SQL Module Test

Tree_Node

Problem Description :

Write a query to report the type of each node in the tree.

- **Leaf** if the node is a leaf node.
- **Root** if the node is the root node of the tree.
- **Inner** if the node is neither a leaf node nor the root node.

Note:

1. Save the type of the node column as "Type".
2. Return the output in the ascending order of the Id column.

- Return the columns **Id** and **Type**.

Sample Input:

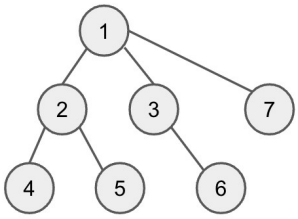

Table: tree

Id	p_id
1	NULL
2	1
3	1
4	2
5	2
6	3
7	1

Sample Output:

Id	Type
1	Root
2	Inner
3	Inner
4	Leaf
5	Leaf
6	Leaf
7	Leaf

Explanation:

The column **p_id(parent id)** indicates whether the node is either Root, Leaf, or Inner.
For Id = **1** the p_id is null which indicates that it is a **root** node.
Similarly, **2** and **3 Ids** have the p_id as **1** which indicates these two are **inner** nodes. **4**, **5**, and **6, 7** do not contain their ids in p_id which means these are **leaf** nodes.

```
select id,
case
when p_id is null then 'Root'
when id not in ( select distinct p_id from tree where p_id is not null ) then 'Leaf'
else 'Inner'
end as Type
from tree
order by 1;
```

Product Sales Analysis II

Problem Statement:

Write a query that reports the total quantity sold for every product id.

- Return the resulting table ordered by product_id in ascending order.

Sample Input:

Table: sales

sale_id	product_id	year	quantity	price
1	100	2008	10	5000
2	100	2009	12	5000
7	200	2011	15	9000

Table: product

product_id	product_name
100	Nokia
200	Apple
300	Samsung

Sample output:

product_id	total_quantity
100	22
200	15

Explanation:

- Total quantity of the product having product_id = 100 is 10+12 = 22.
- Total quantity of the product having product_id = 200 is 15.

```
with cte_total_quantity as
(
select product_id, sum(quantity) as total_quantity from sales
group by product_id
)
select P.product_id, coalesce(Q.total_quantity,0) as total_quantity
```

```
from product P
left join cte_total_quantity Q on P.product_id = Q.product_id
order by 1;
```

Popularity Percentage

Problem Statement:

In the given dataset, the ‘user1’ and ‘user2’ columns represent pairs of friends who use Facebook.

Write a query to find the **popularity percentage** of each of these users.

- Output each user along with their popularity percentage.
- Return the popularity percentage rounded up to **three** decimal places.
- Return the result ordered by the **popularity_percent** in **descending** order and by **user1** in **ascending** order.

Sample Input:

Table: **facebook**

user1	user2
2	1
1	3
4	1
1	5
1	6
2	6
7	2
8	3
3	9

Note: There are no cases of similar pairs like **(1,2)** and **(2,1)**.

Sample Output:

user1	popularity_percent
1	55.556
2	33.333
3	33.333
6	22.222
4	11.111
7	11.111

Sample Explanation:

The popularity percentage is defined as the total number of friends the user has divided by the total number of users on the platform, then converted into a percentage by multiplying by 100.

```
with cts_users as
(
select user1 from facebook
union
select user2 from facebook
),
cts_user1_count as
(
select user1, count(*) as count
from facebook
group by 1
),
cts_user2_count as
(
select user2, count(*) as count
from facebook
group by 1
),
cts_friend_count as
(
select U.user1,(coalesce(U1.count,0) + coalesce(U2.count,0)) as friend_count
from cts_users U
left join cts_user1_count U1 on U.user1 = U1.user1
left join cts_user2_count U2 on U.user1 = U2.user2
)
select user1, round(100*(friend_count/(Select count(1) from cts_friend_count)),3) as popularity_percent
from cts_friend_count
order by 2 desc, 1;
```

Employee Salary II

Problem Statement:

Given the following **employee** table,

Write a query to find the employees who have a salary **greater than** their manager's salary.

- Return the result ordered by **emp_name** in descending order.

Sample Input:

Table: **employee**

emp_id	emp_name	manager_id	department	salary
1	jaime	0	IT	85000
2	robert	1	IT	75000
3	lisa	1	IT	65000
4	chris	1	IT	55000
5	mary	7	SALES	55000
6	richard	7	SALES	85000
7	jane	0	SALES	80000
8	trevor	7	SALES	65000
9	joan	12	HR	55000
10	jennifer	12	HR	71000
11	trish	12	HR	58000
12	marge	0	HR	70000

Sample Output:

emp_name	department	employee_salary	manager_salary
richard	SALES	85000	80000
jennifer	HR	71000	70000

select E.emp_name, E.department, E.salary as employee_salary, M.salary as manager_salary
from employee E
join employee M on E.manager_id = M.emp_id
where E.salary > M.salary
order by 1 DESC
;

Total Spend per Customer

Problem Statement: Given the following tables,

Write a query to find out what is the total amount that each customer spent at the restaurant.

Sample Input:

Table: **sales**

customer_id	order_date	product_id
A	2021-01-01	1
A	2021-01-01	2
A	2021-01-07	2
A	2021-01-10	3
A	2021-01-11	3
A	2021-01-11	3
B	2021-01-01	2
B	2021-01-02	2
B	2021-01-04	1
B	2021-01-11	1
B	2021-01-16	3
B	2021-02-01	3
C	2021-01-01	3
C	2021-01-01	3
C	2021-01-07	3

Table: **menu**

product_id	product_name	price
1	sushi	10
2	curry	15
3	ramen	12

Note: Sort the result set in **decreasing** order of the total amount spent (i.e., total_spent) by each customer at the restaurant.

Sample Output:

c_id	total_spent
A	76
B	74
C	36

```
select
S.customer_id as c_id, sum(price) as total_spent
from sales S
join menu M on S.product_id = M.product_id
group by 1
order by 2 DESC;
```

Customers from North

Consider two large tables, Customers (columns: CustomerID, Name, RegionID) and Orders (columns: OrderID, CustomerID, OrderDate, TotalAmount). You want to find all customers from the “North” region who have not placed any orders. **Which of the following queries is the most efficient?**

A)

```
SELECT c.CustomerID, c.Name
FROM Customers c
LEFT JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE c.RegionID = 'North' AND o.OrderID IS NULL;
```

B)

```
SELECT c.CustomerID, c.Name
FROM Customers c
WHERE c.RegionID = 'North' AND NOT EXISTS (
 SELECT 1
 FROM Orders o
 WHERE c.CustomerID = o.CustomerID
);
```

C)

```
SELECT c.CustomerID, c.Name
FROM Customers c
WHERE c.RegionID = 'North' AND c.CustomerID NOT IN (
 SELECT CustomerID
 FROM Orders
);
```

D)

```
SELECT CustomerID, Name
FROM Customers
WHERE RegionID = 'North' AND CustomerID NOT IN (
 SELECT DISTINCT CustomerID
 FROM Orders
);
```

Option B

Which Average

Consider a table Products with columns ProductID, ProductName, BrandID, and Price, and another table Brands with columns BrandID and BrandName. Review the following query:

```
SELECT BrandName
FROM Brands
WHERE BrandID IN (
 SELECT BrandID
 FROM Products
 WHERE Price > (
 SELECT AVG(Price)
 FROM Products
 )
);
```

What does this query return?

The names of brands that have at least one product priced above the average price of all products.

Claims and fraud

Consider a dataset from ABC Corp which is a mid-sized insurer in the US with columns: policy_num, state, claim_cost, fraud_score.

Review the following Query:

```
select policy_num,
state,
claim_cost,
fraud_score
from

(select policy_num,
state,
claim_cost,
fraud_score,
ntile(100) over(partition by state order by fraud_score desc) as fraud_tile
from fraud_score) x

where fraud_tile <= 5
```

What does this query return?

This query returns the policy number, state, claim cost, and fraud score for the top 5% of claims with the highest fraud score within each state.

Employees and Departments

Consider two tables employees and departments

Review this Query:

```sql
SELECT
    e.employee_id,
    e.employee_name,
    d.department_name
FROM
    employees e
JOIN
    departments d ON e.department_id = d.department_id
WHERE
    EXISTS (
        SELECT 1
        FROM
            employees
        WHERE
            department_id = e.department_id
        GROUP BY
            department_id
        HAVING
            COUNT(*) > 10
    )
AND
    e.salary = (
        SELECT
            MAX(salary)
        FROM
            employees
        WHERE
            department_id = e.department_id
    );
```

What would this query return?

Employees who work in departments with more than 10 employees and have the highest salary in their department.

Facebook Session

Calculate each user's average session time. A session is defined as the time difference between a page_load and page_exit. For simplicity, assume a user has only 1 session per day and if there are multiple of the same events on that day, consider only the

latest page_load and earliest page_exit, with an obvious restriction that load time event should happen before exit time event . Output the user_id and their average session time.

Sample data

Table: facebook_session

user_id	timestamp	action
0	2019-04-25 13:30:15	page_load
0	2019-04-25 13:30:18	page_load
0	2019-04-25 13:30:40	scroll_down
0	2019-04-25 13:30:45	scroll_up
0	2019-04-25 13:31:10	scroll_down
0	2019-04-25 13:31:25	scroll_down
0	2019-04-25 13:31:40	page_exit
1	2019-04-25 13:40:00	page_load
1	2019-04-25 13:40:10	scroll_down
1	2019-04-25 13:40:15	scroll_down
1	2019-04-25 13:40:20	scroll_down
1	2019-04-25 13:40:25	scroll_down
1	2019-04-25 13:40:30	scroll_down
1	2019-04-25 13:40:35	page_exit
2	2019-04-25 13:41:21	page_load
2	2019-04-25 13:41:30	scroll_down
2	2019-04-25 13:41:35	scroll_down
2	2019-04-25 13:41:40	scroll_up
1	2019-04-26 11:15:00	page_load
1	2019-04-26 11:15:10	scroll_down
1	2019-04-26 11:15:20	scroll_down
1	2019-04-26 11:15:25	scroll_up
1	2019-04-26 11:15:35	page_exit
0	2019-04-28 14:30:15	page_load
0	2019-04-28 14:30:10	page_load
0	2019-04-28 13:30:40	scroll_down
0	2019-04-28 15:31:40	page_exit

Expected Output

user_id	session_time
0	1883.5000
1	35.0000

- Return the column user_id ordered in ascending order.
- Return the session_time for each user.

```
with CTE_page_load as
(
select user_id, DATE_FORMAT(timestamp, '%Y-%m-%d') as Load_Date , max(timestamp) as
page_load_time
from facebook_session
where action = 'page_load'
group by 1, 2
),
CTE_page_exit as
(
select user_id, DATE_FORMAT(timestamp, '%Y-%m-%d') as Exit_Date , min(timestamp) as
page_exit_time
from facebook_session
where action = 'page_exit'
group by 1,2
)
select L.user_id, avg(timestampdiff(second, L.page_load_time,E.page_exit_time)) as session_time
from CTE_page_load L
join CTE_page_exit E on L.user_id = E.user_id and L.Load_Date = E.Exit_Date
group by 1
order by 1;
```