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

**Ans.**  SELECT \*

FROM CITY

WHERE population > 100000 AND country code = "USA"

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

**Ans.** SELECT NAME

FROM CITY

WHERE population > 120000 AND country code = "USA"

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

**Ans.** SELECT \*FROM CITY;

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

**Ans.** SELECT \* FROM CITY WHERE ID=1661;

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

**Ans**. select \* from city where country code ="JPN" ;

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

**Ans**. SELECT name FROM CITY WHERE COUNTRYCODE="JPN";

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

**Ans**. 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

**Ans.** 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.

Ans.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

**Ans**.SELECT city, CHAR\_LENGTH(city) FROM station ORDER BY CHAR\_LENGTH(city),

city LIMIT 1;

SELECT city, CHAR\_LENGTH(city) FROM station ORDER BY CHAR\_LENGTH(city) DESC, city 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. Input Format

**Ans**.select distinct city from station where city like 'a%'

or city like 'e%'

or city like 'i%'

or city like 'o%'

or city like 'u%'

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

**Ans.**select distinct city from station where city like '%a' or city like '%e' or city like '%i' or city like '%o' or city like '%u';

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

**Ans**. select distinct city from station where left(city,1) not in ('a','e','i','o','u');

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

**Ans**. select distinct city from station where right(city,1) not in ('a','e','i','o','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

**Ans**. select distinct city from station where left(city,1) not in ('a','e','i','o','u') or right(city,1) not in ('a','e','i','o','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

**Ans**.select distinct city from station where left(city,1) not in ('a','e','i','o','u') and right(city,1) not in ('a','e','i','o','u');

**Q17**.

**Ans**.(select p.product\_id, p.product\_name FROM Product p INNER JOIN Sales s on p.product\_id = s.product\_id where s.sale\_date >= '2019-01-01' and s.sale\_date <= '2019-03-31') EXCEPT (select p.product\_id, p.product\_name FROM Product p INNER JOIN Sales s on p.product\_id = s.product\_id where s.sale\_date < '2019-01-01' OR s.sale\_date > '2019-03-31')

**Q18**.

**Ans.** select distinct author\_id as id from views where author\_id = viewer\_id order by author\_id asc;

**Q19**.

**Ans**.select round((select count(\*) from delivery where order\_date = customer\_pref\_delivery\_date)/count(\*)\*100,2) as immediate\_percentage from delivery;

**Q20**.

**Ans**. select t.ad\_id, (case when base != 0 then round(t.num/t.base\*100,2) else 0 end) as Ctr from (select ad\_id, sum( case when action = 'clicked' or action = 'viewed' then 1 else 0 end) as base, sum( case when action = 'clicked' then 1 else 0 end) as num from ads group by ad\_id)t order by Ctr desc, t.ad\_id asc;

**Q21.**

**Ans**. select employee\_id, count(team\_id) over (partition by team\_id) as team\_size from employee order by employee\_id;

**Q22.**

**Ans.** select c.country\_name, case when avg(weather\_state) <= 15 then 'Cold' when avg(weather\_state) >= 25 then 'Hot' else 'Warm' end as weather\_state from countries c left join weather w on c.country\_id = w.country\_id where month(day) = 11 group by c.country\_name;

**Q23.**

**Ans**.select p.product\_id, round(sum(u.units\*p.price)/sum(u.units),2) as average\_price from prices p left join unitssold u on p.product\_id = u.product\_id where u.purchase\_date >= start\_date and u.purchase\_date <= end\_date group by product\_id order by product\_id;

**Q24.**

**Ans**.select t.player\_id, event\_date as first\_login from (select player\_id, event\_date, row\_number() over(partition by player\_id order by event\_date) as num from activity)t where t.num = 1;

**Q25.**

**Ans**.select t.player\_id, t.device\_id from (select player\_id, device\_id, row\_number() over(partition by player\_id order by event\_date) as num from activity)t where t.num = 1;

**Q26**.

**Ans**.select p.product\_name, sum(o.unit) as unit from Products p left join Orders o on p.product\_id = o.product\_id where month(o.order\_date) = 2 and year(o.order\_date) = 2020 group by p.product\_id having unit >= 100;

**Q27**.

**Ans**.select user\_id, name, mail from Users where mail regexp '^[a-zA-Z]+[a-zA-Z0-9\_\.\-]\*@leetcode[\.]com' order by user\_id;

**Q28**.

**Ans**.select t.customer\_id, t.name from (select c.customer\_id, c.name, sum(case when month(o.order\_date) = 6 and year(o.order\_date) = 2020 then p.price\*o.quantity else 0 end) as june\_spent, sum(case when month(o.order\_date) = 7 and year(o.order\_date) = 2020 then p.price\*o.quantity else 0 end) as july\_spent from Orders o left join Product p on o.product\_id = p.product\_id left join Customers c on o.customer\_id = c.customer\_id group by c.customer\_id) t where june\_spent >= 100 and july\_spent >= 100;

**Q29**.

**Ans**.select c.Title from Content c left join TVProgram t on c.content\_id = t.content\_id where c.Kids\_content = 'Y' and c.content\_type = 'Movies' and month(t.program\_date) = 6 and year(t.program\_date) = 2020;

**Q30**.

**Ans**.select q.\*, coalesce(n.Npv,0) as Npv from Queries q left join NPV n on q.Id = n.Id and q.Year = n.Year;

**Q31**.

**Ans.**select q.\*, coalesce(n.Npv,0) as Npv from Queries q left join NPV n on q.Id = n.Id and q.Year = n.Year;

**Q32**.

**Ans**.select u.unique\_id, e.name from employees e left join employeeUNI u on e.id = u.id;

**Q33.**

**Ans**.select u.name, coalesce(sum(r.distance),0) as travelled\_distance from users u left join rides r on u.id = r.user\_id group by u.name order by travelled\_distance desc, u.name;

**Q34.**

**Ans**.select p.product\_name, sum(o.unit) as unit from Products p left join Orders o on p.product\_id = o.product\_id where month(o.order\_date) = 2 and year(o.order\_date) = 2020 group by p.product\_id having unit >= 100;

**Q35.**

**Ans**.(select t1.name as Results from (select u.name, count(u.user\_id), dense\_rank() over(order by count(user\_id) desc, u.name) as r1 FROM Users u left join MovieRating m on u.user\_id = m.user\_id group by u.user\_id) t1 where r1 = 1) union (select t2.title as Results from (select mo.title, avg(m.rating), dense\_rank() over(order by avg(m.rating)desc, mo.title) as r2 from Movies mo left join MovieRating m on mo.movie\_id = m.movie\_id where month(m.created\_at) = 2 and year(m.created\_at) = 2020 group by m.movie\_id) t2 where r2 = 1);

**Q36.**

**Ans**.select u.name, coalesce(sum(r.distance),0) as travelled\_distance from users u left join rides r on u.id = r.user\_id group by u.name order by travelled\_distance desc, u.name;

**Q37.**

**Ans.**select u.unique\_id, e.name from employees e left join employeeUNI u on e.id = u.id;

**Q38.**

Ans.select id, name from Students where department\_id not in (select id from Departments);

**Q39.**

**Ans.**select t.person1, t.person2, count(\*) as call\_count, sum(t.duration) as total\_duration from (select duration, case when from\_id < to\_id then from\_id else to\_id end as person1, case when from\_id > to\_id then from\_id else to\_id end as person2 from Calls) t group by t.person1, t.person2;

**Q40.**

**Ans**.select p.product\_id, round(sum(u.units\*p.price)/sum(u.units),2) as average\_price from prices p left join unitssold u on p.product\_id = u.product\_id where u.purchase\_date >= start\_date and u.purchase\_date <= end\_date group by product\_id order by product\_id;

**Q41.**

**Ans.**select w.name as warehouse\_name, sum(p.width\*p.length\*p.height\*w.units) as volume from warehouse w left join products p on w.product\_id = p.product\_id group by w.name order by w.name;

**Q42.**

**Ans.**select t.sale\_date, (t.apples\_sold - t.oranges\_sold) as diff from (select sale\_date, max(CASE WHEN fruit = 'apples' THEN sold\_num ELSE 0 END )as apples\_sold, max(CASE WHEN fruit = 'oranges' THEN sold\_num ELSE 0 END )as oranges\_sold FROM sales group by sale\_date) t ORDER BY t.sale\_date;

**Q43.**

**Ans**.select round(t.player\_id/(select count(distinct player\_id) from activity),2) as fraction from ( select distinct player\_id, datediff(event\_date, lead(event\_date, 1) over(partition by player\_id order by event\_date)) as diff from activity ) t where diff = -1;

**Q44.**

**Ans.**select t.name from (select a.id, a.name, count(b.managerID) as no\_of\_direct\_reports from employee a INNER JOIN employee b on a.id = b.managerID group by b.managerID) t where no\_of\_direct\_reports >= 5 order by t.name;

**Q45.**

**Ans**.select d.dept\_name, count(s.dept\_id) as student\_number from department d left join student s on s.dept\_id = d.dept\_id group by d.dept\_id order by student\_number desc, dept\_name;

**Q46.**

**Ans**.select customer\_id from customer group by customer\_id having count(distinct product\_key)=(select count(\*) from product);

**Q47.**

**Ans.**select t.project\_id, t.employee\_id from (select p.project\_id, e.employee\_id, dense\_rank() over(partition by p.project\_id order by e.experience\_years desc) as r from project p left join employee e on p.employee\_id = e.employee\_id) t where r = 1 order by t.project\_id;

**Q48.**

**Ans.**select t1.book\_id, t1.name from ( (select book\_id, name from Books where available\_from < '2019-05-23') t1 left join (select book\_id, sum(quantity) as quantity from Orders where dispatch\_date > '2018-06-23' and dispatch\_date<= '2019-06-23' group by book\_id having quantity < 10) t2 on t1.book\_id = t2.book\_id );

**Q49.**

**Ans.**select t.student\_id, t.course\_id, t.grade from (select student\_id, course\_id, grade, dense\_rank() over(partition by student\_id order by grade desc, course\_id) as r from enrollments) t where r = 1 order by t.student\_id;

**Q50.**

**Ans.**select t2.group\_id, t2.player\_id from ( select t1.group\_id, t1.player\_id, dense\_rank() over(partition by group\_id order by score desc, player\_id) as r from ( select p.\*, case when p.player\_id = m.first\_player then m.first\_score when p.player\_id = m.second\_player then m.second\_score end as score from Players p, Matches m where player\_id in (first\_player, second\_player) ) t1 ) t2 where r = 1;

**Q51.**

**Ans**.select name, population, area from World where area >= 3000000 or population >= 25000000;

**Q52.**

Ans.select name from Customer where refree\_id != 2 or refree\_id is NULL;

**Q53**.

**Ans**.select c.name from Customers c left join Orders o on c.id = o.customerID where o.id is NULL;

**Q54.**

**Ans**.count(team\_id) over(partition by team\_id) as team\_size from Employee order by employee\_id;

**Q55.**

**Ans**.select t3.Name from ( select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg\_call\_duration, avg(t1.duration) over() as global\_average from ((select cl.caller\_id as id, cl.duration from Calls cl) union (select cl.callee\_id as id, cl.duration from Calls cl)) t1 left join (select p.id, c.Name from Person p left JOIN Country c ON cast(left(p.phone\_number,3) as int) = cast(c.country\_code as int)) t2 ON t1.id = t2.id) t3 where t3.avg\_call\_duration > global\_average group by t3.Name;

**Q56.**

**Ans.**select t.player\_id, t.device\_id from (select player\_id, device\_id, row\_number() over(partition by player\_id order by event\_date) as num from activity)t where t.num = 1;

**Q57.**

**Ans**.select customer\_number from Orders group by customer\_number order by count(order\_number) desc limit 1;

**Q58.**

**Ans**.select t.seat\_id from (select seat\_id, lead(seat\_id,1,seat\_id) over(order by seat\_id) as next from Cinema where Free != 0 ) t where next - seat\_id in (0,1) order by seat\_id;

**Q59.**

**Ans.**select Name from SalesPerson where sales\_id not in (select o.sales\_id from Orders o left join Company c on o.com\_id = c.com\_id where c.Name = 'Red');

**Q60.**

**Ans.**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.**

**Ans.**select min(t.diff) as shortest from (select lead(X,1) over(order by X) - X as diff from Point) t;

**Q62.**

Ans.select actor\_id, director\_id from ActorDirector group by actor\_id, director\_id having count(\*) >= 3;

**Q63.**

**Ans.**select p.product\_name, s.year, sum(price) as price from Sales s left join Product p on s.product\_id = p.product\_id group by p.product\_name, s.year;

**Q64.**

**Ans.**select p.project\_id, round(avg(e.experience\_years),2) as average\_years from Project p left join Employee e on p.employee\_id = e.employee\_id group by p.project\_id;

**Q65.**

**Ans.**select t.seller\_id from (select seller\_id , sum(price), dense\_rank() over(order by sum(price) desc) as r from Sales group by seller\_id) t where t.r = 1;

**Q66.**

**Ans.**select buyer\_id from ( select t1.buyer\_id, sum(case when t1.product\_name = 'S8' then 1 else 0 end) as S8\_count, sum(case when t1.product\_name = 'iPhone' then 1 else 0 end) as iphone\_count from ( select s.buyer\_id, p.product\_name from Sales s left join Product p on s.product\_id = p.product\_id ) t1 group by t1.buyer\_id ) t2 where t2.S8\_count = 1 and t2.iphone\_count = 0;

**Q67.**

**Ans**.select t2.visited\_on, t2.amount, t2.average\_amount from (select t1.visited\_on, t1.prev\_date\_interval\_6, round(sum(amount) over(order by visited\_on range between interval '6' day preceding and current row),2) as amount, round(avg(amount) over(order by visited\_on range between interval '6' day preceding and current row),2) as average\_amount from (select visited\_on, sum(amount) as amount, lag(visited\_on,6) over(order by visited\_on) as prev\_date\_interval\_6 from Customer group by visited\_on order by visited\_on) t1 ) t2 where prev\_date\_interval\_6 is not null;

**Q68.**

**Ans**.select gender, day, sum(score\_points) over(partition by gender order by day) as total from Scores group by gender, day order by gender, day;

**Q69.**

**Ans**.select distinct start.log\_id as start\_id, min(end.log\_id) over(partition by start.log\_id) as end\_id from (select log\_id from Logs where log\_id - 1 not in (select \* from Logs)) start, (select log\_id from Logs where log\_id + 1 not in (select \* from Logs)) end where start.log\_id <= end.log\_id;

**Q70.**

**Ans.**select t.student\_id, t.student\_name , t.subject\_name, count(e.subject\_name) as attended\_exams from (select student\_id, student\_name, subject\_name from Students, Subjects) t left join Examinations e on t.student\_id = e.student\_id and t.subject\_name = e.subject\_name group by t.student\_id, t.subject\_name order by t.student\_id, t.subject\_name;

**Q71.**

**Ans.**with recursive new as ( select employee\_id from Employees where employee\_id = 1 union select e2.employee\_id from new e1 inner join Employees e2 on e1.employee\_id = e2.manager\_id ) select \* from new where employee\_id <> 1;

**Q72.**

**Ans.**select month(trans\_date) as Month, Country, count(Id) as trans\_count, sum(case when State = 'approved' then 1 else 0 end) as approved\_count, sum(amount) as trans\_total\_amount, sum(case when State = 'approved' then amount else 0 end) as approved\_total\_amount from Transactions group by Month, Country;

**Q73.**

**Ans.**select round(avg(t.daily\_percent), 2) as average\_daily\_percent from ( select sum(case when remove\_date > action\_date then 1 else 0 end)/ count(tmp.action\_date)\*100 as daily\_percent from ( select post\_id, action\_date, extra from Actions where extra = 'spam') tmp left join Removals r on tmp.post\_id = r.post\_id group by action\_date ) t;

**Q74.**

**Ans.**select round(t.player\_id/(select count(distinct player\_id) from activity),2) as fraction from ( select distinct player\_id, datediff(event\_date, lead(event\_date, 1) over(partition by player\_id order by event\_date)) as diff from activity ) t where diff = -1;

**Q75.**

**Ans.**select round(t.player\_id/(select count(distinct player\_id) from activity),2) as fraction from ( select distinct player\_id, datediff(event\_date, lead(event\_date, 1) over(partition by player\_id order by event\_date)) as diff from activity ) t where diff = -1;

**Q76.**

**Ans**.select company\_id, employee\_id, employee\_name, (case when max(salary) over(partition by company\_id) < 1000 then salary when max(salary) over(partition by company\_id) < 10000 then round(0.76\*salary) else round(0.51\*salary) end) as Salary from Salaries;

**Q77.**

**Ans**.select t.left\_operand, t.operator, t.right\_operand, (case when t.value > v2.value and operator = '>' then "true" when t.value < v2.value and operator = '<' then "true" when t.value = v2.value and operator = '=' then "true" else "false" end) as value from(select e.\*, v1.value from Expressions e inner join Variables v1 on e.left\_operand = v1.name) t inner join Variables v2 on t.right\_operand = v2.name;

**Q78.**

**Ans**.select t3.Name from ( select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg\_call\_duration, avg(t1.duration) over() as global\_average from ((select cl.caller\_id as id, cl.duration from Calls cl) union (select cl.callee\_id as id, cl.duration from Calls cl)) t1 left join (select p.id, c.Name from Person p left JOIN Country c ON cast(left(p.phone\_number,3) as int) = cast(c.country\_code as int)) t2 ON t1.id = t2.id) t3 where t3.avg\_call\_duration > global\_average group by t3.Name;

**Q79.**

**Ans**.select name from Employee order by name

**Q80.**

**Ans**.select year, product\_id, curr\_year\_spend, coalesce(prev\_year\_spend,'') as prev\_year\_spend, coalesce(round((curr\_year\_spend - prev\_year\_spend)/prev\_year\_spend \*100,2),'') as yoy\_rate from ( select year(transaction\_date) as year, product\_id, spend as curr\_year\_spend, round(lag(spend,1) over(partition by product\_id order by transaction\_date),2) as prev\_year\_spend from user\_transactions ) t;

**Q81.**

**Ans**.select item\_type, (case when item\_type = 'prime\_eligible' then floor(500000/sum(square\_footage)) \* count(item\_type) when item\_type = 'not\_prime' then floor((500000 -(select floor(500000/sum(square\_footage)) \* sum(square\_footage) from inventory where item\_type = 'prime\_eligible'))/sum(square\_footage)) \* count(item\_type) end) as item\_count from inventory group by item\_type order by count(item\_type) desc;

**Q82.**

**Ans**.select month(a.event\_date) as month, count(distinct a.user\_id) as monthly\_active\_users from user\_actions a inner join user\_actions b on concat(month(a.event\_date),year(a.event\_date)) = concat(1+month(b.event\_date),year(b.event\_date)) and a.user\_id = b.user\_id where a.event\_type in ('sign-in', 'like', 'comment') and b.event\_type in ('sign-in', 'like', 'comment') and concat(month(a.event\_date),'/',year(a.event\_date)) = '7/2022' and concat(1+month(b.event\_date),'/',year(b.event\_date)) = '7/2022' group by month(a.event\_date);

**Q83.**

**Ans.**with recursive seq as ( select searches, num\_users, 1 as c from search\_frequency union select searches, num\_users, c+1 from seq where c < num\_users ) select round(avg(t.searches),1) as median from (select searches,row\_number() over(order by searches, c) as r1, row\_number() over(order by searches desc, c desc) as r2 from seq order by searches) t where t.r1 in (t.r2, t.r2 - 1, t.r2 + 1);

**Q84.**

**Ans**.select user\_id, case when status in ('NEW','EXISTING','CHURN','RESURRECT') and user\_id not in (select user\_id from daily\_pay) then 'CHURN' when status in ('NEW','EXISTING','RESURRECT') and user\_id in (select user\_id from daily\_pay) then 'EXISTING' when status = 'CHURN' and user\_id in (select user\_id from daily\_pay) then 'RESURRECT' end as new\_status from advertiser order by user\_id;

**Q85**.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. Level - Hard Hint1. Calculate individual uptimes 2. Sum those up to obtain the uptime of the whole fleet, keeping in mind that the result must be output in units of full days

**Ans**.select sum(t.individual\_uptime) as total\_uptime\_days from ( select case when session\_status = 'stop' then timestampdiff(day, lag(status\_time) over(partition by server\_id order by status\_time), status\_time) end as individual\_uptime from server\_utilization ) t;

**Q86.**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. Level - Hard Hint- Use Partition and order by

**Ans**.select sum(case when (unix\_timestamp(t.next\_transaction) - unix\_timestamp(t.transaction\_timestamp))/60 <= 10 then 1 else 0 end) as payment\_count from (select transaction\_timestamp, lead(transaction\_timestamp,1) over(partition by merchant\_id, credit\_card\_id, Amount order by transaction\_timestamp) as next\_transaction from transactions)t;

**Q87.**DoorDash's Growth Team is trying to make sure new users (those who are making orders in their first 14 days) have a great experience on all their orders in their 2 weeks on the platform. Unfortunately, many deliveries are being messed up because: ● the orders are being completed incorrectly (missing items, wrong order, etc.) ● the orders aren't being received (wrong address, wrong drop off spot) ● the orders are being delivered late (the actual delivery time is 30 minutes later than when the order was placed). Note that the estimated\_delivery\_timestamp is automatically set to 30 minutes after the order\_timestamp.

**Ans**.select round(avg(t1.bad\_exp\_pct\_per\_cust),2) as bad\_exp\_pct from ( select t.customer\_id, 100\*sum(case when o.status <> 'completed successfully' then 1 else 0 end)/count(\*) as bad\_exp\_pct\_per\_cust from ( select customer\_id, signup\_timestamp from customers where month(signup\_timestamp) = 6 ) t inner join orders o on o.customer\_id = t.customer\_id where timestampdiff(day, t.signup\_timestamp, o.order\_timestamp) <= 13 group by t.customer\_id ) t1;

**Q88.**

**Ans**.select gender, day, sum(score\_points) over(partition by gender order by day) as total from Scores group by gender, day order by gender, day;

**Q89.**

**Ans**.select t3.Name from ( select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg\_call\_duration, avg(t1.duration) over() as global\_average from ((select cl.caller\_id as id, cl.duration from Calls cl) union (select cl.callee\_id as id, cl.duration from Calls cl)) t1 left join (select p.id, c.Name from Person p left JOIN Country c ON cast(left(p.phone\_number,3) as int) = cast(c.country\_code as int)) t2 ON t1.id = t2.id) t3 where t3.avg\_call\_duration > global\_average group by t3.Name;

**Q90**.

**Ans**.with recursive seq as ( select num, frequency, 1 as c from Numbers union select num, frequency, c+1 from seq where c < frequency ) select round(avg(t.num),1) as median from ( select num,row\_number() over(order by num, c) as r1, row\_number() over(order by num desc, c desc) as r2 from seq order by num ) t where t.r1 in (t.r2, t.r2 - 1,t.r2 + 1);

**Q91.**

**Ans**.select distinct concat(year(t.pay\_date),'-',month(t.pay\_date)) as pay\_month, t.department\_id, case when monthly\_department\_avg\_salary > monthly\_average\_salary then 'higher' when monthly\_department\_avg\_salary < monthly\_average\_salary then 'lower' else 'same' end as Comparison from (select s.pay\_date, e.department\_id, avg(s.amount) over(partition by month(s.pay\_date), e.department\_id) as monthly\_department\_avg\_salary, avg(s.amount) over(partition by month(s.pay\_date)) as monthly\_average\_salary from Salary s left join Employee e on s.employee\_id = e.employee\_id) t order by t.department\_id;

**Q92.**

**Ans**.select t1.install\_dt, count(player\_id) as installs, round(count(t1.next\_install)/count(t1.player\_id),2) as Day1\_retention from ( select t.player\_id, t.install\_dt, a.event\_date as next\_install from ( select player\_id, min(event\_date) as install\_dt from Activity group by player\_id ) t left join Activity a on t. player\_id = a.player\_id and a.event\_date = t.install\_dt + 1 ) t1 group by install\_dt;

**Q93.**

**Ans**.select t2.group\_id, t2.player\_id from ( select t1.group\_id, t1.player\_id, dense\_rank() over(partition by group\_id order by score desc, player\_id) as r from ( select p.\*, case when p.player\_id = m.first\_player then m.first\_score when p.player\_id = m.second\_player then m.second\_score end as score from Players p, Matches m where player\_id in (first\_player, second\_player) ) t1 ) t2 where r = 1;

**Q94.**

**Ans**.select t.student\_id, t.student\_name from (select s.student\_name, s.student\_id, count(e.student\_id) over(partition by student\_name) as exams\_given, case when e.score > min(e.score) over(partition by e.exam\_id) and e.score < max(e.score) over(partition by e.exam\_id) then 1 else 0 end as quiet # 1 means student is quiet, 0 means student is not quiet from Exam e left join Student s on e.student\_id = s.student\_id)t group by t.student\_name, t.student\_id, t.exams\_given having sum(t.quiet) = t.exams\_given

**Q95.**

**Ans.**select t.student\_id, t.student\_name from (select s.student\_name, s.student\_id, count(e.student\_id) over(partition by student\_name) as exams\_given, case when e.score > min(e.score) over(partition by e.exam\_id) and e.score < max(e.score) over(partition by e.exam\_id) then 1 else 0 end as quiet # 1 means student is quiet, 0 means student is not quiet from Exam e left join Student s on e.student\_id = s.student\_id)t group by t.student\_name, t.student\_id, t.exams\_given having sum(t.quiet) = t.exams\_given

**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.

**Ans.**select t.user\_id, t.song\_id, sum(t.song\_plays) as song\_plays from ( select user\_id, song\_id, song\_plays from songs\_history union all select user\_id, song\_id, 1 as song\_plays from songs\_weekly where date(listen\_time) <= '2022/08/04') t group by user\_id, song\_id;

**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.

**Ans**.select round(sum(case when t.signup\_action = 'Confirmed' then 1 else 0 end)/count(\*),2) as confirm\_rate from emails e join texts t on e.email\_id = t.email\_id;

**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 Important Assumptions: ● Rows in this table are consecutive and ordered by date. ● Each row represents a different day ● A day that does not correspond to a row in this table is not counted. The most recent day is the next row above the current row.

**Ans**.select user\_id, date\_format(tweet\_date, '%m/%d/%Y %h:%i:%s') as tweet\_date, round(avg(count(distinct tweet\_id)) over(order by tweet\_date rows between 2 preceding and current row),2) as rolling\_avg\_3days from tweets group by user\_id, tweet\_date

**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.

**Ans**.select b.age\_bucket, round(100\*sum(case when a.activity\_type = 'Send' then a.time\_spent else 0 end)/sum(a.time\_spent),2) send\_perc, round(100\*sum(case when a.activity\_type = 'Open' then a.time\_spent else 0 end)/sum(a.time\_spent),2) open\_perc from activities a join age\_breakdown b on a.user\_id = b.user\_id where activity\_type in ('Open', 'Send') group by b.age\_bucket order by b.age\_bucket;

**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. Level - Medium Hint- Use join and group by Write a query to return the IDs of these LinkedIn power creators in ascending order

**Ans**.select p.profile\_id from personal\_profiles p join employee\_company e on p.profile\_id = e.personal\_profile\_id join company\_pages c on e.company\_id = c.company\_id group by p.profile\_id, p.followers having p.followers > sum(c.followers) order by profile\_id;

**Q101.**

**Ans**.with new as (select t.username, t.activity, t.startDate, t.endDate from( select username, activity, startDate, endDate, dense\_rank() over(partition by username order by endDate desc) as r from UserActivity)t where r = 2 ) select \* from new union select n.username, n.activity, n.startDate, n.endDate from( select username, activity, startDate, endDate, dense\_rank() over(partition by username order by endDate desc) as r from UserActivity)n where r = 1 and username not in (select username from new);

**Q102.**

**Ans**.with new as (select t.username, t.activity, t.startDate, t.endDate from( select username, activity, startDate, endDate, dense\_rank() over(partition by username order by endDate desc) as r from UserActivity)t where r = 2 ) select \* from new union select n.username, n.activity, n.startDate, n.endDate from( select username, activity, startDate, endDate, dense\_rank() over(partition by username order by endDate desc) as r from UserActivity)n where r = 1 and username not in (select username from new);

**Q103.**

**Ans.**select name from Students where marks > 75 order by right(name, 3), id;

**Q104**.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.

**Ans**.select name from Employee where salary > 2000 and months < 10 order by employee\_id;

**Q105**.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.

**Ans**.select case when A+B > C and B+C > A and C+A > B then ( case when A != B and B != C then 'Scalene' when A = B and B = C then 'Equilateral' else 'Isosceles' end ) else 'Not A Triangle' end as Result from Triangles;

**Q106.**Samantha was tasked with calculating the average monthly salaries for all employees in the EMPLOYEES table, but did not realise her keyboard's 0 key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeros removed), and the actual average salary. Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries), and round it up to the next integer.

**Ans**.select ceil(avg(salary) - avg(replace(salary, 0, ''))) as calculation\_difference from Employees;

**Q107.**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.

**Ans**.select concat(max(t.earnings), ' ', sum(case when earnings = max\_salary then 1 else 0 end)) as Output from ( select max(salary\*months) over() as max\_salary, salary\*months as earnings from Employee) t;

**Q108**.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: Level - Medium There are a total of [occupation\_count] [occupation]s. 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.

**Ans**.select concat(name, '(', left(occupation,1),')') as name\_occupation) from Occupations order by name; select concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's.') as occupation\_count from Occupations group by occupation order by count(occupation), occupation;

**Q109**.Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

**Ans**.select max(case Occupation when 'Doctor' then Name end) as Doctors, max(case Occupation when 'Professor' then Name end) as Professors, max(case Occupation when 'Singer' then Name end) as Singers, max(case Occupation when 'Actor' then Name end) as Actors from ( select occupation, name, row\_number() over(partition by Occupation order by name) as r from Occupations ) t group by r;

**Q110.**

**Ans**.select ( case when P is NULL then 'Root' when N not in (select distinct P from BST where P is not null) then 'Leaf' else 'Inner' end ) as Node\_Type from BST order by N;

**Q111.**

**Ans.**select concat(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)) as Output from Company c left outer join Lead\_Manager l on c.company\_code = l.company\_code left join Senior\_Manager s on l.lead\_manager\_code = s.lead\_manager\_code left join Manager m on s.senior\_manager\_code = m.senior\_manager\_code left join Employee e on m.manager\_code = e.manager\_code group by c.company\_code, c.founder order by c.company\_code;

**Q112**.Write a query to print all prime numbers less than or equal to 1000. Print your result on a single line, and use the ampersand () character as your separator (instead of a space). For example, the output for all prime numbers <=10 would be: 2&3&5&7

**Ans.**with recursive cte as ( select 2 as num union select num+1 from cte where num+1 <= 1000 ) select GROUP\_CONCAT(num SEPARATOR "&") as prime from ( select 2 as num union select c1.num from cte c1 inner join cte c2 on c2.num <= round(c1.num/2) group by num having min(c1.num % c2.num) > 0 order by num )t;

**Q113.**

**Ans.**with recursive num(n) as ( select 1 union select n + 1 from num where n + 1 <= 20 ) select lpad('', num.n, '\*') as 'P(20)' from num;

**Q114.**

**Ans.**with recursive num(n) as ( select 20 union select n - 1 from num where n - 1 >= 1 ) select lpad('', num.n, '\*') as 'P(20)' from num;

**Q115.**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.

**Ans**.select name from Students where marks > 75 order by right(name, 3), id;

**Q116.**

**Ans**.select distinct a.X, a.Y from (select \*, row\_number() over(order by X) as r1 from Functions) a inner join (select \*,row\_number() over(order by X) as r2 from Functions) b on a.X = b.Y and b.X = a.Y where a.X <= a.Y and a.r1 <> b.r2 order by a.X

**Q117**.Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than $2000 per month who have been employees for less than 10 months. Sort your result by ascending employee\_id.

Ans.

**Q118.**

**Ans**.select case when A+B > C and B+C > A and C+A > B then ( case when A != B and B != C then 'Scalene' when A = B and B = C then 'Equilateral' else 'Isosceles' end ) else 'Not A Triangle' end as Result from Triangles;

**Q119**.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).

**Ans**.select year, product\_id, curr\_year\_spend, coalesce(prev\_year\_spend,'') as prev\_year\_spend, coalesce(round((curr\_year\_spend - prev\_year\_spend)/prev\_year\_spend \*100,2),'') as yoy\_rate from ( select year(transaction\_date) as year, product\_id, spend as curr\_year\_spend, round(lag(spend,1) over(partition by product\_id order by transaction\_date),2) as prev\_year\_spend from user\_transactions ) t;

**Q120**.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.

**Ans**.select item\_type, (case when item\_type = 'prime\_eligible' then floor(500000/sum(square\_footage)) \* count(item\_type) when item\_type = 'not\_prime' then floor((500000 -(select floor(500000/sum(square\_footage)) \* sum(square\_footage) from inventory where item\_type = 'prime\_eligible'))/sum(square\_footage)) \* count(item\_type) end) as item\_count from inventory group by item\_type order by count(item\_type) desc;

**Q121.**. 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.

**Ans**.For July Month

select month(a.event\_date) as month, count(distinct a.user\_id) as monthly\_active\_users from user\_actions a inner join user\_actions b on concat(month(a.event\_date),year(a.event\_date)) = concat(1+month(b.event\_date),year(b.event\_date)) and a.user\_id = b.user\_id where a.event\_type in ('sign-in', 'like', 'comment') and b.event\_type in ('sign-in', 'like', 'comment') and concat(month(a.event\_date),'/',year(a.event\_date)) = '7/2022' and concat(1+month(b.event\_date),'/',year(b.event\_date)) = '7/2022' group by month(a.event\_date);

**Q122**.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.

**Ans**.with recursive seq as ( select searches, num\_users, 1 as c from search\_frequency union select searches, num\_users, c+1 from seq where c < num\_users ) select round(avg(t.searches),1) as median from (select searches,row\_number() over(order by searches, c) as r1, row\_number() over(order by searches desc, c desc) as r2 from seq order by searches) t where t.r1 in (t.r2, t.r2 - 1, t.r2 + 1);

**Q123**.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.

**Ans**.select user\_id, case when status in ('NEW','EXISTING','CHURN','RESURRECT') and user\_id not in (select user\_id from daily\_pay) then 'CHURN' when status in ('NEW','EXISTING','RESURRECT') and user\_id in (select user\_id from daily\_pay) then 'EXISTING' when status = 'CHURN' and user\_id in (select user\_id from daily\_pay) then 'RESURRECT' end as new\_status from advertiser order by user\_id;

**Q124**.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.

**Ans**.select sum(t.individual\_uptime) as total\_uptime\_days from ( select case when session\_status = 'stop' then timestampdiff(day, lag(status\_time) over(partition by server\_id order by status\_time), status\_time) end as individual\_uptime from server\_utilization ) t;

**Q125**. 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.

**Ans**.select sum(case when (unix\_timestamp(t.next\_transaction) - unix\_timestamp(t.transaction\_timestamp))/60 <= 10 then 1 else 0 end) as payment\_count from (select transaction\_timestamp, lead(transaction\_timestamp,1) over(partition by merchant\_id, credit\_card\_id, Amount order by transaction\_timestamp) as next\_transaction from transactions)t;

**Q126**.DoorDash's Growth Team is trying to make sure new users (those who are making orders in their first 14 days) have a great experience on all their orders in their 2 weeks on the platform. Unfortunately, many deliveries are being messed up because: ● the orders are being completed incorrectly (missing items, wrong order, etc.) ● the orders aren't being received (wrong address, wrong drop off spot) ● the orders are being delivered late (the actual delivery time is 30 minutes later than when the order was placed). Note that the estimated\_delivery\_timestamp is automatically set to 30 minutes after the order\_timestamp.

**Ans**.select round(avg(t1.bad\_exp\_pct\_per\_cust),2) as bad\_exp\_pct from ( select t.customer\_id, 100\*sum(case when o.status <> 'completed successfully' then 1 else 0 end)/count(\*) as bad\_exp\_pct\_per\_cust from ( select customer\_id, signup\_timestamp from customers where month(signup\_timestamp) = 6 ) t inner join orders o on o.customer\_id = t.customer\_id where timestampdiff(day, t.signup\_timestamp, o.order\_timestamp) <= 13 group by t.customer\_id ) t1;

**Q127.**

**Ans.**select gender, day, sum(score\_points) over(partition by gender order by day) as total from Scores group by gender, day order by gender, day;

**Q128.**

**Ans.**select t3.Name from ( select t2.Name, avg(t1.duration) over(partition by t2.Name) as avg\_call\_duration, avg(t1.duration) over() as global\_average from ((select cl.caller\_id as id, cl.duration from Calls cl) union (select cl.callee\_id as id, cl.duration from Calls cl)) t1 left join (select p.id, c.Name from Person p left JOIN Country c ON cast(left(p.phone\_number,3) as int) = cast(c.country\_code as int)) t2 ON t1.id = t2.id) t3 where t3.avg\_call\_duration > global\_average group by t3.Name;

**Q129.**

**Ans**.with recursive seq as ( select num, frequency, 1 as c from Numbers union select num, frequency, c+1 from seq where c < frequency ) select round(avg(t.num),1) as median from ( select num,row\_number() over(order by num, c) as r1, row\_number() over(order by num desc, c desc) as r2 from seq order by num ) t where t.r1 in (t.r2, t.r2 - 1,t.r2 + 1);

**Q130.**

**Ans**.select distinct concat(year(t.pay\_date),'-',month(t.pay\_date)) as pay\_month, t.department\_id, case when monthly\_department\_avg\_salary > monthly\_average\_salary then 'higher' when monthly\_department\_avg\_salary < monthly\_average\_salary then 'lower' else 'same' end as Comparison from (select s.pay\_date, e.department\_id, avg(s.amount) over(partition by month(s.pay\_date), e.department\_id) as monthly\_department\_avg\_salary, avg(s.amount) over(partition by month(s.pay\_date)) as monthly\_average\_salary from Salary s left join Employee e on s.employee\_id = e.employee\_id) t order by t.department\_id;

**Q131.**

**Ans**.select t1.install\_dt, count(player\_id) as installs, round(count(t1.next\_install)/count(t1.player\_id),2) as Day1\_retention from ( select t.player\_id, t.install\_dt, a.event\_date as next\_install from ( select player\_id, min(event\_date) as install\_dt from Activity group by player\_id ) t left join Activity a on t. player\_id = a.player\_id and a.event\_date = t.install\_dt + 1 ) t1 group by install\_dt;

**Q132.**

**Ans**.select t2.group\_id, t2.player\_id from ( select t1.group\_id, t1.player\_id, dense\_rank() over(partition by group\_id order by score desc, player\_id) as r from ( select p.\*, case when p.player\_id = m.first\_player then m.first\_score when p.player\_id = m.second\_player then m.second\_score end as score from Players p, Matches m where player\_id in (first\_player, second\_player) ) t1 ) t2 where r = 1;

**Q133.**

**Ans.**select t.student\_id, t.student\_name from (select s.student\_name, s.student\_id, count(e.student\_id) over(partition by student\_name) as exams\_given, case when e.score > min(e.score) over(partition by e.exam\_id) and e.score < max(e.score) over(partition by e.exam\_id) then 1 else 0 end as quiet # 1 means student is quiet, 0 means student is not quiet from Exam e left join Student s on e.student\_id = s.student\_id)t group by t.student\_name, t.student\_id, t.exams\_given having sum(t.quiet) = t.exams\_given

**Q134.**

**Ans.**select s.student\_name, s.student\_id, count(e.student\_id) over(partition by student\_name) as exams\_given, case when e.score > min(e.score) over(partition by e.exam\_id) and e.score < max(e.score) over(partition by e.exam\_id) then 1 else 0 end as quiet # 1 means student is quiet, 0 means student is not quiet from Exam e left join Student s on e.student\_id = s.student\_id)t group by t.student\_name, t.student\_id, t.exams\_given having sum(t.quiet) = t.exams\_given

**Q135.**

**Ans.**with new as (select t.username, t.activity, t.startDate, t.endDate from( select username, activity, startDate, endDate, dense\_rank() over(partition by username order by endDate desc) as r from UserActivity)t where r = 2 ) select \* from new union select n.username, n.activity, n.startDate, n.endDate from( select username, activity, startDate, endDate, dense\_rank() over(partition by username order by endDate desc) as r from UserActivity)n where r = 1 and username not in (select username from new);

**Q136.**

**Ans**.with new as (select t.username, t.activity, t.startDate, t.endDate from( select username, activity, startDate, endDate, dense\_rank() over(partition by username order by endDate desc) as r from UserActivity)t where r = 2 ) select \* from new union select n.username, n.activity, n.startDate, n.endDate from( select username, activity, startDate, endDate, dense\_rank() over(partition by username order by endDate desc) as r from UserActivity)n where r = 1 and username not in (select username from new);

**Q137.**

**Ans**.select ceil(avg(salary) - avg(replace(salary, 0, ''))) as calculation\_difference from Employees;

**Q138**.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.

**Ans**.select concat(max(t.earnings), ' ', sum(case when earnings = max\_salary then 1 else 0 end)) as Output from ( select max(salary\*months) over() as max\_salary, salary\*months as earnings from Employee) t;

**Q139**.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: Level - Medium There are a total of [occupation\_count] [occupation]s. 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.

**Ans**.select concat(name, '(', left(occupation,1),')') as name\_occupation) from Occupations order by name; select concat('There are a total of', ' ', count(occupation), ' ', lower(occupation), 's.') as occupation\_count from Occupations group by occupation order by count(occupation), occupation;

**Q140.**Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

**Ans**.select max(case Occupation when 'Doctor' then Name end) as Doctors, max(case Occupation when 'Professor' then Name end) as Professors, max(case Occupation when 'Singer' then Name end) as Singers, max(case Occupation when 'Actor' then Name end) as Actors from ( select occupation, name, row\_number() over(partition by Occupation order by name) as r from Occupations ) t group by r;

**Q141.**

Ans.select ( case when P is NULL then 'Root' when N not in (select distinct P from BST where P is not null) then 'Leaf' else 'Inner' end ) as Node\_Type from BST order by N;

**Q142**.

**Ans**.select concat(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)) as Output from Company c left outer join Lead\_Manager l on c.company\_code = l.company\_code left join Senior\_Manager s on l.lead\_manager\_code = s.lead\_manager\_code left join Manager m on s.senior\_manager\_code = m.senior\_manager\_code left join Employee e on m.manager\_code = e.manager\_code group by c.company\_code, c.founder order by c.company\_code;

**Q144.**

**Ans**.select distinct a.X, a.Y from (select \*, row\_number() over(order by X) as r1 from Functions) a inner join (select \*,row\_number() over(order by X) as r2 from Functions) b on a.X = b.Y and b.X = a.Y where a.X <= a.Y and a.r1 <> b.r2 order by a.X

**Q145**.Julia just finished conducting a coding contest, and she needs your help assembling the leaderboard! Write a query to print the respective hacker\_id and name of hackers who achieved full scores for more than one challenge. Order your output in descending order by the total number of challenges in which the hacker earned a full score. If more than one hacker received full scores in the same number of challenges, then sort them by ascending hacker\_id.

**Ans**.select concat(t1.hacker\_id, ' ', t1.name) as Result from ( select t.hacker\_id, t.name, dense\_rank() over(order by full\_score\_challenge\_count desc) as r from ( select h.hacker\_id, h.name, count(h.hacker\_id) as full\_score\_challenge\_count from Submissions s join Hackers h on s.hacker\_id = h.hacker\_id join Challenges c on s.challenge\_id = c.challenge\_id join Difficulty d on d.difficulty\_level = c.difficulty\_level where s.score = d.score group by h.hacker\_id, h.name having full\_score\_challenge\_count > 1 ) t ) t1 where t1.r = 1 order by t1.hacker\_id;

**Q146.**

**Ans**.select s.start\_date, min(e.end\_date) as end\_date, (min(e.end\_date) - s.start\_date) as number\_of\_days from (select start\_date from Projects where start\_date - 1 not in (select start\_date from Projects)) s, (select end\_date from Projects where end\_date + 1 not in (select end\_date from Projects)) e where s.start\_date <= e.end\_date group by s.start\_date;

**Q147**.In an effort to identify high-value customers, Amazon asked for your help to obtain data about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more consecutive days. List the user IDs who have gone on at least 1 shopping spree in ascending order.

**Ans**.select distinct t.user\_id from ( select user\_id, transaction\_date as first, lead(transaction\_date,1) over(partition by user\_id order by transaction\_date) as second, lead(transaction\_date,2) over(partition by user\_id order by transaction\_date) as third from transactions ) t where timestampdiff(day, first, second) = 1 and timestampdiff(day, second, third) = 1;

**Q148**.You are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A two-way unique relationship is established when two people send money back and forth. Write a query to find the number of two-way unique relationships in this data

**Ans**.select count(\*) as unique\_relationshis from (select count(\*) as relation\_count from ( select greatest(payer\_id, recipient\_id) as person1, least(payer\_id, recipient\_id) as person2 from (select distinct \* from payments) t ) t1 group by person1, person2 ) t2 where relation\_count = 2;

**Q149.**Assume you are given the table below on user transactions. Write a query to obtain the list of customers whose first transaction was valued at $50 or more. Output the number of users. Clarification: ● Use the transaction\_date field to determine which transaction should be labeled as the first for each user. ● Use a specific function (we can't give too much away!) to account for scenarios where a user had multiple transactions on the same day, and one of those was the first.

**Ans.**select count(\*) as users from ( select transaction\_id, user\_id, spend, row\_number() over(partition by user\_id order by transaction\_date) as r from user\_transactions ) t where t.r =1 and t.spend >= 50;

**Q150**.Assume you are given the table below containing measurement values obtained from a sensor over several days. Measurements are taken several times within a given day. Write a query to obtain the sum of the odd-numbered and even-numbered measurements on a particular day, in two different columns. Note that the 1st, 3rd, 5th measurements within a day are considered odd-numbered measurements and the 2nd, 4th, 6th measurements are even-numbered measurements.

**Ans**.select measurement\_day, round(sum(case when r % 2 != 0 then measurement\_value else 0 end),2) as odd\_sum, round(sum(case when r % 2 = 0 then measurement\_value else 0 end),2) as even\_sum from ( select date\_format(measurement\_time, '%m/%d/%Y 00:00:00') as measurement\_day, measurement\_value, row\_number() over(partition by date(measurement\_time) order by measurement\_time) as r from measurements )t group by measurement\_day;

**Q151**.In an effort to identify high-value customers, Amazon asked for your help to obtain data about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more consecutive days. List the user IDs who have gone on at least 1 shopping spree in ascending order.

**Ans.**select distinct t.user\_id from ( select user\_id, transaction\_date as first, lead(transaction\_date,1) over(partition by user\_id order by transaction\_date) as second, lead(transaction\_date,2) over(partition by user\_id order by transaction\_date) as third from transactions ) t where timestampdiff(day, first, second) = 1 and timestampdiff(day, second, third) = 1;

**Q152.**The Airbnb Booking Recommendations team is trying to understand the "substitutability" of two rentals and whether one rental is a good substitute for another. They want you to write a query to find the unique combination of two Airbnb rentals with the same exact amenities offered. Output the count of the unique combination of Airbnb rentals

**Ans.**select count(t1.amenity\_count) as matching\_airbnb from ( select t.amenities, count(\*) as amenity\_count from ( select rental\_id, group\_concat(amenity order by amenity) amenities from rental\_amenities group by rental\_id )t group by t.amenities )t1 where t1.amenity\_count>1;

**Q153.**Google marketing managers are analysing the performance of various advertising accounts over the last month. They need your help to gather the relevant data. Write a query to calculate the return on ad spend (ROAS) for each advertiser across all ad campaigns. Round your answer to 2 decimal places, and order your output by the advertiser\_id.

**Ans.**select advertiser\_id, sum(revenue)/sum(spend) as ROAS from ad\_campaigns group by advertiser\_id order by advertiser\_id;

**Q154.**Your team at Accenture is helping a Fortune 500 client revamp their compensation and benefits program. The first step in this analysis is to manually review employees who are potentially overpaid or underpaid. An employee is considered to be potentially overpaid if they earn more than 2 times the average salary for people with the same title. Similarly, an employee might be underpaid if they earn less than half of the average for their title. We'll refer to employees who are both underpaid and overpaid as compensation outliers for the purposes of this problem. Write a query that shows the following data for each compensation outlier: employee ID, salary, and whether they are potentially overpaid or potentially underpaid (refer to Example Output below)

**Ans.**select t.employee\_id, t.salary, case when t.salary > t.base\_for\_overpaid then 'Overpaid' when t.salary < t.base\_for\_underpaid then 'Underpaid' end as status from (select employee\_id, salary, 2\*avg(salary) over(partition by title) as base\_for\_overpaid, 0.5\*avg(salary) over(partition by title) as base\_for\_underpaid from employee\_pay )t having status is not null order by t.employee\_id;

**Q155.Y**ou are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A two-way unique relationship is established when two people send money back and forth. Write a query to find the number of two-way unique relationships in this data. Assumption: ● A payer can send money to the same recipient multiple times.

**Ans.**select count(\*) as unique\_relationshis from (select count(\*) as relation\_count from ( select greatest(payer\_id, recipient\_id) as person1, least(payer\_id, recipient\_id) as person2 from (select distinct \* from payments) t ) t1 group by person1, person2 ) t2 where relation\_count = 2;

**Q156.**Assume you are given the table below containing information on user purchases. Write a query to obtain the number of users who purchased the same product on two or more different days. Output the number of unique users. PS. On 26 Oct 2022, we expanded the purchases data set, thus the oficial output may vary from before.

**Ans.**select count(distinct t.user\_id) as repeat\_purchasers from ( select user\_id, product\_id, count(\*) as c from purchases group by user\_id, product\_id having c > 1 ) t;

**Q157.**Say you have access to all the transactions for a given merchant account. Write a query to print the cumulative balance of the merchant account at the end of each day, with the total balance reset back to zero at the end of the month. Output the transaction date and cumulative balance. Hint-You should use CASE.

**Ans.**select distinct DATE\_FORMAT(transaction\_date, '%m/%d/%Y 12:00:00'), round(sum(amount) over(partition by month(transaction\_date) order by transaction\_date),2) as balance from ( select transaction\_date, case when type = 'deposit' then amount else -amount end as amount from transactions ) t;

**Q158.**Assume you are given the table below containing information on Amazon customers and their spend on products belonging to various categories. Identify the top two highest-grossing products within each category in 2022. Output the category, product, and total spend.

**Ans.**select t.category, t.product, t.total\_spend from ( select category, product, round(sum(spend),2) as total\_spend, dense\_rank() over(partition by category order by sum(spend) desc) as r from product\_spend group by category, product ) t where r <= 2

**Q159**.Facebook is analysing its user signup data for June 2022. Write a query to generate the churn rate by week in June 2022. Output the week number (1, 2, 3, 4, ...) and the corresponding churn rate rounded to 2 decimal places. For example, week number 1 represents the dates from 30 May to 5 Jun, and week 2 is from 6 Jun to 12 Jun

**Ans.**according to week of year select week(signup\_date), round(100\*sum(case when timestampdiff(day,signup\_date,last\_login) <= 28 then 1 else 0 end)/count(\*),2) as churn\_rate from users group by week(signup\_date); .