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

```
select * from CITY
where countrycode = 'USA' and population > 100000;
```

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

```
select name from CITY
        where countrycode = 'USA' and population > 120000;
```

Q3. Query all columns (attributes) for every row in the CITY table.

```
Select * from CITY;
```

Q4. Query all columns for a city in CITY with the ID 1661.

```
select * from CITY where ID = 1661;
```

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

```
select * from CITY where countrycode = 'JPN';
```

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

```
Select name from CITY  where countrycode = ‘JPN’;
```

Q7. Query a list of CITY and STATE from the STATION table.

```
select city, state from station;
```

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

```
select distinct(city) from station
where id%2 = 0;
```

Q9. Find the difference between the total number of CITY entries in the table and the number of distinct CITY entries in the table.

```
select (count(city)-count(distinct(city))) from station;
```

Q10. Query the two cities in STATION with the shortest and longest CITY names, as well as their respective lengths (i.e.: number of characters in the name). If there is more than one smallest or largest city, choose the one that comes first when ordered alphabetically.

```
(select city, length(city) from station order by length(city) asc, city asc limit 1)
UNION
(select city, length(city) as len from station order by length(city) DESC, city DESC limit 1);
```

Q11. Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) from STATION. Your result cannot contain duplicates.

```
select distinct(city) from station
where (city LIKE"A%") OR (city LIKE"E%") OR (city LIKE"I%") OR (city LIKE"O%") or (city LIKE"U%");
```

Q12. Query the list of CITY names ending with vowels (a, e, i, o, u) from STATION. Your result cannot contain duplicates.

```
select distinct(city) from station
where (city LIKE"%a") OR (city LIKE"%e") OR (city LIKE"%i") OR (city LIKE"%o") or (city LIKE"%u");
```

Q13. Query the list of CITY names from STATION that do not start with vowels. Your result cannot contain duplicates.

```
select distinct(city) from station
where city rlike '^[^aeiouAEIOU].*';
```

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

```
select distinct(city) from station
where (city NOT LIKE '%a') and (city NOT LIKE '%e') and (city NOT LIKE '%i') and
(city NOT LIKE '%o') and (city NOT LIKE '%u');
```

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

```
select distinct(city) from station
where (city rlike '^[^aeiouAEIOU].*') or (city NOT LIKE '%a') or (city NOT LIKE '%e') or (city NOT
LIKE '%i') or
(city NOT LIKE '%o') or (city NOT LIKE '%u');
```

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

```
select distinct(city) from station
where (city rlike '^[^aeiouAEIOU].*') and (city NOT LIKE '%a') and (city NOT LIKE '%e') and (city
NOT LIKE '%i') and (city NOT LIKE '%o') and (city NOT LIKE '%u');
```

Q17. Write an SQL query that reports the products that were only sold in the first quarter of 2019. That is, between 2019-01-01 and 2019-03-31 inclusive.

```
select s.product_id, p.product_name
from sales s, product p
where s.product_id = p.product_id
group by s.product_id, p.product_name
having min(s.sale_date) >= '2019-01-01' and max(s.sale_date) <= '2019-03-31';
```

Q18. Write an SQL query to find all the authors that viewed at least one of their own articles. Return the result table sorted by id in ascending order.

```
select distinct(author_id) as id from views
where author_id = viewer_id
order by author_id;
```

Q19. Write an SQL query to find the percentage of immediate orders in the table, rounded to 2 decimal places.

```
select round((sum(order_date = customer_pref_delivery_date)*100/COUNT(*)),2)
as immediate_percentage from delivery;
                                OR
select round((select count(customer_id)*100 from Delivery
where order_date = customer_pref_delivery_date)/count(*),2) from Delivery ;
```

Q20. Write an SQL query to find the ctr of each Ad. Round ctr to two decimal points. Return the result table ordered by ctr in descending order and by ad_id in ascending order in case of a tie.

```
select ad_id, ifnull(round(
sum(action='Clicked')*100/(sum(action='Clicked')+sum(action='Viewed')),2),0) as ctr
from ads
group by ad_id
order by ctr desc;
```

Q21. Write an SQL query to find the team size of each of the employees. Return result table in any order.

```
select e.employee_id, (select count(team_id) from employee where e.team_id = team_id) as
team_size
from employee e;
```

OR

```
SELECT employee_id, COUNT(team_id) OVER (PARTITION BY team_id) team_size
FROM employee;
```

Q22. Write an SQL query to find the type of weather in each country for November 2019. The type of weather is:

- Cold if the average weather_state is less than or equal 15,
- Hot if the average weather_state is greater than or equal to 25, and
- Warm otherwise.

```
  SELECT c.country_name,
    CASE
       WHEN AVG(w.weather_state ) <= 15 THEN 'Cold'
       WHEN AVG(w.weather_state ) >= 25 THEN 'Hot'
       ELSE 'Warm'
    END AS weather_type
FROM countries c
INNER JOIN weather w ON c.country_id = w.country_id
```

```
WHERE w.day BETWEEN '2019-11-01' AND '2019-11-30'
GROUP BY c.country_id;
```

Q23. Write an SQL query to find the average selling price for each product. average_price should be rounded to 2 decimal places.

```
select p.product_id,round(sum(p.price*u.units)/sum(u.units),2) as average_price
from Prices p
inner join UnitsSold u
on p.product_id = u.product_id
where u.purchase_date between p.start_date and p.end_date
group by p.product_id;
```

Q24. Write an SQL query to report the first login date for each player. Return the result table in any order.

```
select distinct(player_id),event_date as first_login from Activity
group by player_id;

                OR
select player_id, min(event_date) as first_login
from Activity
group by player_id;
```

Q25. Write an SQL query to report the device that is first logged in for each player. Return the result table in any order.

```
with t1 as(select distinct(player_id), min(event_date) as first_date from Activity
group by player_id), t2 as(select * from Activity)
select t1.player_id, t2.device_id from t1 inner join t2
on t1.player_id = t2.player_id
where t1.first_date = t2.event_date;
```

Q26. Write an SQL query to get the names of products that have at least 100 units ordered in February 2020 and their amount. Return result table in any order.

```
select tmp.product_name , tmp.unit
from (select p.product_name, sum(o.unit) as unit from Orders o INNER JOIN Products p
on o.product_id = p.product_id
where o.order_date BETWEEN '2020-02-01' AND '2020-02-29'
GROUP BY o.product_id) tmp
WHERE tmp.unit >= 100;
```

Q27. Write an SQL query to find the users who have valid emails. A valid e-mail has a prefix name and a domain where:

● The prefix name is a string that may contain letters (upper or lower case), digits, underscore '_', period '.', and/or dash '-'. The prefix name must start with a letter.

● The domain is '@leetcode.com'.

Return the result table in any order.

```
SELECT * FROM Users
WHERE mail REGEXP '^[a-zA-Z]+[a-zA-Z0-9_.-]*(@leetcode[.]com)$';
```

Q28. Write an SQL query to report the customer_id and customer_name of customers who have spent at least $100 in each month of June and July 2020. Return the result table in any order.

```
select tmp.customer_id, tmp.name from
(with tbl1 as(select
o.customer_id,year(o.order_date) as year,month(o.order_date) as month,
SUM(o.quantity*p.price) as price
from Orders o inner join Product p
on o.product_id = p.product_id
where year(o.order_date)=2020 and month(o.order_date) in (6,7)
group by o.customer_id,year(o.order_date),month(o.order_date)) ,
tbl2 as(select * from Customers)
select tbl1.*,tbl2.name from tbl1 inner join tbl2
on tbl1.customer_id = tbl2.customer_id) tmp
having sum(case when tmp.month = 6
then tmp.price end) >= 100
and
sum(case when tmp.month = 7
then tmp.price end) >= 100;
```

Q29. Write an SQL query to report the distinct titles of the kid-friendly movies streamed in June 2020. Return the result table in any order.

```
SELECT tmp.title FROM(select t.content_id,t.program_date,c.title,c.kids_content
from TVProgram t inner join Content c
ON t.content_id = c.content_id
WHERE date_format(t.program_date, '%Y-%m')='2020-06') tmp
WHERE tmp.Kids_content = 'Y';
```

Q30. Write an SQL query to find the npv of each query of the Queries table. Return the result table in any order.

```
SELECT  q.id, q.year, ifnull(n.npv,0) FROM Queries q LEFT JOIN NPV n
ON (q.id,q.year) = (n.id ,n.year);
```

Q31. Write an SQL query to find the npv of each query of the Queries table. Return the result table in any order.

```
SELECT  q.id, q.year, ifnull(n.npv,0) FROM Queries q LEFT JOIN NPV n
ON (q.id,q.year) = (n.id ,n.year);
```

Q32. Write an SQL query to show the unique ID of each user, If a user does not have a unique ID replace just show null. Return the result table in any order.

```
select u.unique_id,e.name from Employees e
left join EmployeeUNI u
ON e.id = u.id;
```

Q33. Write an SQL query to report the distance travelled by each user. Return the result table ordered by travelled_distance in descending order, if two or more users travelled the same distance, order them by their name in ascending order.

```
SELECT tmp.name, sum(distance) AS travelled_distance FROM
   (select u.name,ifnull(r.distance,0) as distance from Users u left join Rides r
on u.id = r.user_id) tmp
GROUP BY tmp.name
ORDER BY travelled_distance DESC,tmp.name;
```

Q34. Repeated as Q.29

Q35. Write an SQL query to:

- Find the name of the user who has rated the greatest number of movies. In case of a tie, return the lexicographically smaller user name.

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

```
SELECT rum.name as output FROM (SELECT tmp.name, count(tmp.rating) as freq_rating FROM(
select m.*,e.title,u.name
from MovieRating m inner join Movies e
on m.movie_id = e.movie_id
inner join Users u on u.user_id = m.user_id
) tmp
group by tmp.user_id
ORDER BY freq_rating DESC, tmp.name
limit 1) rum
UNION
SELECT tum.title as output from(SELECT tmp.title, sum(tmp.rating) as max_rating
FROM(select m.*,e.title,u.name
from MovieRating m inner join Movies e
on m.movie_id = e.movie_id
inner join Users u on u.user_id = m.user_id
where month(m.created_at)=02) tmp
GROUP BY tmp.title
ORDER BY max_rating DESC, tmp.title
limit 1) tum;
```

Q36. Same as Q.33

Q37. Same as Q32

Q38. Write an SQL query to find the id and the name of all students who are enrolled in departments that no longer exist. Return the result table in any order.

```
SELECT tmp.id, tmp.name FROM
(select s.*,IFNULL(d.id,0) as dept_id, d.name as dept from Students s left join Departments d
ON d.id = s.department_id) tmp
WHERE tmp.dept_id = 0;
```

Q39. Write an SQL query to report the number of calls and the total call duration between each pair of distinct persons (person1, person2) where person1 < person2. Return the result table in any order.

```
select least(from_id,to_id) as person1, greatest(from_id,to_id) as person2,
count(*) as call_count, sum(duration) as total_duration
from Calls
group by person1,person2;
```

Q40. Write an SQL query to find the average selling price for each product. average_price should be rounded to 2 decimal places. Return the result table in any order.

```
SELECT tmp.product_id, ROUND(SUM(tmp.price)/SUM(tmp.units),2) AS average_price
FROM (SELECT p.product_id,p.price*u.units as price, u.units
FROM Prices p JOIN UnitsSold u
ON p.product_id = u.product_id
WHERE u.purchase_date BETWEEN p.start_date AND p.end_date) tmp
GROUP BY tmp.product_id;
```

Q41. Write an SQL query to report the number of cubic feet of volume the inventory occupies in each warehouse. Return the result table in any order.

```
SELECT tmp.name AS warehouse_name , SUM(tmp.volume) AS Volume
FROM(SELECT w.name, (w.units*p.Width*p.Length*p.Height) AS volume
FROM Warehouse w JOIN Products p
ON w.product_id = p.product_id) tmp
GROUP BY tmp.name;
```

Q42. Write an SQL query to report the difference between the number of apples and oranges sold each day. Return the result table ordered by sale_date.

```
with
 t1 AS(SELECT sale_date, sold_num FROM Sales
 WHERE fruit = 'apples'),
 t2 AS(SELECT sale_date, sold_num FROM Sales
 WHERE fruit = 'oranges')
 SELECT t1.sale_date, (t1.sold_num-t2.sold_num) AS diff
 FROM t1 JOIN t2 ON t1.sale_date=t2.sale_date;
```

Q43. 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.

```
SELECT round(count(b.player_id)/count(a.player_id),2) AS fraction
FROM(SELECT player_id, MIN(event_date) AS event_date FROM Activity
GROUP BY player_id) a LEFT JOIN Activity b
ON  a.player_id=b.player_id AND a.event_date+1=b.event_date;
```

Q44. Write an SQL query to report the managers with at least five direct reports. Return the result table in any order.

```
SELECT name FROM Employee WHERE id IN
(SELECT managerId FROM Employee
GROUP BY managerId
HAVING (COUNT(distinct(id))) >=5);
```

Q45. Write an SQL query to report the respective department name and number of students majoring in each department for all departments in the Department table (even ones with no current students). Return the result table ordered by student_number in descending order. In case of a tie, order them by dept_name alphabetically.

```
SELECT d.dept_name,COUNT(s.dept_id) AS student_number
FROM Department d LEFT JOIN Student s
ON d.dept_id = s.dept_id
GROUP BY d.dept_id,s.dept_id
ORDER BY student_number DESC,d.dept_name;
```

Q46. Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table. Return the result table in any order.

```
SELECT tmp.customer_id FROM
(select customer_id, count(DISTINCT(p.product_key)) as `key`
from Customer c left join Product p
on c.product_key = p.product_key
group by c.customer_id) tmp
WHERE tmp.key IN (SELECT COUNT(*) FROM Product);
```

Q47. Write an SQL query that reports the most experienced employees in each project. In case of a tie, report all employees with the maximum number of experience years. Return the result table in any order.

```
SELECT tmp.project_id,tmp.employee_id
FROM(SELECT p.project_id, p.employee_id, e.experience_years,
dense_rank() OVER (PARTITION BY p.project_id ORDER BY e.experience_years DESC) AS rank_id
FROM Project p JOIN Employee e
ON p.employee_id = e.employee_id) tmp
WHERE tmp.rank_id = 1;
```

Q48. Write an SQL query that reports the books that have sold less than 10 copies in the last year, excluding books that have been available for less than one month from today. Assume today is 2019-06-23. Return the result table in any order.

```
select Books.book_id, name from Books join Orders
on Books.book_id = Orders.book_id
where available_from < '2019-05-23'
and dispatch_date between '2018-06-23' and '2019-06-23'
group by Books.book_id
having sum(quantity) < 10;
```

Q49. Write a SQL query to find the highest grade with its corresponding course for each student. In case of a tie, you should find the course with the smallest course_id. Return the result table ordered by student_id in ascending order.

```
SELECT tmp.student_id,tmp.course_id,tmp.grade FROM(select *,
dense_rank() over (partition by student_id order by grade desc,course_id) as rank1
from Enrollments) tmp
WHERE tmp.rank1 = 1;
```

Q50. The winner in each group is the player who scored the maximum total points within the group. In the case of a tie, the lowest player_id wins. Write an SQL query to find the winner in each group. Return the result table in any order.

```
SELECT t.group_id, t.first_player
FROM(SELECT tmp.*,dense_rank() over(partition by tmp.group_id
ORDER BY tmp.group_id,tmp.first_score desc,tmp.first_player) as rank1
FROM((select m.first_player,m.first_score,p.group_id from Matches m
join Players p on m.first_player = p.player_id)
UNION
(select m.second_player,m.second_score,p.group_id from Matches m
join Players p on m.second_player = p.player_id))tmp) t
WHERE t.rank1 =1;
```

Q51. Write an SQL query to report the name, population, and area of the big countries. Return the result table in any order.

```
SELECT t.name , t.population, t.area
FROM(select name , population, area,
CASE
   when population >= 25000000 or area >= 3000000 then 'big'
   else 'small'
   end as country_type
from World) t
WHERE t.country_type = 'big'
```

Q52. 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  t.name
FROM(select id,name,IFNULL(referee_id,0) as refer_id
from Customer) t
WHERE t.refer_id !=2;
```

Q53. Write an SQL query to report all customers who never order anything. Return the result table in any order.

```
SELECT t.name as Customers
FROM(SELECT c.*, IFNULL(o.customerId,0) as customer FROM Customers c LEFT JOIN Orders o
ON c.id = o.customerId) t
WHERE t.customer = 0;
```

Q54. 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) as team_size
FROM Employee
ORDER BY team_id;
```

Q55. A telecommunications company wants to invest in new countries. The company intends to invest in the countries where the average call duration of the calls in this country is strictly greater than the global average call duration. Write an SQL query to find the countries where this company can invest. Return the result table in any order.

```
SELECT y.name FROM
(SELECT tmp.name, SUM(tmp.duration) as duration
FROM((select a.name,b.duration
FROM(with t1 as(select *, SUBSTRING_INDEX(phone_number,'-',1) AS country_code from Person),
t2 as(select * from Country)
SELECT t1.id,t1.country_code,t2.name
FROM t1 JOIN t2 ON t1.country_code = t2.country_code) a JOIN Calls b
ON a.id = b.caller_id)
UNION
(select a.name,b.duration
FROM(with t1 as(select *, SUBSTRING_INDEX(phone_number,'-',1) AS country_code from Person),
t2 as(select * from Country)
SELECT t1.id,t1.country_code,t2.name
FROM t1 JOIN t2 ON t1.country_code = t2.country_code) a JOIN Calls b
ON a.id = b.callee_id)) tmp
group by tmp.name
ORDER BY duration DESC
limit 1) y;
```

Q56. Reprated Question

Q57. 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 t.customer_number FROM
(SELECT distinct(customer_number),
COUNT(order_number) over(partition by customer_number) as freq
FROM Orders
ORDER BY freq DESC) t
LIMIT 1;
```

Q58. Write an SQL query to report all the consecutive available seats in the cinema. Return the result table ordered by seat_id in ascending order.

```
select distinct(c2.seat_id) from Cinema c1 JOIN Cinema c2
ON c1.seat_id = c2.seat_id+1 or c1.seat_id = c2.seat_id-1
WHERE c2.free = 1 and c2.free = c1.free;
```

Q59. Write an SQL query to report the names of all the salespersons who did not have any orders related to the company with the name "RED". Return the result table in any order.

```
SELECT tmp.name FROM
(SELECT s.name,IFNULL(t.sal_id,0) as new_id FROM
(SELECT o.sales_id as sal_id FROM Orders o JOIN Company c
ON o.com_id = c.com_id
WHERE c.name = 'RED') as t RIGHT JOIN SalesPerson s
ON t.sal_id = s.sales_id) tmp
WHERE tmp.new_id = 0;
```

Q60. 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 *,
   case
      when (x+y)>z AND (y+z)>x AND (z+x)>y then 'Yes'
      else 'No'
      end as triangle
 from Triangle;
```

Q61. Write an SQL query to report the shortest distance between any two points from the Point table.

```
select (p1.x-p2.x) as shortest from Point p1 JOIN Point p2
ON p1.x != p2.x AND p1.x > p2.x
ORDER BY shortest ASC
LIMIT 1;
```

Q62. Write a SQL query for a report that provides the pairs (actor_id, director_id) where the actor has cooperated with the director at least three times. Return the result table in any order.

```
SELECT actor_id,director_id FROM ActorDirector
WHERE actor_id = director_id
GROUP BY director_id
HAVING COUNT(actor_id) = 3;
```

Q63. Write an SQL query that reports the product_name, year, and price for each sale_id in the Sales table. Return the resulting table in any order.

```
SELECT p.product_name, s.year, s.price
FROM Sales s JOIN Product p
ON p.product_id = s.product_id;
```

Q64. 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 p.project_id,round(AVG(e.experience_years),2) as average_years
from Project p JOIN Employee e
on p.employee_id = e.employee_id
GROUP BY p.project_id;
```

Q65. Write an SQL query that reports the best seller by total sales price, If there is a tie, report them all. Return the result table in any order.

```
with t1 AS(SELECT MAX(t.Max_Price) as max_price FROM
(SELECT seller_id, SUM(price) AS Max_Price FROM Sales
GROUP BY seller_id
ORDER BY Max_Price DESC) t),
t2 AS(select seller_id, SUM(price) as Max_Price from Sales
GROUP BY seller_id
ORDER BY Max_Price DESC)
SELECT t2.seller_id FROM
t1 JOIN t2
WHERE t2.Max_Price = t1.max_price;
```

Q66. Write an SQL query that reports the buyers who have bought S8 but not iPhone. Note that S8 and iPhone are products present in the Product table.

```
SELECT s.buyer_id FROM Sales s JOIN Product p
ON s.product_id = p.product_id
WHERE p.product_name = 'S8' and p.product_name != "iPhone" ;
```

Q67. You are the restaurant owner and you want to analyse a possible expansion (there will be at least one customer every day). 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 tmp.visited_on, tmp.amount, round(tmp.average_amount,2) as average_amount
FROM(SELECT t.visited_on, t.amount,
AVG(t.amount) over(ORDER BY t.visited_on rows BETWEEN 6 preceding AND CURRENT ROW) as
average_amount
FROM (select visited_on, sum(amount) as amount
    from Customer group by visited_on ORDER BY visited_on) t) tmp
WHERE day(tmp.visited_on)>=(SELECT MIN(day(visited_on))+6 FROM Customer)
```

Q68. Write an SQL query to find the total score for each gender on each day. Return the result table ordered by gender and day in ascending order.

```
(SELECT gender , day,
SUM(score_points) OVER (ORDER BY day ASC) as total
FROM Scores
WHERE gender = 'F')
UNION
(SELECT gender , day,
SUM(score_points) OVER (ORDER BY day ASC) as total
FROM Scores
WHERE gender = 'M');
```

Q69. Write an SQL query to find the start and end number of continuous ranges in the table Logs. Return the result table ordered by start_id.

```
SELECT MIN(t.log_id) AS start_id, MAX(t.log_id) AS end_id FROM
(select *,log_id - row_number() over (order by log_id) as diff from Logs) t
GROUP BY t.diff;
```

Q70. Write an SQL query to find the number of times each student attended each exam. Return the result table ordered by student_id and subject_name.

```
WITH t1 AS(SELECT * FROM Students JOIN Subjects),
t2 AS(select student_id,subject_name,count(subject_name) as freq from Examinations
GROUP BY student_id, subject_name)
SELECT t1.student_id, t1.student_name, t1.subject_name,IFNULL(t2.freq,0) as attended_exams
FROM t1 LEFT JOIN t2
ON t1.student_id = t2.student_id AND t1.subject_name = t2.subject_name
ORDER BY t1.student_id,t1.subject_name;
```

Q71. Write an SQL query to find employee_id of all employees that directly or indirectly report their work to the head of the company. The indirect relation between managers will not exceed three managers as the company is small. Return the result table in any order

```
select employee_id as EMPLOYEE_ID from Employees where manager_id in
(select employee_id from Employees WHERE manager_id in
(select employee_id from Employees where manager_id =1))
and employee_id !=1;
```

Q72. Write an SQL query to find for each month and country, the number of transactions and their total amount, the number of approved transactions and their total amount. Return the result table in any order.

```
SELECT tmp.month, tmp.country,count(tmp.state) as trans_count,
    SUM(tmp.app_count) as approved_count,sum(tmp.amount) as trans_total_amount,
    SUM(tmp.amnt) as approved_total_amount
FROM(SELECT t.country, t.state, t.amount, substring_index(t.trans_date,'-',2) as
month,s.app_count,s.amnt
FROM(select id as new_id, case
        when state='approved' then 1 else 0 end as app_count, CASE
        when state='approved' then amount else 0 end as amnt
from Transactions) s JOIN Transactions t
ON s.new_id = t.id) tmp
GROUP BY tmp.tmp.month,tmp.country;
```

Q73. Write an SQL query to find the average daily percentage of posts that got removed after being reported as spam, rounded to 2 decimal places.

```
SELECT round(AVG(tmp.daily_percent)) as average_daily_percent FROM
(SELECT SUM(t.num)/SUM(t.monitor)*100 as daily_percent FROM
(SELECT a.action_date, (a.extra='spam') as monitor, (r.post_id != 'NULL') as num
FROM Actions a LEFT JOIN Removals r
```

```
ON a.post_id=r.post_id and a.extra = 'spam') t
GROUP BY t.action_date) tmp;
```

Q74. Repeated as Previous Question

Q75. Same as Q74 which is again repeated

Q76. 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 company_id, employee_id, employee_name ,
      CASE
        when salary<1000 then salary
        when salary>=1000 and salary<=10000 then round(salary*76/100)
        else round(salary*51/100) end as salary
   FROM Salaries;
```

Q.77 Repeated as Privious Question.

Q.78 Write an SQL query to evaluate the boolean expressions in Expressions table. Return the result table in any order.

```
SELECT t.*,
CASE
   when t.operator = '>' AND t.left_operand > t.right_operand then 'True'
   when t.operator = '<' AND t.left_operand < t.right_operand then 'True'
   when t.operator = '=' AND t.left_operand = t.right_operand then 'True'
   else 'False' end as value FROM
(select v.value as left_operand, e.operator, l.value as right_operand
from Expressions e join Variables v
ON e.left_operand = v.name
join Variables l ON e.right_operand = l.name) t;
```

Q79. Repeated as Previous Question.

Q80. Repeated as Previous Question.

Q81. Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.

```
select name from students
where marks > 75
order by RIGHT(name, 3), id;
```

Q82. 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;
```

Q83. Write a query that prints a list of employee names (i.e.: the *name* attribute) for employees in **Employee** having a salary greater than per month who have been employees for less than months. Sort your result by ascending *employee_id*.

```
select name from Employee
where salary > 2000 and months < 10;
```

Q84. Write a query identifying the type of each record in the TRIANGLES table using its three side lengths. Output one of the following statements for each record in the table:

- Equilateral: It's a triangle with sides of equal length.
- Isosceles: It's a triangle with sides of equal length.
- Scalene: It's a triangle with sides of differing lengths.
- Not A Triangle: The given values of A, B, and C don't form a triangle.

```
select
    case
      when A=B and B=C and A+B>c and A+C>B and B+C>A then 'Equilateral'
      when ((A=B and B!=C) or (A=C and B!=C) or (C=B and B!=A))
      and (A+B>c and A+C>B and B+C>A) then 'Isosceles'
      when A!=B and B!=C and A+B>c and A+C>B and B+C>A then 'Scalene'
        else 'Not A Triangle' end as types
    from triangles;
```

Q85. 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.
Output the year (in ascending order) partitioned by product id, current year's spend, previous year's spend and year-on-year growth rate (percentage rounded to 2 decimal places).

```
SELECT t.year,t.product_id,t.curr_year_spend,t.prev_year_spend,
  ROUND(100*(t.curr_year_spend-t.prev_year_spend)/t.prev_year_spend,2)  AS yoy_rate
FROM(WITH yearly_spend
AS(SELECT EXTRACT(year FROM transaction_date) AS year,product_id,
   spend AS curr_year_spend
  FROM user_transactions)
SELECT *, LAG(curr_year_spend, 1) OVER (PARTITION BY product_id
   ORDER BY product_id, year) AS prev_year_spend
FROM yearly_spend) t;
```

Q86. Amazon wants to maximise the number of items it can stock in a 500,000 square feet warehouse. It wants to stock as many prime items as possible, and afterwards use the remaining square footage to stock the most number of non-prime items. 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.

```
WITH
 t1 AS(select item_type, SUM(square_footage) as area_prime,
  COUNT(item_type) as cnt FROM inventory
  GROUP BY item_type),
 t2 AS(SELECT t1.item_type,TRUNC((500000/t1.area_prime)*t1.cnt,0) as item_count,
  500000-trunc(500000/t1.area_prime,0)*t1.area_prime as prime_rem_area
  FROM t1
  WHERE t1.item_type = 'prime_eligible'),
 t3 AS(select t1.item_type,
  trunc(t2.prime_rem_area/t1.area_prime,0)*t1.cnt AS item_count
  FROM t1,t2
  WHERE t1.item_type='not_prime' AND t2.item_type = 'prime_eligible')
 SELECT t3.item_type, t3.item_count from t3
UNION
 SELECT t2.item_type, t2.item_count from t2
ORDER BY item_type DESC;
```

Q87. Assume you have the table below containing information on Facebook 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).
Hint: An active user is a user who has user action ("sign-in", "like", or "comment") in the current month and last month.

```
with
 t1 AS(SELECT *,EXTRACT(month FROM event_date) AS month FROM user_actions
  WHERE (event_date BETWEEN '07/01/2022 12:00:00' AND '07/31/2022 12:00:00')
  AND (event_type='sign-in')
  ORDER BY user_id,event_date),
 t2 AS(SELECT *,EXTRACT(month FROM event_date) AS month FROM user_actions
  WHERE (event_date BETWEEN '07/01/2022 12:00:00' AND '07/31/2022 12:00:00')
  AND (event_type != 'sign-in')
  ORDER BY user_id,event_date)
SELECT DISTINCT(t.month),
 SUM(t.monthly_active_users) AS monthly_active_users
 FROM(SELECT t1.month,
  COUNT(DISTINCT(t1.user_id  )) AS monthly_active_users
  FROM t1 JOIN t2
  ON t1.user_id=t2.user_id AND t2.event_date>=t1.event_date
  GROUP BY t1.month, t1.user_id) t
 GROUP BY t.monthly_active_users,t.month;
```

Q88. Google's marketing team is making a Superbowl commercial and needs a simple statistic to put on their TV ad: the median number of searches a person made last year.
However, at Google scale, querying the 2 trillion searches is too costly. Luckily, you have access to the summary table which tells you the number of searches made last year and how many Google users fall into that bucket.
Write a query to report the median of searches made by a user. Round the median to one decimal point.

```
SELECT
ROUND(PERCENTILE_CONT(0.50) WITHIN GROUP (ORDER BY t.searches)::DECIMAL, 1) AS median
```

```
FROM(SELECT searches
    FROM search_frequency
    GROUP BY searches,
    GENERATE_SERIES(1, num_users)) t;
```

Q.89 Write a query to update the Facebook advertiser's status using the daily_pay table. Advertiser is a two-column table containing the user id and their payment status based on the last payment and 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.

```
SELECT t.*
FROM((SELECT a.user_id,CASE
 WHEN ((a.status='NEW') OR (a.status='EXISTING') OR (a.status='CHURN') OR
 (a.status='RESURRECT'))
 AND (d.paid is NULL) THEN 'CHURN'
 WHEN ((a.status='NEW') OR (a.status='EXISTING') OR (a.status='RESURRECT'))
 AND (d.paid is NOT NULL) THEN 'EXISTING'
 WHEN a.status='CHURN' AND d.paid is NOT NULL THEN 'RESURRECT'
 END AS new_status
 FROM advertiser a LEFT JOIN daily_pay d
 ON a.user_id=d.user_id)
 UNION
 (
 SELECT d.user_id,CASE
  WHEN a.status is NULL AND d.paid is NOT NULL THEN 'NEW'
  END AS new_status
  FROM advertiser a FULL OUTER JOIN daily_pay d
  ON a.user_id=d.user_id
  WHERE a.user_id is NULL
 )) t
 ORDER BY t.user_id;
```

Q93. 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 day, SUM(score_points) over(PARTITION BY gender order by day), gender
FROM Scores;
```

Q94. Repeated Question.

Q95. 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.

```
SELECT
ROUND(PERCENTILE_CONT(0.50) WITHIN GROUP (ORDER BY t.num)::DECIMAL, 1) AS median
FROM(select num from numbers
group by num, generate_series(1,frequency)
ORDER by num) t;
```

Q96. 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.

```
with t1 as(SELECT SUM(amount)/COUNT(amount) as avg_salary,
   substring_index(pay_date,'-',2) as pay_month
   from Salary
   GROUP BY substring_index(pay_date,'-',2)),
t2 as (SELECT substring_index(s.pay_date,'-',2) as pay_month, e.department_id,
   sum(s.amount)/ COUNT(s.amount) as avg_amount
   FROM Salary s JOIN Employee e
   ON s.employee_id=e.employee_id
   GROUP BY e.department_id,substring_index(s.pay_date,'-',2))
SELECT t2.pay_month, t2.department_id,
CASE
   when t2.avg_amount > t1.avg_salary then 'Higher'
   when t2.avg_amount = t1.avg_salary then 'Same'
   when t2.avg_amount < t1.avg_salary then 'Lower'
   end as comparison
from t2 join t1 ON t1.pay_month = t2.pay_month
ORDER BY t2.department_id,t2.pay_month;
```

Q97. 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 t1.install_date as install_dt, count(t1.install_date) as installs,
   round(count(t2.event_date) / count(*), 2) as Day1_retention
from (
   select player_id, min(event_date) as install_date
   from Activity
   group by 1
) t1
left join Activity t2
on date_add(t1.install_date, interval 1 day) = t2.event_date
   and t1.player_id = t2.player_id
group by 1
order by 1
```

Q98. The winner in each group is the player who scored the maximum total points within the group. In the case of a tie, the lowest player_id wins. Write an SQL query to find the winner in each group.

```
select y.group_id, y.player as player_id
from(select z.group_id, z.player, max(z.Tot_score)
from (select t.group_id, t.player, sum(score) as Tot_score
from
(select m.match_id, m.first_player as player, m.first_score as score, p.group_id
from Players p join Matches m
on p.player_id = m.first_player
union
select m.match_id, m.second_player as player, m.second_score as score, p.group_id
from Players p join Matches m
on p.player_id = m.first_player) t
group by t.group_id, t.player
```

```
order by t.group_id asc, Tot_score desc, t.player asc) z
group by z.group_id) y;
```

Q99. 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.

```
with t1 as (select e.exam_id, e.student_id, e.score ,s.student_name,
max(e.score) over(partition by e.exam_id) as max_status,
min(e.score) over(partition by e.exam_id) as min_status
from Exam e inner join Student s
on e.student_id = s.student_id),
t2 as (select t1.exam_id, t1.student_id, t1.score ,t1.student_name
from t1
where t1.score < t1.max_status and t1.score > t1.min_status),
t3 as(select t1.exam_id, t1.student_id, t1.score ,t1.student_name
from t1
where t1.score = t1.max_status or t1.score = t1.min_status)
select distinct(t2.student_id),t2.student_name
from t2 left join t3
on t2.student_id = t3.student_id
where t3.student_name is null;
```

Q100. Repeated as previous Question Q99

Q101. 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.

```
select tmp.username, tmp.activity, tmp.startDate, tmp.endDate
from(select t.* ,case
        when t.fre >1 and t.activity = t.match_actv then 1
   when t.fre = 1 then 1
   else 0 end as pair
from(select *,nth_value(activity,2) over(partition by username) as match_actv,
count(*) over(partition by username) as fre
from UserActivity
order by endDate desc) t) tmp
where tmp.pair = 1;
```

Q102. Repeated as previous question

Q103. Repeated as previous question

Q104. Repeated as previous question

Q105. 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;
```

Q106. Repeated as previous question

Q108. We define an employee's total earnings to be their monthly salary * months worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table. Write a query to find the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as 2 space-separated integers

```
select * from (
select months * salary, count(*)
from employee
group by months * salary
order by 1 desc
) where rownum = 1;
```

Q109. Generate the following two result sets:

1. Query an alphabetically ordered list of all names in OCCUPATIONS, immediately followed by the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S). Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output them in the following format:

```
SELECT CONCAT(NAME,'(',LEFT(OCCUPATION, 1),')')
FROM OCCUPATIONS
ORDER BY NAME;
```

2. where [occupation_count] is the number of occurrences of an occupation in OCCUPATIONS and [occupation] is the lowercase occupation name. If more than one Occupation has the same [occupation_count], they should be ordered alphabetically.

```
SELECT CONCAT('There are a total of ', COUNT(*), ' ', LOWER(OCCUPATION), 's.')
FROM OCCUPATIONS
GROUP BY OCCUPATION
ORDER BY COUNT(*), OCCUPATION;
```

Q110. 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 ROW_NUMBER() OVER (PARTITION BY occupation ORDER BY name) as rn, name,
occupation FROM      occupations)
PIVOT
(MAX(name) FOR occupation IN ('Doctor' as Doctor,'Professor' as Professor, 'Singer' as Singer,
'Actor' as Actor))
ORDER BY rn;
```

Q111. You are given a table, BST, containing two columns: N and P, where N represents the value of a node in Binary Tree, and P is the parent of N.
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 DISTINCT(p) from BST) then 'Inner'
        else 'Leaf'
        end as type
from BST;
```

Q112. 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.

```
select c.company_code, c.founder,
     count(distinct l.lead_manager_code),
     count(distinct s.senior_manager_code),
     count(distinct m.manager_code),
     count(distinct e.employee_code)
from Company as c
join Lead_Manager as l
on c.company_code = l.company_code
join Senior_Manager as s
on l.lead_manager_code = s.lead_manager_code
join Manager as m
on m.senior_manager_code = s.senior_manager_code
join Employee as e
on e.manager_code = m.manager_code
group by c.company_code, c.founder
order by c.company_code;
```

Q113. 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). For example, the output for all prime numbers <=10 would be: 2&3&5&7

```
with numSel as (
   select level num from dual connect by level <= 1000
),
primes as (
   select a.num p
    from numSel a, numSel b
   where b.num <= a.num
   group by a.num
   having count(case a.num/b.num when trunc(a.num/b.num) then 'Y' end) = 2
)
select listagg(p, '&') within group (order by p)
```

```
 from primes
;
```

Q114. 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;
```

Q115. Repeated as previous Question

Q116. Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows 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;
```

Q117. 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
```

Q118. 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;
```

Q119. 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;
```

Q120. Repeated Question as Previous.

Q121. Repeated Question as previous.

Q122. Repeated Question as previous.

Q123. Repeated Question as previous.

Q124. Repeated Question as previous.

Q125. Repeated Question as previous.

Q126. Amazon Web Services (AWS) is powered by fleets of servers. Senior management has requested data-driven solutions to optimise server usage.

Write a query that calculates the total time that the fleet of servers was running. The output should be in units of full days.

```
WITH running_time
AS (
  SELECT
    server_id,
    session_status,
    status_time AS start_time,
    LEAD(status_time) OVER (
      PARTITION BY server_id
      ORDER BY status_time) AS stop_time
  FROM server_utilization
)

SELECT
  DATE_PART('days', JUSTIFY_HOURS(SUM(stop_time - start_time))) AS total_uptime_days
FROM running_time
WHERE session_status = 'start'
  AND stop_time IS NOT NULL;
```

Q127. 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;
```

Other Questions are repeated