Hacker Rank:

**1 Query:**

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.

**Solution:**

SELECT N,

CASE

WHEN P IS NULL THEN 'Root'

WHEN N IN (SELECT distinct P FROM BST) THEN 'Inner'

ELSE 'Leaf' END AS node\_type

FROM BST

ORDER BY N;

Explanation:

For any node if the parent is null then by default it is ‘Root’

For any node if the distinct value is present in the parent column thenit is inner

Else leaf

**2 Query:**

Amber's conglomerate corporation just acquired some new companies. Each of the companies follows this

hierarchy: 

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*.

**Note:**

The tables may contain duplicate records.

The *company\_code* is string, so the sorting should not be **numeric**. For example, if the *company\_codes* are *C\_1*, *C\_2*, and *C\_10*, then the ascending *company\_codes* will be *C\_1*, *C\_10*, and *C\_2*.

**Solution:**

select c.company\_code, founder,

count(distinct lead\_manager\_code),

count(distinct senior\_manager\_code),

count(distinct manager\_code),

count(distinct employee\_code)

from Company c

join Employee e on c.company\_code = e.company\_code

group by c.company\_code, founder

order by c.company\_code

Explanation:

All the details are available in the employee table

We need company\_code, founder\_name and the counts

We need to join the two tables based on company code and get the count of distinct attributes of the emp table.

3 Query:

Samantha was tasked with calculating the average monthly salaries for all employees in the EMPLOYEES table, but did not realize 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.:  average monthly salaries), and round it up to the next integer.

Solution:

SELECT CEILING(Salarya - Salarym)

FROM (

SELECT AVG(Salary) AS "Salarya", AVG(REPLACE(Salary, 0, '')) AS "Salarym"

FROM EMPLOYEES

) t;

Explanation:

Basically Samantha did not enter the 0’s

First calculate the avg salaries

Now replace the 0’s with ‘’ and calculate the average again.

Since we need to round it up to the next integer.

Use CEILING and get the difference.

4 Query:

We define an employee's *total earnings* to be their monthly  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  space-separated integers.

Solution:

SELECT CEILING(Salarya - Salarym)

FROM (

SELECT AVG(Salary) AS "Salarya", AVG(REPLACE(Salary, 0, '')) AS "Salarym"

FROM EMPLOYEES

) t;

Explanation:

Basically Samantha did not enter the 0’s

First calculate the avg salaries

Now replace the 0’s with ‘’ and calculate the average again.

Since we need to round it up to the next integer.

Use CEILING and get the difference.

5 Query:

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

Solution:

SELECT

MAX(salary \* months) AS max\_earnings, COUNT(\*) AS num\_max\_earners

FROM Employee

WHERE salary \* months = (SELECT MAX(salary \* months) FROM Employee);

Explanation:

The MAX function is used to find the maximum total earnings for all employees in the Employee table.

The total earnings for each employee is calculated by multiplying their monthly\_salary by months\_worked.

The COUNT function is used to count the number of employees who have the maximum total earnings.

The WHERE clause filters the results to include only those employees whose total earnings match the maximum total earnings.

6 Query:

The sum of all values in *LAT\_N* rounded to a scale of  decimal places.

The sum of all values in *LONG\_W* rounded to a scale of  decimal places

**Solution**

select round(sum(LAT\_N),2) as lat, round(sum(LONG\_W),2)as lon from STATION;

7 Query:

Query the sum of *Northern Latitudes* (*LAT\_N*) from **STATION** having values greater than 38.7880 and less than 137.2345. Truncate your answer to 4 decimal places.

**Solution:**

SELECT ROUND(SUM(LAT\_N),4) FROM STATION where LAT\_N between 38.7880 and 137.2345 ;

8 Query:

Query the greatest value of the *Northern Latitudes* (*LAT\_N*) from **STATION** that is less than <137.2345. Truncate your answer 4 to  decimal places.

**Solution:**

SELECT TRUNCATE(LAT\_N,4) FROM STATION WHERE LAT\_N<137.2345 ORDER BY LAT\_N DESC LIMIT 1;

9 Query:

Query the *Western Longitude* (*LONG\_W*) for the largest *Northern Latitude* (*LAT\_N*) in **STATION** that is less than 137.2345. Round your answer to 4 decimal places.

**Solution**

**SELECT ROUND(LONG\_W,4) FROM STATION WHERE LAT\_N = (SELECT MAX(LAT\_N) FROM STATION WHERE LAT\_N <137.2345)**

10 Query:

Query the smallest *Northern Latitude* (*LAT\_N*) from **STATION** that is greater than 38.7880. Round your answer to 4 decimal places.

**Solution**

**SELECT ROUND(LAT\_N,4) FROM STATION WHERE LAT\_N < 38.7880 order by LAT\_N asc limit 1;**

11 Query:

Query the *Western Longitude* (*LONG\_W*)where the smallest *Northern Latitude* (*LAT\_N*) in **STATION** is greater than 38.7880. Round your answer to  decimal places.

**Solution**

**SELECT ROUND(LONG\_W,4) FROM STATION WHERE LAT\_N = (SELECT min(LAT\_N) FROM STATION WHERE LAT\_N > 38.7880 )**

11 Query:

*A*[median](https://en.wikipedia.org/wiki/Median)*is defined as a number separating the higher half of a data set from the lower half. Query the*median*of the*Northern Latitudes*(*LAT\_N*) from****STATION****and round your answer to  decimal places.*

**Solution**

**SELECT ROUND(AVG(LAT\_N), 4) AS MEDIAN\_LAT\_N**

**FROM (**

**SELECT LAT\_N, ROW\_NUMBER() OVER (ORDER BY LAT\_N) AS RowNum, COUNT(\*) OVER () AS TotalRows FROM STATION ) t**

**WHERE RowNum IN ((TotalRows + 1) / 2, (TotalRows + 2) / 2);**

Explanation:

ROW\_NUMBER() OVER (ORDER BY LAT\_N) AS RowNum: assigns a row number to each record in the STATION table, based on the order of the LAT\_N column.

COUNT(\*) OVER () AS TotalRows: calculates the total number of rows in the STATION table.

SELECT ROUND(AVG(LAT\_N), 4) AS MEDIAN\_LAT\_N: calculates the average of the LAT\_N values and rounds the result to 4 decimal places.

FROM (subquery) t: specifies the subquery as the source table for the outer query, and assigns it an alias t.

WHERE RowNum IN ((TotalRows + 1) / 2, (TotalRows + 2) / 2): filters the rows where the row number is equal to the median row number(s).

12 Query

Ketty*gives*Eve*a task to generate a report containing three columns:*Name*,*Grade*and*Mark*.*Ketty*doesn't want the NAMES of those students who received a grade lower than*8*. The report must be in descending order by grade -- i.e. higher grades are entered first. If there is more than one student with the same grade (8-10) assigned to them, order those particular students by their name alphabetically. Finally, if the grade is lower than 8, use "NULL" as their name and list them by their grades in descending order. If there is more than one student with the same grade (1-7) assigned to them, order those particular students by their marks in ascending order*.

**Solution**

**SELECT CASE WHEN G.GRADE>=8 THEN S.NAME ELSE 'NULL' END,**

**G.GRADE, S.MARKS**

**FROM STUDENTS S, GRADES G**

**WHERE S.MARKS<=MAX\_MARK AND S.MARKS>=MIN\_MARK**

**ORDER BY G.GRADE DESC, S.NAME, S.MARKS**

Explanation:

CASE WHEN G.Grade < 8 THEN NULL ELSE s.Name END: This creates a new column called "Name" that will be set to "NULL" if the student's grade is less than 8. Otherwise, it will be set to the student's actual name where Grades as g ON s.marks >= min\_mark AND s.marks <= max\_mark: This joins the Students and Grades tables based on the range of marks. The clause checks if the Marks value for each student falls within the range defined by the min\_mark and max\_mark values for each grade.

ORDER BY g.Grade DESC,s.Name

13 Query

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 same number of challenges, then sort them by ascending *hacker\_id*.. Table

Description automatically generated Table

Description automatically generated Table

Description automatically generatedTable

Description automatically generated

**Solution**

**select h.hacker\_id , h.name from Hackers h**

**join Submissions s on h.hacker\_id = s.hacker\_id**

**join Challenges c on s.challenge\_id = c.challenge\_id**

**join Difficulty d on c.difficulty\_level = d.difficulty\_level**

**where s.score = d.score**

**group by h.hacker\_id , h.name**

**having count(\*)> 1**

**order by count(\*) desc, h.hacker\_id**

13 Query

Harry Potter and his friends are at Ollivander's with Ron, finally replacing Charlie's old broken wand.

Hermione decides the best way to choose is by determining the minimum number of gold galleons needed to buy each *non-evil* wand of high power and age. Write a query to print the *id*, *age*, *coins\_needed*, and *power* of the wands that Ron's interested in, sorted in order of descending *power*. If more than one wand has same power, sort the result in order of descending *age*. Table

Description automatically generated Table

Description automatically generated

**Solution**

**select w.id as id, wp.age as age , w.coins\_needed as coins\_needed, w.power as Power\_**

**from Wands w**

**join Wands\_Property wp on w.code = wp.code**

**where wp.is\_evil = 0 AND**

**w.coins\_needed = (SELECT MIN(coins\_needed) FROM wands WHERE code = w.code AND power = w.power)**

**order by w.power desc, wp.age desc;**

13 Query

Julia asked her students to create some coding challenges. Write a query to print the *hacker\_id*, *name*, and the total number of challenges created by each student. Sort your results by the total number of challenges in descending order. If more than one student created the same number of challenges, then sort the result by *hacker\_id*. If more than one student created the same number of challenges and the count is less than the maximum number of challenges created, then exclude those students from the result.

**Solution**

**WITH T1 AS(**

**SELECT h.hacker\_id AS hacker\_id, name, COUNT(challenge\_id) AS challenges\_created FROM Hackers h JOIN Challenges c ON h.hacker\_id = c.hacker\_id GROUP BY h.hacker\_id, name**

**),**

**T2 AS(**

**SELECT challenges\_created, COUNT(challenges\_created) AS c FROM T1 GROUP BY challenges\_created**

**)**

**SELECT T1.hacker\_id, name, T1.challenges\_created FROM T1 JOIN T2 ON T1.challenges\_created = T2.challenges\_created WHERE c=1 OR T1.challenges\_created=(SELECT MAX(challenges\_created) FROM T1) ORDER BY T1.challenges\_created DESC, T1.hacker\_id;**

14 Query

The total score of a hacker is the sum of their maximum scores for all of the challenges. Write a query to print the *hacker\_id*, *name*, and total score of the hackers ordered by the descending score. If more than one hacker achieved the same total score, then sort the result by ascending *hacker\_id*. Exclude all hackers with a total score of  from your result.

**Solution**

**select h.hacker\_id, h.name, SUM(max\_score) as total\_score**

**from (select hacker\_id, max(score)as max\_score from submissions group by challenge\_id, hacker\_id ) as s**

**join hackers h on s.hacker\_id = h.hacker\_id**

**group by h.hacker\_id, name**

**having sum(max\_score) > 0**

**order by total\_score desc, hacker\_id asc**

15 Query

If the *End\_Date* of the tasks are consecutive, then they are part of the same project. Samantha is interested in finding the total number of different projects completed.

Write a query to output the start and end dates of projects listed by the number of days it took to complete the project in ascending order. If there is more than one project that have the same number of completion days, then order by the start date of the project.

**Solution**

**select start\_date, end\_date**

**from (select start\_date, row\_number() over() as rn from projects where start\_date not in (select end\_date from projects)) a**

**join (select end\_date, row\_number() over() as rn from projects where end\_date not in (select start\_date from projects)) b**

**on a.rn = b.rn**

**order by datediff(start\_date, end\_date) desc, start\_date asc**