.seat_id
FROM Cinema AS t1 JOIN Cinema AS t2
ON abs(t1.seat_id - t2.seat_id) = 1
AND t1.free = 1 AND t2.free = 1
ORDER BY 1;


--Q59.
--Table: SalesPerson
create table if not exists SalesPerson
(
    sales_id int,
    name VARCHAR(50),
    salary int,
    commission_rate int,
```

```
    hire_date date,
    constraint pk PRIMARY KEY (sales_id)
);

INSERT into SalesPerson VALUES
(1,'John',100000,6,'4/1/2006'),(2,'Amy',12000,5,'5/1/2010'),(3,'Mark',6
5000,12,'12/25/2008'),(4,'Pam',25000,25,'1/1/2005'),(5,'Alex',5000,10,'
2/3/2007');


select * from SalesPerson;


--Table: Company
create table if not exists Company
(
    com_id int,
    name VARCHAR(50),
    city VARCHAR(50),
    constraint pk PRIMARY KEY (com_id)
);

insert into Company VALUES (1,'RED','Boston'),(2,'ORANGE','New
York'),(3,'YELLOW','Boston'),(4,'GREEN','Austin');

select * from Company;

--Table: Orders
create table if not exists Orders
(
    order_id int,
    order_date DATE,
    com_id int,
    sales_id int,
    amount int,
    constraint pk PRIMARY KEY (order_id),
    constraint fk FOREIGN KEY (com_id) REFERENCES Company(com_id),
    constraint fk FOREIGN KEY (sales_id) REFERENCES
SalesPerson(sales_id)
);


insert into Orders VALUES
(1,'1/1/2014',3,4,10000),(2,'2/1/2014',4,5,5000),(3,'3/1/2014',1,1,5000
0),(4,'4/1/2014',1,4,25000);

select * from Orders;


--Write an SQL query to report the names of all the salespersons who
did not have any orders related to the company with the name "RED".
```

```
--Return the result table in any order.
SELECT name
FROM Salesperson
WHERE sales_id
NOT IN (
    SELECT s.sales_id FROM Orders o
    INNER JOIN Salesperson s ON o.sales_id = s.sales_id
    INNER JOIN Company c ON o.com_id = c.com_id
    WHERE c.name = 'RED'
);


--Q60.
--Table: Triangle
create table if not exists Triangle
(
    x int,
    y int,
    z int,
    constraint pk PRIMARY KEY (x,y,z)
);

insert into Triangle VALUES (13,15,30),(10,20,15);

select * from Triangle;

--Write an SQL query to report for every three line segments whether
they can form a triangle.
--Return the result table in any order.

SELECT
    x,
    y,
    z,
    CASE WHEN x + y > z AND y + z > x AND z + x > y THEN 'Yes'
        ELSE 'No' END AS triangle
FROM Triangle;


--Q61.
--Table: Point
create table if not exists Point
(
    x int,
    constraint pk PRIMARY KEY (x)
);

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

select * from Point;
```

```
--Write an SQL query to report the shortest distance between any two
points from the Point table.
--The query result format is in the following example.
select min(abs(p2.x-p1.x)) as shortest
from Point p1, Point p2
where p1.x != p2.x;


--Q62.
--Table: ActorDirector
create table if not exists ActorDirector
(
    actor_id int,
    director_id int,
    timestamp int,
    constraint pk PRIMARY KEY (timestamp)
);


insert into ActorDirector VALUES
(1,1,0),(1,1,1),(1,1,2),(1,2,3),(1,2,4),(2,1,5),(2,1,6);

select * from ActorDirector;

--Write a SQL query for a report that provides the pairs (actor_id,
director_id) where the actor has cooperated with the director at least
three times.
--Return the result table in any order.
SELECT actor_id, director_id
FROM ActorDirector
GROUP BY actor_id, director_id
HAVING COUNT(*) >= 3;


--Q63.

--Table: Product
create table if not exists Product
(
    product_id int,
    product_name varchar(50),
    constraint pk PRIMARY KEY (product_id)
);

insert into Product VALUES (100,'Nokia'),(200,'Apple'),(300,'Samsung');

select * from Product;

--Table: Sales
create table if not exists Sales
(
```

```
    sale_id int,
    product_id int,
    year int,
    quantity int,
    price int,
    constraint pk PRIMARY KEY (sale_id, year),
    constraint fk FOREIGN KEY (product_id) REFERENCES
Product(product_id)
);

insert into Sales VALUES
(1,100,2008,10,5000),(2,100,2009,12,5000),(7,200,2011,15,9000);

select * from Sales;

--Write an SQL query that reports the product_name, year, and price for
each sale_id in the Sales table.
--Return the resulting table in any order.
select p.product_name, s.year, s.price
from Product p
join Sales s
on s.product_id = p.product_id;

--Q64.
--Table: Employee
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',1),(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 average experience years of all
the employees for each project, rounded to 2 digits.
--Return the result table in any order.
select project_id , round(avg(experience_years), 2) as average_years
from Project as p
left join Employee as e
on p.employee_id = e.employee_id
group by project_id;



--Q65.
--Table: Product
create table if not exists Product
(
    product_id int,
    product_name VARCHAR(50),
    unit_price int,
    constraint pk PRIMARY KEY (product_id)
);

insert into Product VALUES
(1,'S8',1000),(2,'G4',800),(3,'iPhone',1400);

--Table: Sales
create table if not exists Sales
(
    seller_id int,
    product_id int,
    buyer_id int,
    sale_date date,
    quantity int,
    price int,
    constraint fk FOREIGN KEY (product_id) REFERENCES
Product(product_id)
);

insert into Sales VALUES
(1,1,1,'2019-01-21',2,2000),(1,2,2,'2019-02-17',1,800),(2,2,3,'2019-06-
02',1,800),(3,3,4,'2019-05-13',2,2800);

select * from Sales;

--Write an SQL query that reports the best seller by total sales price,
If there is a tie, report them all.
--Return the result table in any order.
select a.seller_id
from
(select seller_id, sum(price) as sum
```

```
from Sales
group by seller_id) a
where a.sum = (select max(b.sum)from(select seller_id, sum(price) as
sum
from Sales
group by seller_id)b );


--Q66.
--Table: Product

--Table: Sales

--Write an SQL query that reports the buyers who have bought S8 but not
iPhone. Note that S8 and iPhone are products present in the Product
table.
--Return the result table in any order.
select distinct buyer_id from Sales s
join Product p
on p.product_id = s.product_id
where p.product_name = 'S8'
and buyer_id not in
(
select buyer_id from Sales s
    join Product p on p.product_id = s.product_id
    where p.product_name = 'iPhone'
);


--Q67.
--Table: Customer
create table if not exists Customer
(
    customer_id int,
    name VARCHAR(50),
    visited_on date,
    amount int,
    constraint pk PRIMARY KEY (customer_id, visited_on)
);

INSERT into Customer VALUES
(1,'Jhon','2019-01-01',100),(2,'Daniel','2019-01-02',110),(3,'Jade','20
19-01-03',120),(4,'Khaled','2019-01-04',130),(5,'Winston','2019-01-05',
110),(6,'Elvis','2019-01-06',140),(7,'Anna','2019-01-07',150),(8,'Maria
','2019-01-08',80),(9,'Jaze','2019-01-09',110),(1,'Jhon','2019-01-10',1
30),(3,'Jade','2019-01-10',150);

select * from Customer;

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

```
--window (i.e., current day + 6 days before). average_amount should be
rounded to two decimal places.
--Return result table ordered by visited_on in ascending order.

select c1.visited_on, sum(c2.amount) as amount,
    round(avg(c2.amount), 2) as average_amount
from (select visited_on, sum(amount) as amount
      from Customer group by visited_on) c1
join (select visited_on, sum(amount) as amount
      from Customer group by visited_on) c2
on datediff(c1.visited_on, c2.visited_on) between 0 and 6
group by c1.visited_on
having count(c2.amount) = 7;


--Q68.
--Table: 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);

select * from Scores;


--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.
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;


--Q69.
--Table: Logs
create table if not exists Logs
(
    log_id int,
    constraint pk PRIMARY KEY (log_id)
```

```
);

insert into Logs VALUES (1),(2),(3),(7),(8),(10);

select * from Logs;

--Write an SQL query to find the start and end number of continuous
ranges in the table Logs.
--Return the result table ordered by start_id
select min(log_id) as start_id, max(log_id) as end_id
from (select l.log_id, (l.log_id - l.row_num) as diff
      from (select log_id, row_number() over() as row_num from Logs) l
      ) l2
group by diff;


--Q70.
--Table: Students
create table if not exists Students
(
    student_id int,
    student_name VARCHAR(50),
    constraint pk PRIMARY KEY (student_id)
);

insert into Students VALUES
(1,'Alice'),(2,'Bob'),(13,'John'),(6,'Alex');

select * from Students;

--Table: Subjects
create table if not exists Subjects
(
    subject_name VARCHAR(50),
    constraint pk PRIMARY KEY (subject_name)
);

insert into Subjects VALUES ('Math'),('Physics'),('Programming');

select * from Subjects;

--Table: Examinations
create table if not exists Examinations
(
    student_id int,
    subject_name VARCHAR(50)
);

INSERT into Examinations VALUES
(1,'Math'),(1,'Physics'),(1,'Programming'),(2,'Programming'),(1,'Physic
```

```
s'),(1,'Math'),(13,'Math'),(13,'Programming'),(13,'Physics'),(2,'Math')
,(1,'Math');

select * from Examinations;

--Write an SQL query to find the number of times each student attended
each exam.
--Return the result table ordered by student_id and subject_name.
select a.student_id, a.student_name, b.subject_name,
count(c.subject_name) as attended_exams
from Students as a
join Subjects as b
left join Examinations as c
on a.student_id = c.student_id and b.subject_name = c.subject_name
group by a.student_id, b.subject_name;


--Q71.
--Table: Employees
create table if not exists Employees
(
    employee_id int,
    employee_name VARCHAR(50),
    manager_id int,
    constraint pk PRIMARY KEY (employee_id)
);

insert into Employees VALUES
(1,'Boss',1),(3,'Alice',3),(2,'Bob',1),(4,'Daniel',2),(7,'Luis',4),(8,'
Jhon',3),(9,'Angela',8),(77,'Robert',1);

select * from Employees;

--Write an SQL query to find employee_id of all employees that directly
or indirectly report their work to the head of the company.
--The indirect relation between managers will not exceed three managers
as the company is small.
--Return the result table in any order.
select e3.employee_id from Employees e1, Employees e2, Employees e3
where e1.manager_id = 1 and e2.manager_id = e1.employee_id and
e3.manager_id = e2.employee_id and e3.employee_id != 1;


--Q72.
--Table: Transactions
create table if not exists Transactions
(
    id int,
    country VARCHAR(50),
    state enum('approved', 'declined'),
    amount int,
```

```
    trans_date date,
    constraint pk PRIMARY KEY (id)
);

insert into Transactions VALUES
(121,'US','approved',1000,'2018-12-18'),(122,'US','declined',2000,'2018
-12-19'),(123,'US','approved',2000,'2019-01-01'),(124,'DE','approved',2
000,'2019-01-07');


select * from Transactions;

--Write an SQL query to find for each month and country, the number of
transactions and their total amount, the number of approved
transactions and their total amount.
--Return the result table in any order.
SELECT DATE_FORMAT(trans_date, '%Y-%m') AS month, country
      , COUNT(1) AS trans_count
      , COUNT(if(state = 'approved', 1, NULL)) AS approved_count
      , SUM(amount) AS trans_total_amount
      , SUM(if(state = 'approved', amount, 0)) AS
approved_total_amount
FROM Transactions
GROUP BY month, country;



--Q73.
--Table: Actions
create table if not exists Actions
(
    user_id int,
    post_id int,
    action_date date,
    action enum('view', 'like', 'reaction', 'comment', 'report',
'share'),
    extra VARCHAR(50)
);

insert into Actions VALUES
(1,1,'2019-07-01','view',null),(1,1,'2019-07-01','like',null),(1,1,'201
9-07-01','share',null),(2,2,'2019-07-04','view',null),(2,2,'2019-07-04'
,'report','spam'),(3,4,'2019-07-04','view',null),(3,4,'2019-07-04','rep
ort','spam'),(4,3,'2019-07-02','view',null),(4,3,'2019-07-02','report',
'spam');

select * from Actions;

--Table: Removals
create table if not exists Removals
(
```

```
    post_id int,
    remove_date date,
    constraint pk PRIMARY KEY (post_id)
);

insert into Removals VALUES (2,'2019-07-20'),(3,'2019-07-18');

select * from Removals;

--Write an SQL query to find the average daily percentage of posts that
got removed after being reported as spam, rounded to 2 decimal places.
SELECT ROUND(AVG(percentage),2) AS average_daily_percent
FROM (
SELECT action_date,
(COUNT(DISTINCT b.post_id)/COUNT(DISTINCT a.post_id))*100 AS percentage
FROM Actions AS a
LEFT JOIN Removals AS b
ON a.post_id = b.post_id
WHERE a.action = 'report'
AND a.extra = 'spam'
GROUP BY a.action_date
) AS tmp;


--Q74.
--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
16-03-02',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;


--Q75.
--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
16-03-02',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;


--Q76.
--Table Salaries:
create table if not exists Salaries
(
    company_id int,
    employee_id int,
    employee_name VARCHAR(50),
    salary int,
    constraint pk PRIMARY KEY (company_id, employee_id)
);

insert into Salaries VALUES
(1,1,'Tony',2000),(1,2,'Pronub',21300),(1,3,'Tyrrox',10800),(2,1,'Pam',
300),(2,7,'Bassem',450),(2,9,'Hermione',700),(3,7,'Bocaben',100),(3,2,'
Ognjen',2200),(3,13,'Nyan Cat',3300),(3,15,'Morning Cat',7777);

--Write an SQL query to find the salaries of the employees after
applying taxes. Round the salary to the nearest integer.
--The tax rate is calculated for each company based on the following
criteria:
--● 0% If the max salary of any employee in the company is less than
$1000.
--● 24% If the max salary of any employee in the company is in the
range [1000, 10000] inclusive.
--● 49% If the max salary of any employee in the company is greater
than $10000.
--Return the result table in any order.
SELECT
    t1.company_id,
    t1.employee_id,
    t1.employee_name,
    ROUND(CASE WHEN t2.max_sal >= 1000 AND t2.max_sal <= 10000 then
salary * 0.76
        WHEN t2.max_sal > 10000 THEN salary * 0.51
        Else salary end, 0) as salary
FROM Salaries as t1 JOIN (SELECT company_id, MAX(salary) as max_sal
FROM Salaries GROUP BY 1) as t2
ON t1.company_id = t2.company_id;



--Q77.
--Table Variables:
create table if not exists Variables
(
    name varchar(50),
    value int,
    constraint pk PRIMARY KEY (name)
```

```
);

insert into Variables VALUES ('x',66),('y',77);

select * from Variables;

--Table Expressions:
create table if not exists Expressions
(
    left_operand varchar(50),
    operator enum ('<', '>', '='),
    right_operand VARCHAR(50),
    constraint pk PRIMARY KEY (left_operand, operator, right_operand)
);


insert into Expressions VALUES ('x','>','y'),('x','<','y')
,('x','=','y'),('y','>','x'),('y','<','x'),('x','=','x');

select * from Expressions;

--Write an SQL query to evaluate the boolean expressions in Expressions
table.
--Return the result table in any order.

select e.left_operand, e.operator, e.right_operand,
    case
        when e.operator = '<' then if(l.value < r.value,'true','false')
        when e.operator = '>' then if(l.value > r.value,'true','false')
        else if(l.value = r.value,'true','false')
    end as value
from Expressions e
left join Variables l on e.left_operand = l.name
left join Variables r on e.right_operand = r.name;



--Q78.
--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;

--Table Country:
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);
```

```
--Q79.
--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;




--Q80.
--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 (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;

--Assume you are given the table below containing information on user
transactions for particular products. Write a query to obtain the
year-on-year growth rate for the total spend of each product for each
year.
```

```
--Q81.
--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;


--Q82.
--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);


--Q83.
--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;


--Q84.
--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;

--Q85.
--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.



--Q86.
--transactions Table:
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;


--Q87.
--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.

```
--Q88
--Table: 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;



--Q89.
--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);


 --Q90.
--Table: Numbers
create table if not exists Numbers
(
    num int,
    frequency int,
    constraint pk PRIMARY KEY (num)
);

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.
SET @rowindex := -1;

SELECT
   AVG(d.frequency) as Median
FROM
   (SELECT @rowindex:=@rowindex + 1 AS rowindex,
           Numbers.frequency AS frequency
    FROM Numbers
    ORDER BY Numbers.frequency) AS d
WHERE
d.rowindex IN (FLOOR(@rowindex / 2), CEIL(@rowindex / 2));


--Q91.
--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);

--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;


--Q92.
--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;


--Q93.
--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 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;



--Q94.
--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);


--Q95.
--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);



--Q96.
--Table: song_history
--Table: songs_weekly
create table song_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 datetime);
```

```
-- Q96.You're given two tables on Spotify users' streaming data.
songs_history table contains the historical
-- streaming data and songs_weekly table contains the current week's
streaming data.
-- 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_plays from(
select user_id,song_id,song_plays from
song_history union all
select user_id,song_id,count(song_id) as song_plays
from songs_weekly
where listen_time<='2022-08-04 23:59:59'
GROUP BY user_id,song_id)report
group by user_id,song_id
order by song_plays desc;



----Q97.
--Table: emails
--Table: texts
create table emails(
email_id int,
user_id int,
signup_date datetime);

create table texts(
text_id int,
email_id int,
signup_action varchar(30));

-- Q97.New TikTok users sign up with their emails, so each signup
requires a text confirmation to activate the
-- new user's account.
-- 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 * from texts;
select * from emails;

with cte as(
select e.email_id,t.text_id,e.signup_date,t.signup_action
from emails e left join texts t
on e.email_id=t.email_id
group by email_id having text_id=max(text_id) or text_id is null)
select
    round((count(case when signup_action='Confirmed' then 1 else null
end)/count(email_id)),2) as confirm_rate
    from cte;
```

```
----Q98.
--Table: tweets
create table tweets(
tweet_id int,
user_id int,
tweet_date timestamp);

-- Q98. The table below contains information about tweets over a given
period of time. Calculate the 3-day
-- rolling average of tweets published by each user for each date that
a tweet was posted. Output the
-- user id, tweet date, and rolling averages rounded to 2 decimal
places.
-- Hint- Use Count and group by

select * from tweets;

with cte as(
select user_id,tweet_date,count(user_id) as tweet_count
from tweets
group by user_id,date(tweet_date)
)
select user_id,tweet_date,
round(sum(tweet_count) over(partition by user_id order by tweet_date
rows between 2 preceding and current row) /
count(user_id) over(partition by user_id order by tweet_date rows
between 2 preceding and current row),2) as rolling_avg_3_days
from cte
order by user_id;


----Q99.
--Table: activities
--Table: age_breakdown
create table activities(
activity_id int,
user_id int,
activity_type enum ('send', 'open', 'chat'),
time_spent float,
activity_date datetime);

create table age_breakdown(
user_id int,
age_bucket enum('21-25', '26-30', '31-35')
);

-- Q99.Assume you are given the tables below containing information on
Snapchat users, their ages, and
```

```
-- their time spent sending and opening snaps. 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.

select * from activities;
select * from age_breakdown;

select * from(
select ab.age_bucket,
round(100.0*sum(case when activity_type='send' then time_spent else 0
end)/sum(time_spent),2) as 'send_perc',
round(100.0*sum(case when activity_type='open' then time_spent else 0
end)/sum(time_spent),2) as 'open_perc'
from activities a left join age_breakdown ab
on a.user_id=ab.user_id
group by age_bucket
order by age_bucket)a
where send_perc <> 0 and open_perc <>0;


----Q100.
--Table: personal_profiles
--Table: employee_company
--Table: company_pages

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(50),
followers int);

-- Q100.The LinkedIn Creator team is looking for power creators who use
their personal profile as a company
-- or influencer page. This means that if someone's Linkedin page has
more followers than all the
-- companies they work for, we can safely assume that person is a Power
Creator. Keep in mind that if a
-- person works at multiple companies, we should take into account the
company with the most followers.
-- Write a query to return the IDs of these LinkedIn power creators in
ascending order.
```

```
select * from personal_profiles;
select * from employee_company;
select * from company_pages;

with cte as(
select ec.personal_profile_id,ec.company_id,cp.name,cp.followers as
company_followers
from employee_company ec left join company_pages cp
on ec.company_id=cp.company_id)
select profile_id from personal_profiles p
where followers>(select max(company_followers) from cte where
personal_profile_id=p.profile_id group by personal_profile_id)
order by profile_id;
```