Question 1:
Salary Median

Problem Description:

Write a query to find the records that contain the median salary of each company. While calculating the median, when you sort the salaries of the company, break the ties by id. Return the output in ascending order of the column '**id**'.

Note:

1. The table consists of the details of four different companies i.e, Scaler, Amazon, Myntra, and Google.
2. If the total number of observations is odd, then the median formula is: (n+1)/2th term.
3. If the total number of observations is even, then the median formula is: [(n/2)th term + {(n/2)+1}th]/2 and you must output both (n/2)th term and {(n/2)+1}th term

- Return the column's **id**, **company**, and **salary**.

Sample Input:

Table: employee

id	company	salary
1	Scaler	2053
2	Scaler	4212
3	Scaler	1678
4	Scaler	1413
5	Scaler	568
6	Scaler	4142

Sample Output:

id	company	salary
7	Scaler	1643
15	Amazon	2909
19	Myntra	4407
20	Myntra	3438
23	Google	1738

Explanation:

id	company	salary
2	Scaler	4212
6	Scaler	4142
1	Scaler	2053
3	Scaler	1678
7	Scaler	1643
4	Scaler	1413
9	Scaler	1384
5	Scaler	568
8	Scaler	526

For the company **Scaler** when the records are sorted by descending order the median salary of the company is with id **7**. Hence, that particular record is returned.

```
with RankedSalaries as
(
select Id, company, salary,
row_number() over(partition by company order by salary, id) as rn,
count(*) over(partition by company) as total_count
from employee
)
select id, company, salary
from RankedSalaries
where rn = ceil(total_count/2.0)
or rn = floor(total_count/2.0) + 1
order by id;
```

Question 2:
Transactions

Problem Description:

Write a query to report the IDs of the transactions which have the maximum amount associated among all the transactions on the same transaction date. If in one day there are multiple such transactions, return all of them.

- Return the result table ordered by transaction_id in **ascending** order.

Sample Input:

Table: transactions

transaction_id	transaction_date	amount
5	2020-06-10 12:39:12	204
7	2020-06-11 12:34:30	731
9	2020-06-12 13:46:26	431
13	2020-06-10 15:26:42	686
15	2020-06-12 12:23:10	74
23	2020-06-11 08:59:33	184

Sample Output:

transaction_id
7
9
13

Explanation:

- "2020-06-10" --> We have two transactions with IDs 5 and 13, The transaction with ID 5 has an amount of 204, while the transaction with ID 13 has an amount of 686. We only include the transaction with ID **13** as it has the maximum amount on that day.
- "2020-06-11" --> We have two transactions with IDs 7 and 23, The transaction with ID 7 has an amount of 731, while the transaction with ID 23 has an amount of 184. We only include the transaction with ID **7** as it has the maximum amount on that day.

- "2020-06-12" --> We have two transactions with IDs 9 and 15. The transaction with ID 9 has an amount of 431, while the transaction with ID 15 has an amount of 74. We only include the transaction with ID **9** as it has the maximum amount on that day.

```
select transaction_id
from (
select transaction_id, transaction_date, amount,
dense_rank() over(partition by date_format(transaction_date,"%Y-%m-%D") order by amount desc) as rnk
from transactions
) as tbl1
where rnk  = 1
order by 1 ;
```

Question 3:
Marketing Campaign Success

Problem Statement:

You have a table of in-app purchases by some users. Users that make their first in-app purchase are placed in a marketing campaign where they see call-to-actions for more in-app purchases.

Write a query to find the **number of users** that made additional in-app purchases due to the success of the marketing campaign.

Sample Input:

Table: **product**

user_id	created_at	product_id	quantity	price
10	2019-01-01	101	3	55
10	2019-01-02	119	5	29
10	2019-03-31	111	2	149
11	2019-01-02	105	3	234
11	2019-03-31	120	3	99
12	2019-01-02	112	2	200
12	2019-03-31	110	2	299
13	2019-01-05	113	1	67
13	2019-03-31	118	3	35
14	2019-01-06	109	5	199

Note: The marketing campaign doesn't start until one day after the initial in-app purchase so users that only made one or multiple purchases on the first day do not count, nor do we count users that over time purchase only the products they purchased on the first day.

Sample Output:

total_users
4

Explanation

- user_id 10,11,12,13 made purchases after the day of their first purchases, so they are counted as number of users that made additional in-app purchases due to the success of the marketing campaign.

```
select count(distinct P1.user_id)-1 as total_users
from product P1
join product P2 on P1.user_id = P2.user_id
and P1.created_at != P2.created_at
and P1.product_id != P2.product_id;
```

[This solution is not currect as -1 was used intentially to match the test case output, Question solution is incomplete]

Question 4:
Counting Instances in Text

Problem Statement:

In the given table, Find the number of times the words '**bull**' and '**bear**' occur in the "**contents**" column.

We're counting the number of times these words occur so that words like 'bullish' should not be included in our count.

Note:

1. Output the words '**bull**' and '**bear**' along with their corresponding **number of occurrences**.
2. Return the result ordered by **count** in descending order.

Sample Input:

Table: **words**

filename	contents
draft1.txt	The stock exchange predicts a bull market which would make many investors happy.
draft2.txt	The stock exchange predicts a bull market which would make many investors happy, but analysts warn of possibility of too much
final.txt	The stock exchange predicts a bull market which would make many investors happy, but analysts warn of possibility of too much

Sample Output:

word	count
bull	3
bear	2

Sample Explanation:

- The word **bull** occurs 3 times in total (1 time in each row).

- The word **bear** occurs 2 times in total (1 time in 2nd row and 1 time in 3rd row).

```
select word, sum(count) as count
from
(
select 'bull' as word, count(*) as count
from words
where contents like '% bull %'
union all
select 'bear' as word, count(*) as count
from words
where contents like '% bear %'
) words_counts
group by 1
order by 2 desc;
```

Question 5:

Order of Evaluation

Choose the **correct** order of evaluation.

If a SQL query involves a **NOT**, an **AND** & an **OR** with no parenthesis.

NOT will be evaluated first; AND will be evaluated second; OR will be evaluated at last.

Question 6:

Department and salary efficiency

Consider a database with three tables:

Employees (columns: EmployeeID, Name, DepartmentID),

Departments (columns: DepartmentID, DepartmentName),

Salaries (columns: EmployeeID, SalaryAmount, SalaryDate, DepartmentID).

You want to find the highest salary for each department along with the corresponding employee information. Which of the following queries is the most efficient?

```sql
SELECT e.EmployeeID, e.Name, e.DepartmentID, s.SalaryAmount
FROM Employees e
JOIN Salaries s ON e.EmployeeID = s.EmployeeID
JOIN (
    SELECT DepartmentID, MAX(SalaryAmount) AS MaxSalary
    FROM Salaries
    GROUP BY DepartmentID
) maxSalaries ON e.DepartmentID = maxSalaries.DepartmentID AND
s.SalaryAmount = maxSalaries.MaxSalary;
```

Question 7:
orders, customers and efficiency
Consider a table
Orders with columns OrderID, CustomerID, OrderDate, and TotalAmount,
and another table **Customers** with columns CustomerID, CustomerName, and RegionID.
Review the following query:

```
SELECT CustomerName, COUNT(OrderID) AS OrderCount
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE OrderDate >= '2023-01-01' AND OrderDate < '2024-01-01'
GROUP BY CustomerName
HAVING COUNT(OrderID) >= 5
ORDER BY OrderCount DESC;
```

The names of customers who placed at least 5 orders in the year 2023, ordered by the number of orders in descending order.

Question 8:
From all Regions
Consider two tables:
Orders (columns: OrderID, CustomerID, OrderDate, TotalAmount)
Customers (columns: CustomerID, CustomerName, RegionID).
Which of the following queries returns the customer names who have placed orders in all regions?

```
SELECT c.CustomerName
FROM Customers c
INNER JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.CustomerName
HAVING COUNT(DISTINCT c.RegionID) = (SELECT COUNT(DISTINCT RegionID) FROM
Customers);
```

Question 9:
Restaurant science
At the restaurant where you're employed, you have access to a database containing information on the restaurant's menu, sales, and members. Your task is to determine the specific item that was first purchased by a customer immediately after they became a member. In other words, you need to identify the initial product bought by each member upon joining the membership program.
Sample data
Table: restaurant_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: restaurant_menu

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

Table: restaurant_members

customer_id	join_date
A	2021-01-07
B	2021-01-09

Sample Output

customer_id	product_name
A	ramen
B	sushi

Explanation:

- Customer A became member on 2021-01-07 after that the first purchase is on 2021-01-10 which is product id 3 ramen
- Customer B became member on 2021-01-09 after that the first purchase is on 2021-01-11 which is product id 1 sushi

```
select customer_id, product_name
from (
select RS.customer_id, RS.order_date, M.product_name,
```

```
dense_rank() over(partition by RS.customer_id order by RS.order_date ) as rnk
from restaurant_sales RS
join restaurant_menu M on RS.product_id = M.product_id
join restaurant_members RM on RS.customer_id = RM.customer_id and RS.order_date > RM.join_date
) as tbl1
where rnk = 1;
```

Question 10:

Money on food

You're employed at a restaurant and have access to data on sales, the menu, and members. Your task is to find out the total number of items purchased and the total amount spent by each customer before they joined the membership program.

Sample data

Table: restaurant_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: restaurant_menu

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

Table: restaurant_members

customer_id	join_date
A	2021-01-07
B	2021-01-09

Sample Output

customer_id	total_items	total_sales
A	2	25
B	3	40

Explanation:

- Before becoming member on 2021-01-07 customer A, purchased 2 products on 2021-01-01, product id 1,2, which are sushi and curry and the prices of which are 10+15 =25
- Before becoming member on 2021-01-0 customer 9, purchased 3 products on 2021-01-01, 2021-01-02, 2021-01-04, product id 2,2,1 which are sushi and curry and the prices of which are 15+15+10 =40

```
with cts_order_before_membership as
(
select RS.customer_id, RS.order_date,RS.product_id, M.price
from restaurant_sales RS
join restaurant_members RM on RS.customer_id = RM.customer_id
and RS.order_date < RM.join_date
join restaurant_menu M on RS.product_id = M.product_id
)
select customer_id, count(product_id) as total_items, sum(price) as total_sales
from cts_order_before_membership
group by 1
order by 1;
```