Problem Statement:-

Transatcion_tbl Table has four columns CustID, TranID, TranAmt, and TranDate. User has to display all these fields along with maximum TranAmt for each CustID and ratio of TranAmt and maximum TranAmt for each transaction.

```
Select * from Transaction_Tbl;
```

133 %

Results | Messages

	CustID	TranID	TranAmt	TranDate
1	1001	20001	10000	2020-04-25
2	1001	20002	15000	2020-04-25
3	1001	20003	80000	2020-04-25
4	1001	28004	20000	2020-04-25
5	1002	30001	7000	2020-04-25
6	1002	30002	15000	2020-04-25
7	1002	30003	22000	2020-04-25

```
SELECT
  cust_id,
  tran_id,
  dept,
  tran_amount,
  MAX(tran_amount) OVER (PARTITION BY dept) AS max_tran_amount_dept,
  tran_amount / MAX(tran_amount) OVER (PARTITION BY dept) AS amount_ratio
FROM transactions;
```

Problem Statement:-

Given below table Emp as Input which has two columns 'Group' and 'Sequence', Write a SQL query to find the maximum and minimum values of continuous 'Sequence' in each 'Group'

INPUT

```
SELECT * FROM Emp;
```

133 %

Results | Messages

	Group	Sequence
1	A	1
2	A	2
3	A	3
4	A	5
5	A	6
6	A	8
7	A	9
8	B	11
9	C	1
10	C	2
11	C	3

OUTPUT

	Group	Min_Seq	Max_Seq
1	A	1	3
2	A	5	6
3	A	8	9
4	B	11	11
5	C	1	3

```
-- FINAL
SELECT [Group],
MIN([Sequence]) As Min_Seq,
MAX([Sequence]) As Max_Seq
FROM
(
SELECT [Group],
[Sequence],
[Sequence] - ROW_NUMBER() OVER(Partition BY [Group] ORDER BY [Sequence]) as [Split]
From Emp
) A
GROUP BY [Group],[Split]
ORDER BY [Group]
```

Problem Statement:-

Student Table has three columns Student_Name, Total_Marks and Year. User has to write a SQL query to display Student_Name, Total_Marks, Year, Prev_Yr_Marks for those whose Total_Marks are greater than or equal to the previous year

INPUT

```
select * from Student
```

	Student_Name	Total_Marks	Year
1	Rahul	90	2010
2	Sanjay	80	2010
3	Mohan	70	2010
4	Rahul	90	2011
5	Sanjay	85	2011
6	Mohan	65	2011
7	Rahul	80	2012
8	Sanjay	80	2012
9	Mohan	90	2012

OUTPUT

	Student_Name	Total_Marks	Year	Prev_Yr_Marks
1	Rahul	90	2011	90
2	Sanjay	85	2011	80
3	Mohan	90	2012	65

```
SELECT
  student_name, marks, year, prev_year_marks
FROM (
  SELECT
    student_name,marks,year,
    LAG(marks) OVER (
      PARTITION BY student_name
      ORDER BY year
    ) AS prev_year_marks
  FROM student
) t
WHERE marks >= prev_year_marks;
```

Problem Statement:-

Emp_Details Table has four columns EmpID, Gender, EmailID and DeptID. User has to write a SQL query to derive another column called Email_List to display all Emailid concatenated with semicolon associated with a each DEPT_ID as shown below in output Table.

INPUT

```
SELECT * FROM Emp_Details ;
```

	EMPID	Gender	EmailID	DeptID
1	1001	M	YYYYY@gmaix.com	104
2	1002	M	ZZZ@gmaix.com	103
3	1003	F	AAAAA@gmaix.com	102
4	1004	F	PP@gmaix.com	104
5	1005	M	CCCC@yahu.com	101
6	1006	M	DDDDD@yahu.com	100
7	1007	F	E@yahu.com	102
8	1008	M	M@yahu.com	102
9	1009	F	SS@yahu.com	100

OUTPUT

	DeptID	Email_List
1	100	DDDDD@yahu.com;SS@yahu.com
2	101	CCCC@yahu.com
3	102	AAAAA@gmaix.com;E@yahu.com;M@yahu.com
4	103	ZZZ@gmaix.com
5	104	PP@gmaix.com;YYYYY@gmaix.com

```
SELECT DeptID,STRING_AGG(EmailID,';') WITHIN GROUP (Order by EmailID) Email_List
FROM Emp_Details
GROUP BY DeptID
```

INPUT :- ORDER_Tbl

	ORDER_DAY	ORDER_ID	PRODUCT_ID	QUANTITY	PRICE
1	2015-05-01	ODR1	PROD1	5	5
2	2015-05-01	ODR2	PROD2	2	10
3	2015-05-01	ODR3	PROD3	10	25
4	2015-05-01	ODR4	PROD1	20	5
5	2015-05-02	ODR5	PROD3	5	25
6	2015-05-02	ODR6	PROD4	6	20
7	2015-05-02	ODR7	PROD1	2	5
8	2015-05-02	ODR8	PROD5	1	50
9	2015-05-02	ODR9	PROD6	2	50
10	2015-05-02	ODR10	PROD2	4	10

Order_Tbl has four columns namely ORDER_DAY, ORDER_ID, PRODUCT_ID, QUANTITY and PRICE

Problem Statements :-

(a) Write a SQL to get all the products that got sold on both the days and the number of times the product is sold.

	PRODUCT_ID	COUNT
1	PROD1	3
2	PROD2	2
3	PROD3	2

(b) Write a SQL to get products that was ordered on 02-May-2015 but not on 01-May-2015

	PRODUCT_ID
1	PROD4
2	PROD5
3	PROD6

a) Select product_id, count(order_id) as count
From orders
Group by product_id
Where count(distinct(order_Day))=2

```
SELECT PRODUCT_ID
FROM Order_Tbl WHERE ORDER_DAY='2015-05-02'
EXCEPT
SELECT PRODUCT_ID
FROM Order_Tbl WHERE ORDER_DAY='2015-05-01'
```

B) select distinct product_id from Order_Tbl
where ORDER_DAY = '2015-05-02' and
PRODUCT_ID not in (select distinct product_id from Order_Tbl where ORDER_DAY = '2015-05-01')

Problem Statements :-

(a) Write a SQL to get the highest sold Products (Quantity*Price) on both the days

	ORDER_DAY	PRODUCT_ID	Sold_Amount
1	2015-05-01	PROD3	250
2	2015-05-02	PROD3	125

(b) Write a SQL to get all product's total sales on 1st May and 2nd May adjacent to each other

	PRODUCT_ID	Total_Sales_01	Total_Sales_02
1	PROD1	125	10
2	PROD2	20	40
3	PROD3	250	125
4	PROD4	0	120
5	PROD5	0	50
6	PROD6	0	100

(c) Write a SQL to get all products day wise, that was ordered more than once

	ORDER_DAY	PRODUCT_ID
1	2015-05-01	PROD1

A) SELECT
product_id,
SUM(quantity * price) AS total_sales
FROM sales
GROUP BY product_id
HAVING COUNT(DISTINCT date) = 2
ORDER BY total_sales DESC
LIMIT 1;

b) select product_id ,
sum(case when order_Day = '2015-05-01' then quantity*price else 0 end),
sum(case when order_Day = '2015-05-02' then quantity*price else 0 end)
from Order_Tb
group by product_id

c)

```
-- Write a SQL to get all products day wise, that was ordered more than once

SELECT ORDER_DAY,PRODUCT_ID--,COUNT(*)
FROM Order_Tbl
GROUP BY ORDER_DAY,PRODUCT_ID
HAVING COUNT(*) > 1
```

INPUT :- Order_Tbl has three columns namely ORDER_ID, PRODUCT_ID and QUANTITY

	Order_ID	Product_ID	Quantity
1	ODR1	PRD1	5
2	ODR2	PRD2	1
3	ODR3	PRD3	3

Problem Statements :-

Write a SQL which will explode the above data into single unit level records as shown below

	Order_ID	Product_ID	Quantity
1	ODR1	PRD1	1
2	ODR1	PRD1	1
3	ODR1	PRD1	1
4	ODR1	PRD1	1
5	ODR1	PRD1	1
6	ODR2	PRD2	1
7	ODR3	PRD3	1
8	ODR3	PRD3	1
9	ODR3	PRD3	1

```
select order_id, product_id, 1 as quantity
from Order_Tbl o
CROSS APPLY GENERATE_SERIES(1, o.quantity)
```

Input :- Team Table has two columns namely ID and TeamName and it contains 4 TeamsName.

ID	TeamName
1	India
2	Australia
3	England
4	NewZealand

Problem Statements :- Write a SQL which will fetch total schedule of matches between each team vs opposite team:

	Matches
1	India Vs Australia
2	India Vs England
3	India Vs NewZealand
4	Australia Vs England
5	Australia Vs NewZealand
6	England Vs NewZealand

```
Select a.Teamname + 'VS' + b.Teamname from Team a join team b on a.id<b.Id
```

Input :- Transaction_Table has four columns namely AccountNumber, TransactionTime, TransactionID and Balance

	AccountNumber	TransactionTime	TransactionID	Balance
1	550	2020-05-12 05:29:44.120	1001	2000
2	550	2020-05-15 10:29:25.630	1002	8000
3	460	2020-03-15 11:29:23.620	1003	9000
4	460	2020-04-30 11:29:57.320	1004	7000
5	460	2020-04-30 12:32:44.233	1005	5000
6	640	2020-02-18 06:29:34.420	1006	5000
7	640	2020-02-18 06:29:37.120	1007	9000

Problem Statements :- Write SQL to get the most recent / latest balance, and TransactionID for each AccountNumber

	AccountNumber	TransactionTime	TransactionID	Balance
1	550	2020-05-15 10:29:25.630	1002	8000
2	460	2020-04-30 12:32:44.233	1005	5000
3	640	2020-02-18 06:29:37.120	1007	9000

```
with cte as (
select *,
(ROW_NUMBER() over (partition by AccountNumber order by TransactionTime desc )) as rn
from Transaction_Table )
select * from cte where cte.rn = 1
```

Input :- StudentInfo Table has four columns namely StudentName, English, Maths and Science

	StudentName	English	Maths	Science
1	David	85	90	88
2	John	75	85	80
3	Tom	83	80	92

Problem Statements :- Write SQL to turn the columns English, Maths and Science into rows. It should display Marks for each student for each subjects as shown below

	StudentName	Subjects	Marks
1	David	English	85
2	David	Maths	90
3	David	Science	88
4	John	English	75
5	John	Maths	85
6	John	Science	80
7	Tom	English	83
8	Tom	Maths	80
9	Tom	Science	92

```
Select Studentname,'English' as English,English as Marks from un
UNION ALL
Select StudentName,'Math' as Math,math from un
UNION ALL
Select Studentname,'Science' as Science ,science from un

select * from studentinfo

UNPIVOT ( marks for subjects in (English, Maths, Science) ) as unpvt
```

Input :- Trade_Tbl has five columns namely Trade_ID, Trade_Timestamp, Trade_Stock, Quantity and Price

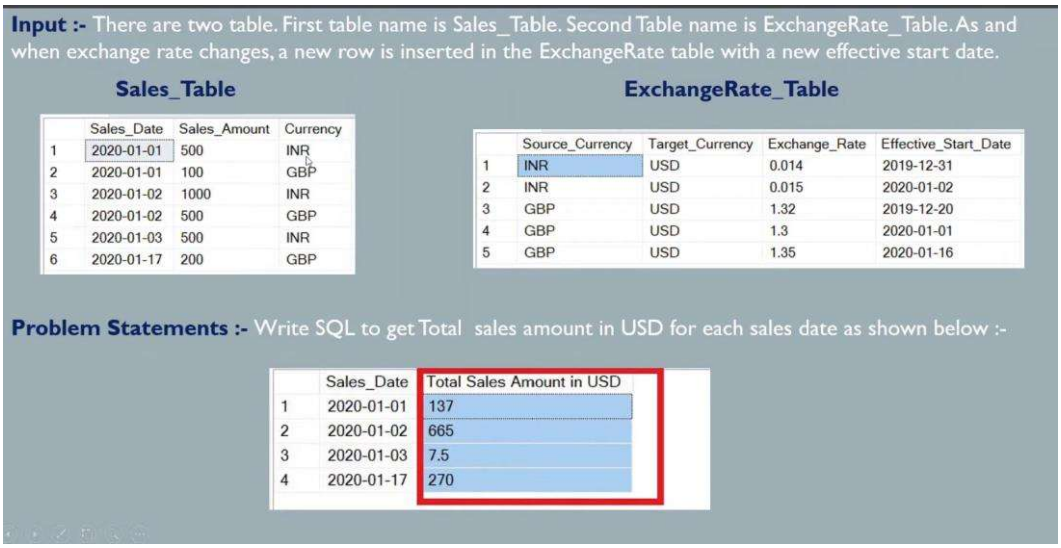
	TRADE_ID	Trade_Timestamp	Trade_Stock	Quantity	Price
1	TRADE1	10:01:05.0000000	ITJunction4All	100	20
2	TRADE2	10:01:06.0000000	ITJunction4All	20	15
3	TRADE3	10:01:08.0000000	ITJunction4All	150	30
4	TRADE4	10:01:09.0000000	ITJunction4All	300	32
5	TRADE5	10:10:00.0000000	ITJunction4All	-100	19
6	TRADE6	10:10:01.0000000	ITJunction4All	-300	19

Problem Statements :- Write SQL to find all couples of trade for same stock that happened in the range of 10 seconds and having price difference by more than 10 %. Output result should also list the percentage of price difference between the 2 trade

	First_Trade	Second_Trade	Price_Diff
1	TRADE1	TRADE2	25
2	TRADE1	TRADE3	50
3	TRADE1	TRADE4	60
4	TRADE2	TRADE3	100
5	TRADE2	TRADE4	113

```
WITH base_cte AS (
  SELECT *,
    LEAD(Trade_id) OVER (
      PARTITION BY Trade_Stock
      ORDER BY Trade_Timestamp
    ) AS trade_2,
    LEAD(Trade_Timestamp) OVER (
      PARTITION BY Trade_Stock
      ORDER BY Trade_Timestamp
    ) AS next_trade_ts,
    LEAD(Price) OVER (
      PARTITION BY Trade_Stock
      ORDER BY Trade_Timestamp
    ) AS next_price
  FROM trades
)
SELECT
  Trade_id,trade_2,
  (ABS(Price - next_price) / Price) * 100 AS Diff
FROM base_cte
WHERE
  DATEDIFF(SECOND, Trade_Timestamp, next_trade_ts) < 11
  AND ABS(Price - next_price) / Price >= 0.01
ORDER BY Trade_Timestamp;
```

```
Select
A.Trade_id As First_Trade ,
B.Trade_id As Second_Trade,
FLOOR(ABS(((B.Price - A.Price)/A.Price)*100)) As Price_Diff
From Trade_CTE  As A
INNER JOIN  Trade_CTE  As B
ON A.Trade_id < B.Trade_id
where DATEDIFF(SECOND, A.Trade_Timestamp , B.Trade_Timestamp) <=10
AND ABS(((B.Price - A.Price)/A.Price)*100) >= 10
order by 1 ;
```



Input :- There are two table. First table name is Sales_Table. Second Table name is ExchangeRate_Table. As and when exchange rate changes, a new row is inserted in the ExchangeRate table with a new effective start date.

Sales_Table

	Sales_Date	Sales_Amount	Currency
1	2020-01-01	500	INR
2	2020-01-01	100	GBP
3	2020-01-02	1000	INR
4	2020-01-02	500	GBP
5	2020-01-03	500	INR
6	2020-01-17	200	GBP

ExchangeRate_Table

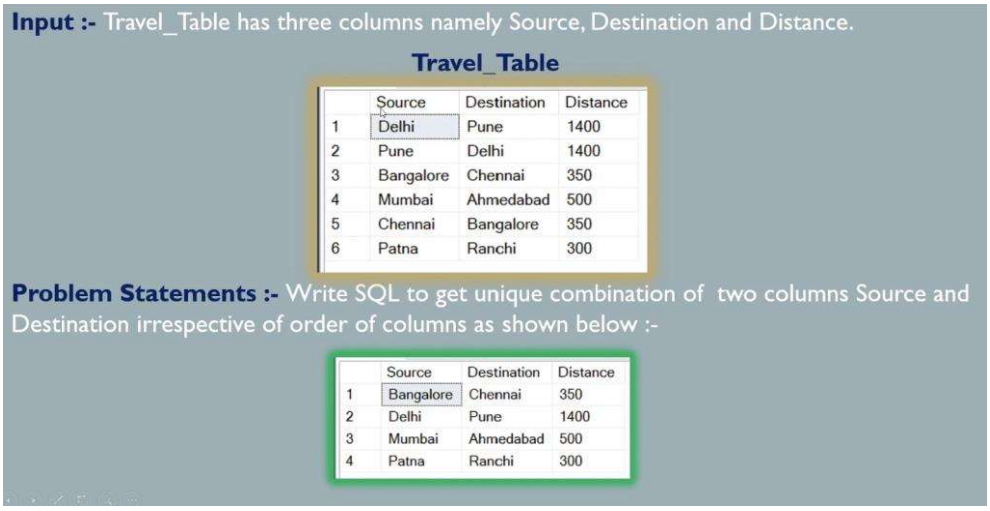
	Source_Currency	Target_Currency	Exchange_Rate	Effective_Start_Date
1	INR	USD	0.014	2019-12-31
2	INR	USD	0.015	2020-01-02
3	GBP	USD	1.32	2019-12-20
4	GBP	USD	1.3	2020-01-01
5	GBP	USD	1.35	2020-01-16

Problem Statements :- Write SQL to get Total sales amount in USD for each sales date as shown below :-

	Sales_Date	Total Sales Amount in USD
1	2020-01-01	137
2	2020-01-02	665
3	2020-01-03	7.5
4	2020-01-17	270

```
WITH exchange_ranges AS (
  SELECT
    source_currency,
    target_currency,
    exchange_rate,
    effective_start_date,
    DATEADD(
      DAY,
      -1,
      LEAD(effective_start_date) OVER (
        PARTITION BY source_currency
        ORDER BY effective_start_date
      )
    ) AS effective_end_date
  FROM ExchangeRate_Table
),
sales_converted AS (
  SELECT
```

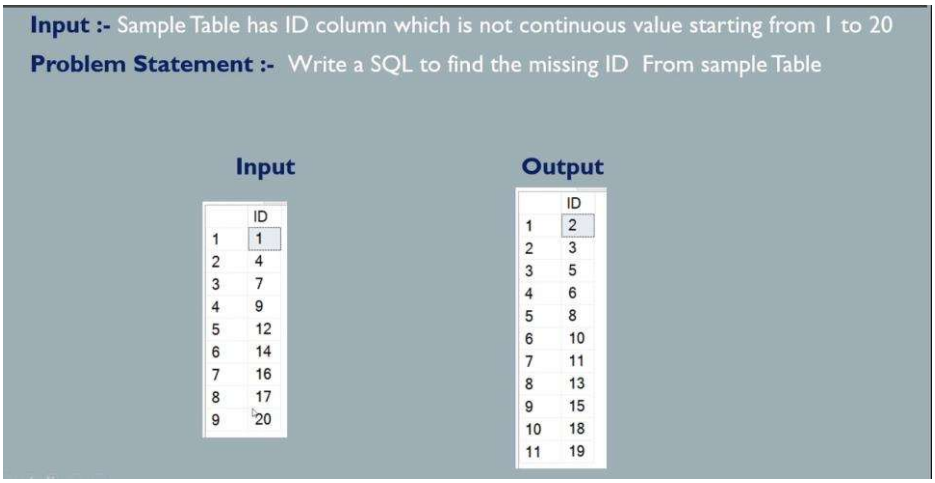
```
    s.sales_date,
    s.sales_amount,
    e.exchange_rate
  FROM Sales_Table s
  JOIN exchange_ranges e
    ON s.currency = e.source_currency
   AND s.sales_date >= e.effective_start_date
   AND s.sales_date <= ISNULL(e.effective_end_date, '9999-12-31')
)
SELECT
  sales_date,
  SUM(sales_amount * exchange_rate) AS total_sales_amount_in_usd
FROM sales_converted
GROUP BY sales_date
ORDER BY sales_date;
```



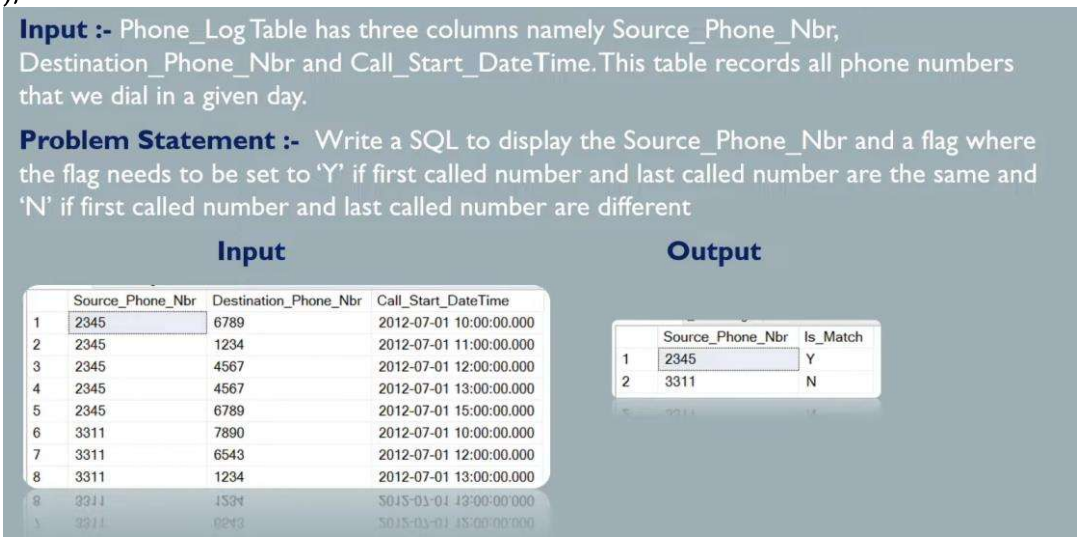
	Source	Destination	Distance
1	Delhi	Pune	1400
2	Pune	Delhi	1400
3	Bangalore	Chennai	350
4	Mumbai	Ahmedabad	500
5	Chennai	Bangalore	350
6	Patna	Ranchi	300

	Source	Destination	Distance
1	Bangalore	Chennai	350
2	Delhi	Pune	1400
3	Mumbai	Ahmedabad	500
4	Patna	Ranchi	300

```
SELECT *
FROM Travel_Table t1
WHERE NOT EXISTS (
  SELECT 1
  FROM Travel_Table t2
  WHERE t1.Distance = t2.Distance
    AND t1.Start_Location = t2.End_Location
    AND t1.End_Location = t2.Start_Location
    AND t1.Start_Location > t2.Start_Location
);
```



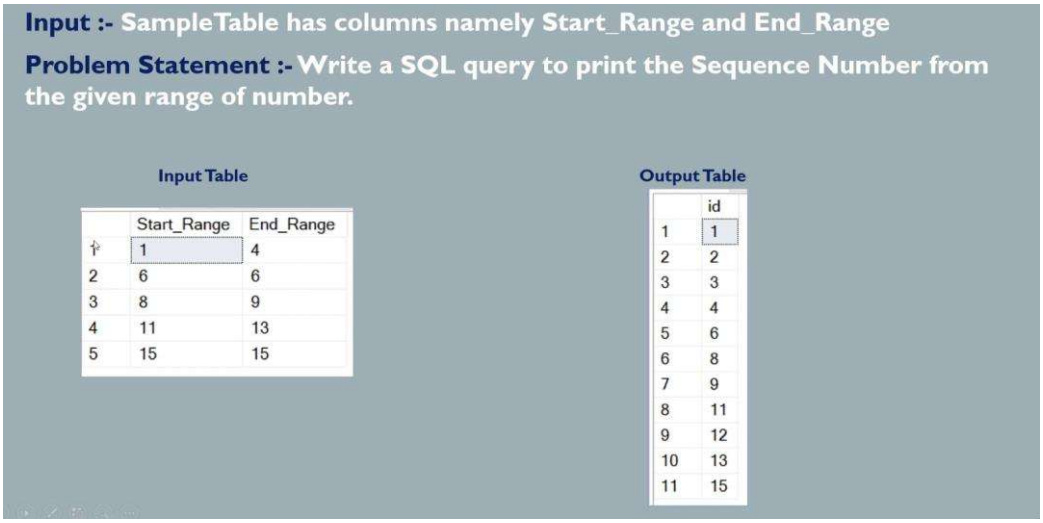

```
with cte as (
select 1 as ID
Union all
select ID+1 from cte
where Id<20 )
select * from cte where ID not in (select id from Sample_Table)
SELECT gs.id
FROM generate_series(1, 20) AS gs(id)
WHERE gs.id NOT IN (
   SELECT id FROM Sample_Table
);
```



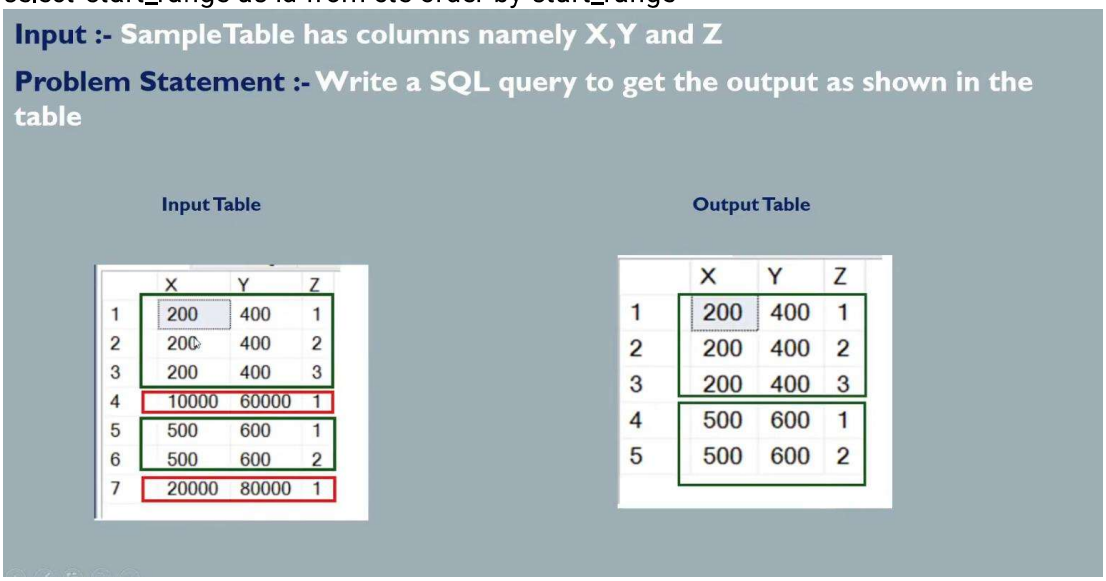

```
select distinct source_phone_nbr,
case when flag = 0 then 'Y' else 'N' end as match from (
select *, first_value(destination_phone_nbr) over(partition by source_phone_nbr order by
call_start_datetime) - first_value(destination_phone_nbr) over(partition by source_phone_nbr order
by call_start_datetime desc)
flag from Phone_Log );


with cte as(
```

```
select *,
first_value(destination_phone_nbr) over(partition by source_phone_nbr)as f_value,
last_value(destination_phone_nbr) over(partition by source_phone_nbr)as l_value from Phone_Log),
 final_cte as(
 select source_phone_nbr,
case when f_value=l_value then 'Y' else 'N' end as is_match,
row_number() over(partition by source_phone_nbr)as rn
from cte)
select source_phone_nbr,is_match from final_cte where rn=1
```




```
with cte as(
 select start_range, end_range
from sampletable
union all
select start_range+1,end_range from cte
where start_range+1<=end_range)
select start_range as id from cte order by start_range
```



```
Select * from Sample_1 where X in
 (Select X from Sample_1
group by X
having count(*) > 1)
```

Input :- Sales Table has three columns namely Id, Product and Sales

Problem Statement :- Write a SQL query to get the output as shown in the Output tables

Input Table

	Id	Product	Sales
1	1001	Keyboard	20
2	1002	Keyboard	25
3	1003	Laptop	30
4	1004	Laptop	35
5	1005	Laptop	40
6	1006	Monitor	45
7	1007	WebCam	50
8	1008	WebCam	55

Output Table 1

	Id	Product	Sales_1
1	1001	Keyboard	20
2	1002	Keyboard	20
3	1003	Laptop	30
4	1004	Laptop	30
5	1005	Laptop	30
6	1006	Monitor	45
7	1007	WebCam	50
8	1008	WebCam	50

Output Table 2

	Id	Product	Sales_2
1	1001	Keyboard	20
2	1002	Keyboard	45
3	1003	Laptop	30
4	1004	Laptop	65
5	1005	Laptop	105
6	1006	Monitor	45
7	1007	WebCam	50
8	1008	WebCam	105

```
Select e.*,
first_value(sales)over(partition by product order by id) as fv,
sum(sales)over (partition by product order by id)
as rt from sales1 e;
```

Input :- StudentInfo Table has three columns namely StudentName, Subjects and Marks

Problem Statement :- Write a SQL query to get the output as shown in the Output table

Input Table

	StudentName	Subjects	Marks
1	David	English	85
2	David	Maths	90
3	David	Science	88
4	John	English	75
5	John	Maths	85
6	John	Science	80
7	Tom	English	83
8	Tom	Maths	80
9	Tom	Science	92

Output Table

	Studentname	English	Maths	Science
1	David	85	90	88
2	John	75	85	80
3	Tom	83	80	92

```
select StudentName,
sum(case when Subjects= 'English' then Marks else 0 end) as English,
sum(case when Subjects= 'Maths' then Marks else 0 end) as Maths,
sum(case when Subjects= 'Science' then Marks else 0 end) as Science
 from StudentInfo1 group by StudentName order by StudentName;
```

```sql
--Solution
Select StudentName,English,Maths,Science
From
(Select StudentName,Subjects,Marks From StudentInfo_1) As SourceTable
PIVOT
(
MAX(Marks)
FOR Subjects IN (English,Maths,Science)
) As PivotTable
```

Problem Statement :- Order Status Table has three columns namely Quote_id, Order_id and Order_Status

When all Orders are in delivered status then Quote status should be 'Complete'.
When one or more Orders in delivered status then " In Delivery".
When One or more in Submitted status then "Awaiting for Submission" Else "Awaiting for Entry" by default

Note :- Order Priority should be Delivered, Submitted and Created

If one order is in delivered and other one is in Submitted then Quote_Status should be "In Delivery"

Similarly if one order is in Submitted and others in Created then the Quote_Status should be "Awaiting for Submission "

Input Table

	Quote_Id	Order_Id	Order_Status
1	A	A1	Delivered
2	A	A2	Delivered
3	A	A3	Delivered
4	B	B1	Submitted
5	B	B2	Delivered
6	B	B3	Created
7	C	C1	Submitted
8	C	C2	Created
9	C	C3	Submitted
10	D	D1	Created

Output Table

	Quote_Id	Quote_Status
1	A	Complete
2	B	In Delivery
3	C	Awaiting For Submission
4	D	Awaiting For Entry

```sql
with cte as (
 select  quote_id ,
 count(case when order_status='Delivered' then 1 end) delivered_count
 count(case when order_status='Submitted' then 1 end) submitted_count ,
 count(*) as total_count
from orderstatus group by quote_id )
select   quote_id ,
case when delivered_count = total_count then 'Complete' ,
when delivered_count > 0 then 'In Delivery'
when submitted_count > 0 then 'Awaiting For Submission'
else 'Awaiting For Entry' end as quote_status from cte
```

Problem Statement :- Employees Table has five columns namely Employee_no, Birth_date , first_name , last_name and Joining_date

Input Table

	Employee_no	Birth_date	First_name	Last_name	Joining_date
1	1001	1988-08-15	ADAM	WAUGH	2012-04-12
2	1002	1990-05-10	Mark	Jennifer	2010-06-25
3	1003	1992-02-07	JOHN	Waugh	2016-02-07
4	1004	1985-06-12	SOPHIA TRUMP		2016-02-15
5	1005	1995-03-25	Maria	Gracia	2013-04-09
6	1006	1994-06-23	ROBERT	PATRICA	2015-06-23
7	1007	1993-04-05	MIKE JOHNSON		2014-03-09
8	1008	1989-04-05	JAMES	OLIVER	2017-06-15

(1.)As a convention the values in first_name and last_name should always be in uppercase. But due to data entry issues some records may not adhere to this convention.Write a query to find all such records where first_name is not in upper case.

(2.) For some records the first_name column has full name and last_name is blank. Write a SQL query to update it correctly,

```
1)  SELECT *
FROM Employees
WHERE first_name <> UPPER(first_name);
2)
UPDATE Employees
SET
   last_name  = SUBSTRING(first_name, CHARINDEX(' ', first_name) + 1, LEN(first_name)),
   first_name = SUBSTRING(first_name, 1, CHARINDEX(' ', first_name) - 1)
WHERE last_name IS NULL
 AND first_name LIKE '% %';
```

Problem Statement :- SalesInfo Table has three columns namely Continents, Country and Sales. Write a SQL query to get the aggregate sum of sales countrywise and display only those which are maximum in each continents.

Input Table

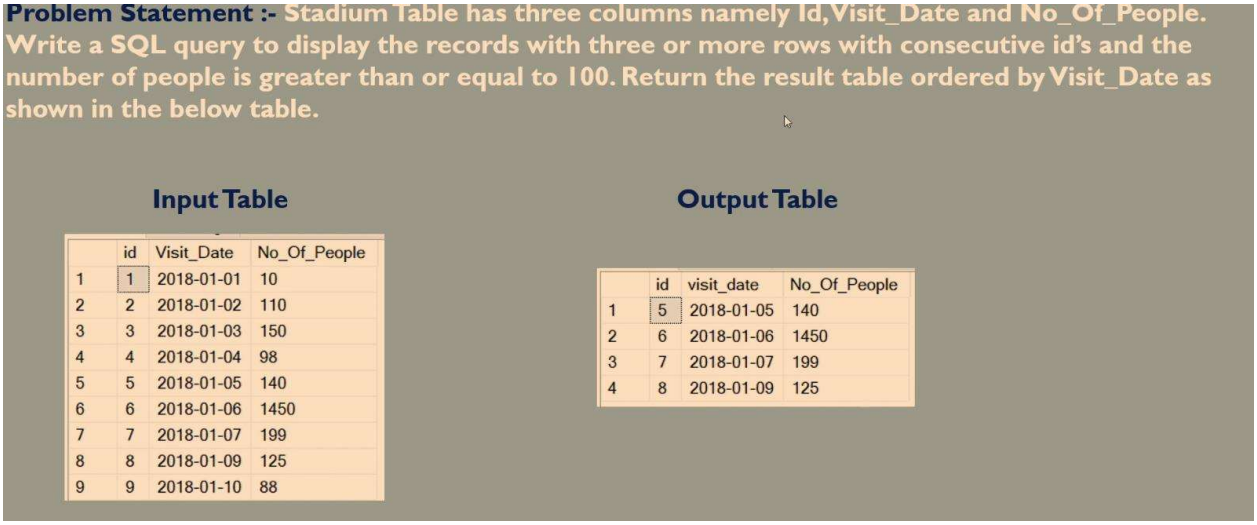
	Continents	Country	Sales
1	Asia	India	50000
2	Asia	India	70000
3	Asia	India	60000
4	Asia	Japan	10000
5	Asia	Japan	20000
6	Asia	Japan	40000
7	Asia	Thailand	20000
8	Asia	Thailand	30000
9	Asia	Thailand	40000
10	Europe	Denmark	40000
11	Europe	Denmark	60000
12	Europe	Denmark	10000
13	Europe	France	60000
14	Europe	France	30000
15	Europe	France	40000

Output Table

	Continents	Country	TotalSales
1	Asia	India	180000
2	Europe	France	130000

```
with cte as (
 select continents, Country,
 sum(sales) as sum,
ROW_NUMBER() over(PARTITION by continents order by sum(sales) desc) as rn
```

from SalesInfo
group by continents, Country)
select continents, Country, sum from cte where rn=1 order by continents, Country



	id	Visit_Date	No_Of_People
1	1	2018-01-01	10
2	2	2018-01-02	110
3	3	2018-01-03	150
4	4	2018-01-04	98
5	5	2018-01-05	140
6	6	2018-01-06	1450
7	7	2018-01-07	199
8	8	2018-01-09	125
9	9	2018-01-10	88

	id	visit_date	No_Of_People
1	5	2018-01-05	140
2	6	2018-01-06	1450
3	7	2018-01-07	199
4	8	2018-01-09	125

◇ 1. What is the Gaps & Islands Problem?

Gaps & Islands refers to identifying:

- **Islands** → continuous sequences of values
- **Gaps** → breaks in continuity

Common continuity dimensions:

Dimension	Example
IDs	1,2,3 → island
Dates	2024-01-01, 02, 03
Numbers	salary, quantity
Events	login streaks

◇ 2. When do you use it?

Typical interview problems:

- Consecutive login days
- Continuous transaction IDs
- 3+ days with high traffic
- Attendance streaks
- Price ranges without gaps

◇ 3. Core Idea (MOST IMPORTANT)

Row numbers always increase by 1
Actual values increase by 1 only if consecutive
→ Their **difference remains constant inside an island**

◇ 4. Sample Dataset

```
id
----
1
2
3
5
6
8
```

◇ 5. Step-by-Step Breakdown

Step 1 — Assign Row Numbers

```
ROW_NUMBER() OVER (ORDER BY id)
id | rn
--------
1  | 1
2  | 2
3  | 3
5  | 4
6  | 5
8  | 6
```

Step 2 — Create Island Key

```
id - ROW_NUMBER() OVER (ORDER BY id) AS grp
id | rn | grp
--------------
1  | 1  | 0
2  | 2  | 0
3  | 3  | 0   ← island
5  | 4  | 1
6  | 5  | 1   ← island
8  | 6  | 2   ← island
```

✓ Same `grp` ⇒ same island

✓ Change in `grp` ⇒ gap detected

◇ 6. Canonical Pattern (Memorize)

```
value - ROW_NUMBER() OVER (ORDER BY value)
```

◇ 7. Final General Query (Single CTE)

```
WITH cte AS (
    SELECT *,
           id - ROW_NUMBER() OVER (ORDER BY id) AS grp
    FROM table_name
)
SELECT *
FROM cte;
```

◇ 8. Filtering Islands (e.g. ≥ 3 rows)

```
WITH cte AS (
    SELECT *,
           id - ROW_NUMBER() OVER (ORDER BY id) AS grp
    FROM table_name
)
SELECT *
FROM cte
WHERE grp IN (
    SELECT grp
    FROM cte
    GROUP BY grp
    HAVING COUNT(*) >= 3
);
```

◇ 9. Gaps & Islands with Conditions

Example: Stadium traffic ≥ 100 people

```
WITH cte AS (
    SELECT *,
           id - ROW_NUMBER() OVER (ORDER BY id) AS grp
    FROM Stadium
    WHERE no_of_people >= 100
)
SELECT id, visit_date, no_of_people
FROM cte
WHERE grp IN (
    SELECT grp
    FROM cte
    GROUP BY grp
    HAVING COUNT(*) >= 3
);
```

◇ 10. Alternative Method — Using LAG( )

Idea

- Detect breaks explicitly
- Use cumulative sum

```
WITH cte AS (
    SELECT *,
           SUM(
               CASE
                   WHEN id = LAG(id) OVER (ORDER BY id) + 1
                   THEN 0
                   ELSE 1
               END
           ) OVER (ORDER BY id) AS grp
    FROM table_name
)
SELECT *
FROM cte;
```

✓ More verbose
✓ Easier to reason
❌ Slightly slower

◇ 11. Dates-Based Islands

```
DATEDIFF(day, '2000-01-01', visit_date)
- ROW_NUMBER() OVER (ORDER BY visit_date)
```

Why?

- Converts dates → continuous integers

◇ 12. Self-Join Method (No Window Functions)

```
SELECT *
FROM table t1
WHERE NOT EXISTS (
    SELECT 1
    FROM table t2
    WHERE t2.id = t1.id - 1
);
```

✓ Works
❌ Hard to extend
❌ Poor performance

◇ 13. Common Mistakes (VERY IMPORTANT)

❌ Using only `LAG()` for 3+ rows
❌ Forgetting to filter before grouping
❌ Grouping directly on ID
❌ Ignoring missing dates
❌ Using ASCII tricks for grouping

◇ 14. Performance Tips

- Filter early (`WHERE condition`)
- Index on ordering column
- Prefer `ROW_NUMBER()` over self-joins
- Avoid nested CTEs if possible

◇ 15. Interview One-Liners (MEMORIZE)

- **Short:**

"I use gaps-and-islands with `value - row_number()`."

- **Detailed:**

"Row numbers increase uniformly; consecutive values keep the difference constant, forming islands."

◇ 16. When NOT to use Gaps & Islands

- When ordering is ambiguous
- When no natural sequence exists
- When duplicates aren't handled

◇ 17. Quick Decision Table

Problem Type	Best Approach
Consecutive IDs	`id - row_number()`
Dates	`datediff - row_number()`
Explicit gaps	`LAG()`
No window functions	Self-join

🗃️ Original Table: `orders`

```
cust_id | order_id | order_date
-------------------------------
```

```
A       | 1         | 2024-01-01
A       | 2         | 2024-01-01
A       | 3         | 2024-01-02
A       | 4         | 2024-01-05
A       | 5         | 2024-01-05
A       | 6         | 2024-01-06
B       | 7         | 2024-01-03
B       | 8         | 2024-01-04
B       | 9         | 2024-01-06
```

STEP 1 — Remove same-day duplicates

Why?
Because `ROW_NUMBER()` increases per row, not per day.

```
SELECT DISTINCT cust_id, order_date
FROM orders;
```

Result: `distinct_dates`

```
cust_id | order_date
--------------------
A       | 2024-01-01
A       | 2024-01-02
A       | 2024-01-05
A       | 2024-01-06
B       | 2024-01-03
B       | 2024-01-04
B       | 2024-01-06
```

STEP 2 — Assign row numbers per customer

```
SELECT
    cust_id,
    order_date,
    ROW_NUMBER() OVER (
        PARTITION BY cust_id
        ORDER BY order_date
    ) AS rn
FROM distinct_dates;
```

Result: `numbered_dates`

```
cust_id | order_date | rn
------------------------
A       | 2024-01-01 | 1
A       | 2024-01-02 | 2
A       | 2024-01-05 | 3
A       | 2024-01-06 | 4
```

```
B       | 2024-01-03 | 1
B       | 2024-01-04 | 2
B       | 2024-01-06 | 3
```

STEP 3 — Create the island key

🔑 Core trick:

```
island_key = order_date - row_number
```

```
SELECT
    cust_id,
    order_date,
    DATE_SUB(order_date, INTERVAL rn DAY) AS island_key
FROM numbered_dates;
```

Result: `islands`

```
cust_id | order_date | island_key
---------------------------------
A       | 2024-01-01 | 2023-12-31
A       | 2024-01-02 | 2023-12-31   ← same island
A       | 2024-01-05 | 2024-01-02
A       | 2024-01-06 | 2024-01-02   ← same island
B       | 2024-01-03 | 2024-01-02
B       | 2024-01-04 | 2024-01-02   ← same island
B       | 2024-01-06 | 2024-01-03
```

❗ Why this works

For consecutive dates:

```
(2024-01-02 - 2) = (2024-01-01 - 1)
```

When a day is missing:

```
the difference changes → new island
```

STEP 4 — Attach island back to original orders

```
SELECT
    o.cust_id,
    o.order_id,
    o.order_date,
    i.island_key
FROM orders o
JOIN islands i
  ON o.cust_id = i.cust_id
 AND o.order_date = i.order_date
ORDER BY o.cust_id, o.order_date, o.order_id;
```

Final Output

```
cust_id | order_id | order_date | island_key
-------------------------------------------
A       | 1        | 2024-01-01 | 2023-12-31
A       | 2        | 2024-01-01 | 2023-12-31
A       | 3        | 2024-01-02 | 2023-12-31
A       | 4        | 2024-01-05 | 2024-01-02
A       | 5        | 2024-01-05 | 2024-01-02
A       | 6        | 2024-01-06 | 2024-01-02
B       | 7        | 2024-01-03 | 2024-01-02
B       | 8        | 2024-01-04 | 2024-01-02
B       | 9        | 2024-01-06 | 2024-01-03
```

STEP 5 — (Optional) Get island ranges

```
SELECT
    cust_id,
    island_key,
    MIN(order_date) AS start_date,
    MAX(order_date) AS end_date,
    COUNT(*) AS orders
FROM (
    SELECT
        o.cust_id,
        o.order_date,
        i.island_key
    FROM orders o
    JOIN islands i
      ON o.cust_id = i.cust_id
     AND o.order_date = i.order_date
) x
GROUP BY cust_id, island_key;
```

Problem Statement :- Write a SQL query to print movie theatre like seating numbers as shown below :-

Output

	Row	Seat_Arrangement
1	A	A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
2	B	B1,B2,B3,B4,B5,B6,B7,B8,B9,B10
3	C	C1,C2,C3,C4,C5,C6,C7,C8,C9,C10
4	D	D1,D2,D3,D4,D5,D6,D7,D8,D9,D10
5	E	E1,E2,E3,E4,E5,E6,E7,E8,E9,E10
6	F	F1,F2,F3,F4,F5,F6,F7,F8,F9,F10
7	G	G1,G2,G3,G4,G5,G6,G7,G8,G9,G10
8	H	H1,H2,H3,H4,H5,H6,H7,H8,H9,H10
9	I	I1,I2,I3,I4,I5,I6,I7,I8,I9,I10
10	J	J1,J2,J3,J4,J5,J6,J7,J8,J9,J10
11	K	K1,K2,K3,K4,K5,K6,K7,K8,K9,K10

```sql
-- Using CTE

With CTE_Alphabet as (
Select CHAR(ASCII('A')) as letter
UNION ALL
Select CHAR(ASCII(letter)+ 1) From CTE_Alphabet where letter <> 'L'
),
CTE_Seat as (
Select 1 as Nmbr
UNION ALL
Select Nmbr +1 From CTE_Seat where Nmbr < 10
),
CTE_Final as (
Select letter , letter + trim(STR(Nmbr)) As Seat_No From CTE_Alphabet Cross Join CTE_Seat
)
Select letter As Row, STRING_AGG(Seat_No, ',') within group(Order by letter)  From CTE_Final
group by letter
```

Problem Statement :- Club Table has three columns namely Club_ID, Member_id and EDU.

Same member can be a part of different club. The EDU column has different rewards. The points for these awards are as follows :-

MM – 0.5, CI – 0.5, CO- 0.5, CD – 1, CL-1, CM – 1

Write a SQL query to find the total points scored by each club as shown in the desired output.

Input : Club Table

	Club_Id	Member_Id	EDU
1	1001	210	NULL
2	1001	211	MM:CI
3	1002	215	CD:CI:CM
4	1002	216	CL:CM
5	1002	217	MM:CM
6	1003	255	NULL

Desired Output :

	Club_Id	Total_Points
1	1001	4.0
2	1002	6.0
3	1003	0.0

```
--By using string_split function

--string_split - A table-valued function that splits a string into rows of substrings, based on a specified separator character.
--Syntax- STRING_SPLIT ( string , separator )
-- OUTER APPLY -- It is similar to LEFT JOIN. The need of OUTER APPLY arises if you want to join table with Table valued function
--Select value From string_split('A,B,C,D',',')

-- 'MM','CI','CO' -> 0.5
-- 'CD','CL','CM' ->  1

Select Club_Id, SUM(Case when Value in ('MM','CI','CO') Then 0.5
                    when Value in ('CD','CL','CM') Then 1 END) As Reward
From(
Select Club_Id,Member_Id,Value From CLub C
OUTER APPLY string_split(C.EDU,':')) X
Group By Club_Id
```

```
WITH CTE AS(
select *,
ISNULL(case when edu LIKE '%MM%' then 0.5 end,0) AS MM,
ISNULL(case when edu LIKE '%CI%' then 0.5 end,0) AS CI,
ISNULL(case when edu LIKE '%CO%' then 0.5 end,0) AS CO,
ISNULL(case when edu LIKE '%CD%' then 1 end,0) AS CD,
ISNULL(case when edu LIKE '%CL%' then 1 end,0) AS CL,
ISNULL(case when edu LIKE '%CM%' then 1 end,0) AS CM from club )
,  CTEE AS(
SELECT CLUB_ID,
 MM+CO+CI+CD+CL+CM AS T
FROM CTE
GROUP BY CLUB_ID, mm, CO, CI, CD, CL, CM)
 SELECT CLUB_ID, SUM(T) AS TOTAL_POINTS FROM CTEE GROUP BY CLUB_ID
```

Problem Statement :- ITEM Table as shown below has two columns namely ItemName and TotalQuantity. Write a SQL query to duplicate the rows based on TotalQuantity in output table by adding two new columns ID and CatID

Input : ITEM Table

	ItemName	TotalQuantity
1	Apple	2
2	Orange	3

Desired Output :

	ID	ItemName	CatID	TotalQuantity
1	1	Apple	1	2
2	2	Apple	2	2
3	3	Orange	1	3
4	4	Orange	2	3
5	5	Orange	3	3

```
with recursive cte as (
select itemname,1 as catid, totalquantity from item
union all
select itemname ,catid+1,totalquantity from cte
where catid <totalquantity) # jase hi id =q hua ab recur= or run nhi hoga for that particular row and moves to next row from main table
select row_number() over(order by itemname,catid) as id, c.* from cte c
```

Problem Statement :- ABC Bank is trying to find their customers transaction mode. Customers are using different apps such as Gpay, PhonePe, Paytm etc along with offline transactions. Bank wants to know which mode/app is used for highest amount of transactions in each location.

Write a SQL query to find the app/mode and the count for highest amount of transactions in each location

Input Table :-

Customer

	Customer_id	Cus_name	Age	Gender	App
1	1	Amelia	23	Female	gpay
2	2	William	16	Male	phonepay
3	3	James	18	Male	paytm
4	4	David	24	Male	paytm
5	5	Ava	21	Female	gpay
6	6	Sophia	31	Female	paytm
7	7	Oliver	23	Male	gpay
8	8	Harry	29	Male	NULL
9	9	Issac	16	Male	gpay

Transaction_Tbl

	Loc_name	Loc_id	Cus_id	Amount_paid	Trans_id
1	Florida	100	1	78899	1000
2	Florida	100	2	55678	1001
3	Florida	100	3	27788	1002
4	Florida	100	4	65886	1003
5	Alaska	101	5	57757	1004
6	Alaska	101	6	34676	1005
7	Alaska	101	7	66837	1006
8	Alaska	101	8	77633	1007
9	Texas	102	9	98766	1008

Consider below points while fetching the record :-

- Columns to be fetched – App, Count
- Count – Number of times the app is used for the highest amount of transaction for each location.
- If the App column is NULL, it means the customer has paid the amount through Offline mode
- The first letter of the app name should be in uppercase letter and the rest followed by the lowercase
- The Count should be in descending order

Output Table

	App	Cnt
1	Gpay	2
2	Offline	1

```
with cte as (
select *,
 rank() Over(partition by Loc_id order by Amount_paid desc) as rnk
from Customer c
join Transaction_Tbls t
on c.Customer_id = t.Cus_id)
select
case when app is NULL then 'Offline'
else concat(upper(left(app,1)) , Lower(substring(app,2,length(app)))) end as Mode,
count(*) as cnt
from cte
where rnk = 1
 group by 1
 order by cnt desc;
```

```sql
With CTE_A As(
Select App,Loc_Name,Amount_paid,
Dense_Rank() OVER(partition by Loc_Name Order by Amount_paid desc) As Rnk
From Customer c inner join Transaction_Tbls t
On c.Customer_id = t.Cus_id
),
CTE_B As (
Select ISNULL(App,'Offline') As AppName, Count(*) As Cnt From CTE_A
where Rnk = 1
Group by App)
Select CONCAT(UPPER(Substring(AppName,1,1)),LOWER(Substring(AppName,2,len(AppName)))) As App, Cnt From CTE_B
Order by Cnt desc
```

Input Table

	Id	UserId	Item	CreatedAt	Revenue
1	1	109	milk	2020-03-03	123
2	2	103	bread	2020-03-29	862
3	3	128	bread	2020-03-04	112
4	4	128	biscuit	2020-03-24	160
5	5	100	banana	2020-03-18	599
6	6	103	milk	2020-03-31	290
7	7	102	bread	2020-03-25	325
8	8	109	bread	2020-03-22	432
9	9	101	milk	2020-03-01	449
10	10	100	milk	2020-03-29	410
11	11	129	milk	2020-03-02	771
12	12	104	biscuit	2020-03-31	957
13	13	110	bread	2020-03-13	210
14	14	128	milk	2020-03-28	498
15	15	109	bread	2020-03-02	362
16	16	110	bread	2020-03-13	262
17	17	105	bread	2020-03-21	562
18	18	101	milk	2020-03-26	740
19	19	100	banana	2020-03-13	175
20	20	105	banana	2020-03-05	815
21	21	129	milk	2020-03-02	489
22	22	105	banana	2020-03-09	972

Problem Statement :- Ecommerce company is trying to identify returning active users. A returning active user is a user that has made a second purchase within 7 days of any other of their purchases. Write a SQL query to display the list of UserId of these returning active users.

Output Table

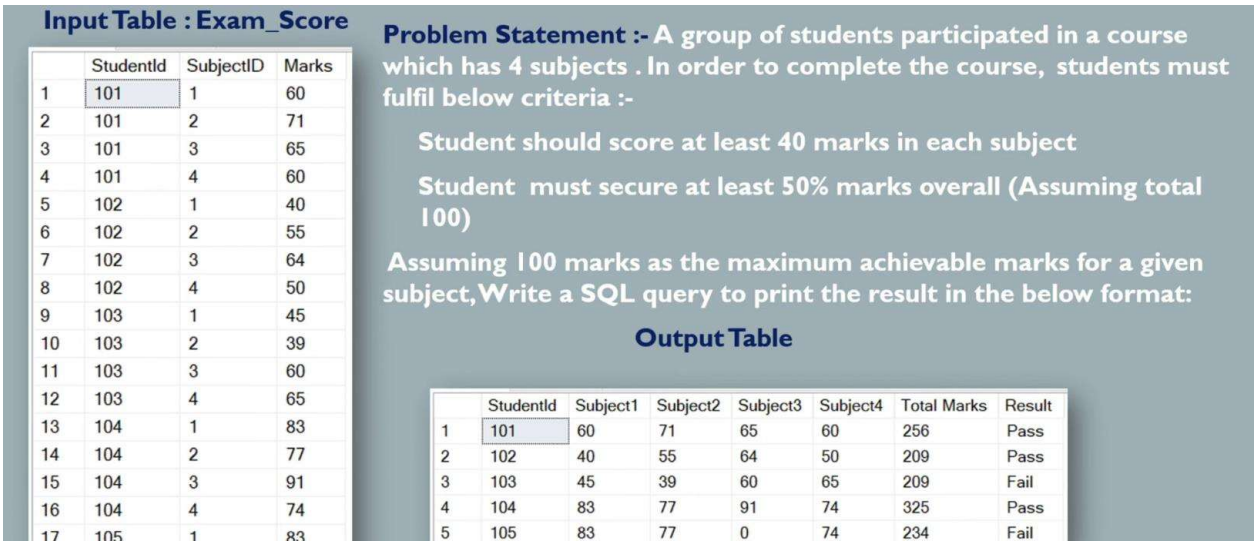
	UserID
1	100
2	103
3	105
4	109
5	110
6	128
7	129

```sql
Select distinct A.UserId From Transactions_Amazon  As A
inner join Transactions_Amazon As B
On A.UserId = B.UserId
where A.ID <> B.Id and Datediff(Day,B.CreatedAt,A.CreatedAt) Between 0 and 7
Order by A.UserId
```

139 %

Results | Messages

	Id	UserId	Item	CreatedAt	Revenue	Id	UserId	Item	CreatedAt	Revenue	(No column name)
1	5	100	banana	2020-03-18	599	19	100	banana	2020-03-13	175	5
2	19	100	banana	2020-03-13	175	24	100	bread	2020-03-07	410	6
3	6	103	milk	2020-03-31	290	2	103	bread	2020-03-29	862	2
4	22	105	banana	2020-03-09	972	20	105	banana	2020-03-05	815	4
5	1	109	milk	2020-03-03	123	15	109	bread	2020-03-02	362	1
6	16	110	bread	2020-03-13	262	13	110	bread	2020-03-13	210	0
7	13	110	bread	2020-03-13	210	16	110	bread	2020-03-13	262	0
8	14	128	milk	2020-03-28	498	4	128	biscuit	2020-03-24	160	4
9	11	129	milk	2020-03-02	771	21	129	milk	2020-03-02	489	0
10	21	129	milk	2020-03-02	489	11	129	milk	2020-03-02	771	0




```
with cte as(
select studentid,
 sum(case when subjectid = 1 then marks else 0 end) as sub_1,
sum(case when subjectid = 2 then marks else 0 end) as sub_2,
 sum(case when subjectid = 3 then marks else 0 end) as sub_3,
sum(case when subjectid = 4 then marks else 0 end) as sub_4
from exam_score
group by studentid)
select *,(sub_1+sub_2+sub_3+sub_4) as tot_marks,
case when sub_1 < 40 or sub_2<40 or sub_3<40 or sub_4<40 then 'Fail' else 'Pass' end as result
from cte
```

Problem Statement :- For the 2021 academic year, students have appeared in the SSC exam. Write a SQL query to calculate the percentage of results using the best of the five rule i.e. You must take the top five grades for each student and calculate the percentage.

Table : SSC_Exam :-

	Id	English	Maths	Science	Geography	History	Sanskrit
1	1	85	99	92	86	86	99
2	2	81	82	83	86	95	96
3	3	76	55	76	76	56	76
4	4	84	84	84	84	84	84
5	5	83	99	45	88	75	90

Output :

	ID	English	Maths	Science	Geography	Sanskrit	History	Percentage
1	1	85	99	92	86	99	86	92.4
2	2	81	82	83	86	96	95	88.4
3	3	76	55	76	76	76	56	72
4	4	84	84	84	84	84	84	84

```
With CTE_1 As (
 Select * From (
   Select Id,English,Maths,Science,Geography,History,Sanskrit From SSC_Exam
   ) As T
Unpivot
(
Marks
For Subject in(English,Maths,Science,Geography,History,Sanskrit)
) unpvt
)
,
CTE_2 As (
Select id,Marks,ROW_NUMBER() OVER (Partition by id Order by Marks desc) As Rnk From CTE_1
),
CTE_3 As (
Select id, Cast(sum(Marks)/500.0 As Float)*100 As Percentage From CTE_2 where Rnk <=5 Group by Id
)
Select * From CTE_3 As C INNER JOIN SSC_Exam As S ON C.ID=S.ID
```

User Purchase Platform Problem

Problem Statement :-

The Spending table keeps the logs of the spendings history of users that make purchases from an online shopping website which has a desktop and a mobile application.

Write an SQL query to find the total number of users and the total amount spent using mobile only, desktop only and both mobile and desktop together for each date.

Table : Spending

	User_id	Spend_date	Platform	Amount
1	1	2019-07-01	Mobile	100
2	1	2019-07-01	Desktop	100
3	2	2019-07-01	Mobile	100
4	2	2019-07-02	Mobile	100
5	3	2019-07-01	Desktop	100
6	3	2019-07-02	Desktop	100

Output :

	Spend_date	Platform	Total_Amount	Total_users
1	2019-07-01	Mobile	100	1
2	2019-07-01	Desktop	100	1
3	2019-07-01	Both	200	1
4	2019-07-02	Mobile	100	1
5	2019-07-02	Desktop	100	1
6	2019-07-02	Both	0	0

```sql
-- Select * From Spending;

With CTE_A As (
-- logic for only desktop or only mobile
Select Spend_date,User_id,MIN(Platform) As Platform,sum(Amount) As Amount  From Spending group by Spend_date,User_id Having
count(distinct Platform) =1
-- Both
UNION ALL
Select Spend_date,User_id,'Both' As Platform,sum(Amount) As Amount  From Spending group by Spend_date,User_id Having
count(distinct Platform) =2
)
```

	Spend_date	User_id	Platform	Amount
1	2019-07-01	2	Mobile	100
2	2019-07-02	2	Mobile	100
3	2019-07-01	3	Desktop	100
4	2019-07-02	3	Desktop	100
5	2019-07-01	1	Both	200

```sql
Select Spend_date,Platform,Sum(Amount) As Total_Amount, count(distinct User_id) As Total_UserID
From CTE_A
Group by Spend_date,Platform
Order by Spend_date,Platform
```

Number of Calls between Two Persons

Problem Statement :-

Calls Table has three columns namely From_Id, To_Id and Duration . It contains duration of calls between From_Id and To_Id. Write a SQL query to report the number of calls and the total call duration between each pair of distinct persons (Person1, Person2) where Person1 < Person2. Return the result as shown in Output Table.

Table : Calls

	From_Id	To_Id	Duration
1	1	2	59
2	2	1	11
3	1	3	20
4	3	4	100
5	3	4	200
6	3	4	200

Output :

	Person1	Person2	Call_Count	Total_Duration
1	1	2	2	70
2	1	3	1	20
3	3	4	4	999

```sql
With CTE_Calls As (
Select
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,
Duration
From Calls
)
Select Person1,Person2,
Count(Duration) As 'Call_Count',
Sum(Duration) As 'Total_Duration'
From CTE_Calls
Group by Person1,
```

Problem Statement :-

Write a SQL query to output the names of those students whose best friends got higher salary package than Student.

Input Tables :

Students_Tbl

	Id	Student_Name
1	1	Mark
2	2	David
3	3	John
4	4	Albert

Friends_Tbl

	Id	Friend_Id
1	1	2
2	2	3
3	3	4
4	4	1

Package_Tbl

	Id	Salary
1	1	18
2	2	12

Output Table :

	Student
1	David
2	John
3	Albert

```
SELECT s.student_name
FROM Students_Tbl s
JOIN Friends_Tbl f ON s.id = f.id
JOIN Package_Tbl p_student ON s.id = p_student.id
JOIN Package_Tbl p_friend ON f.friend_id = p_friend.id
WHERE p_friend.salary > p_student.salary;
```

Input Table : facebook_web_log

	Userid	Time_stamp	Actions
1	0	2019-04-25 13:30:15.000	page_load
2	0	2019-04-25 13:30:18.000	page_load
3	0	2019-04-25 13:30:40.000	scroll_down
4	0	2019-04-25 13:30:45.000	scroll_up
5	0	2019-04-25 13:31:10.000	scroll_down
6	0	2019-04-25 13:31:25.000	scroll_down
7	0	2019-04-25 13:31:40.000	page_exit
8	1	2019-04-25 13:40:00.000	page_load
9	1	2019-04-25 13:40:10.000	scroll_down
10	1	2019-04-25 13:40:15.000	scroll_down
11	1	2019-04-25 13:40:20.000	scroll_down
12	1	2019-04-25 13:40:25.000	scroll_down
13	1	2019-04-25 13:40:30.000	scroll_down
14	1	2019-04-25 13:40:35.000	page_exit
15	2	2019-04-25 13:41:21.000	page_load
16	2	2019-04-25 13:41:30.000	scroll_down
17	2	2019-04-25 13:41:35.000	scroll_down
18	2	2019-04-25 13:41:40.000	scroll_up
19	1	2019-04-26 11:15:00.000	page_load
20	1	2019-04-26 11:15:10.000	scroll_down
21	1	2019-04-26 11:15:20.000	scroll_down
22	1	2019-04-26 11:15:25.000	scroll_up
23	1	2019-04-26 11:15:35.000	page_exit
24	0	2019-04-28 14:30:15.000	page_load
25	0	2019-04-28 14:30:10.000	page_load
26	0	2019-04-28 13:30:40.000	scroll_down
27	0	2019-04-28 15:31:40.000	page_exit

Problem Statement (Asked in FACEBOOK / META) :-

Write a SQL query to calculate each user's average session time.

A session is defined as the time difference between a page_load and page_exit. 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. Output the user_id and their average session time

Output Table :

	Userid	AVG_Session
1	0	1883.5
2	1	35.0

ITJunction4All

```
  SELECT
      userid
      , session_date  = convert(date, time_stamp)
      , session_start = max(case when actions = 'page_load' then time_stamp end)
      , session_end   = min(case when actions = 'page_exit' then time_stamp end)
  FROM        facebook_web_log
  GROUP BY userid, convert(date, time_stamp)
)
SELECT
    userid
     , avg_session_time = CAST(AVG(1.*datediff(second, session_start, session_end)) as NUMERIC(10,1))
FROM    cte
WHERE session_start is not null and session_end is not null
GROUP BY userid
```

Problem Statement :-

Write a SQL query to remove all reversed number pairs from below given table. Keep only one random pair .

Assumption:

1. There will not be same value for both A and B column.

2. There will not be same pair of numbers repeating in this table.

Input Tables : Reverse_duplicates :

	A	B
1	1	2
2	3	2
3	2	4
4	2	1
5	5	6

Output Table :

	A	B
1	1	2
2	3	2
3	2	4

```sql
-- 1st Approach( Self Join)
Select t1.A,t1.B From Reverse_duplicates As t1
left join Reverse_duplicates As t2 On t1.A = t2.B And t1.B = t2.A
where t1.A < t2.A OR (t2.A is null or t2.B is null);

-- 2nd Approach ( Correlated Subquery and Not Exist)
Select * From Reverse_duplicates As t1 where not exists (Select * From Reverse_duplicates As t2
where t1.A = t2.B And t1.B = t2.A And t1.A > t2.A );
```

Problem Statement (Asked in WALMART) :-

Write a SQL query to find the top 3 products that are most frequently bought together (purchased in the same transaction).

Output the name of product #1, name of product #2 and number of combinations in descending order.

Input Tables :

	transaction_id	product_id	users_id	transaction_date
1	523152	222	746	2022-03-06 12:00:00.000
2	523152	444	746	2022-03-06 12:00:00.000
3	256234	222	311	2022-03-07 12:00:00.000
4	256234	333	311	2022-03-07 12:00:00.000
5	231574	111	234	2022-03-01 12:00:00.000
6	231574	444	234	2022-03-01 12:00:00.000
7	231574	222	234	2022-03-01 12:00:00.000
8	141415	333	235	2022-03-02 12:00:00.000
9	137124	111	125	2022-03-05 12:00:00.000
10	137124	444	125	2022-03-05 12:00:00.000

	product_id	product_name
1	111	apple
2	222	soya milk
3	333	instant oatmeal
4	444	banana
5	555	chia seed

Output Table :

	product1	product2	combo_num
1	banana	soya milk	2
2	apple	banana	2
3	instant oatmeal	soya milk	1

Why we use `A.product_name < B.product_name`

When we do **not** add the condition

`A.product_name < B.product_name`

the query produces **all possible pair combinations**, including self-pairs and duplicate reverse pairs, such as:

```
apple              apple
banana             apple
soya milk           apple
apple              banana
banana             banana
soya milk           banana
instant oatmeal    instant oatmeal
soya milk           instant oatmeal
apple              soya milk
banana             soya milk
instant oatmeal    soya milk
soya milk           soya milk
```

1. Combinations that do not make sense

We do **not** require pairs where the product is matched with itself:

```
(apple, apple)
(banana, banana)
(instant oatmeal, instant oatmeal)
(soya milk, soya milk)
```

These pairs add no analytical value and should be removed.

2. Duplicate combinations with reversed order

Some combinations represent the **same logical pair**, just in reverse order:

```
(banana, apple)  and  (apple, banana)
(banana, soya milk)  and  (soya milk, banana)
(instant oatmeal, soya milk)  and  (soya milk, instant oatmeal)
```

From each of these, we need **only one** combination.

How `A.product_name < B.product_name` solves both problems

- `product_name` is a **string (VARCHAR)** datatype.
- SQL compares strings **lexicographically**:
  - First character's ASCII value is compared
  - If equal, the next character is compared, and so on

Effect of the condition:

`A.product_name < B.product_name`

- **Self-pairs are removed**

Example: `(apple, apple)`

Since both values are equal, the < condition fails.

- **Duplicate reverse pairs are removed**

Example:

`(banana, apple)` ❌ → `fails`

`(apple, banana)` ✅ → `passes`

Only **one ordered pair** survives.

Final Outcome

Using `A.product_name < B.product_name` ensures that:

- Self-combinations are excluded
- Duplicate reversed combinations are eliminated
- Each product pair appears **once and only once**

This results in **clean, meaningful, and non-redundant product combinations**.

Problem Statement (Asked in FACEBOOK) :-

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).

An active user is a user who has user action ('sign-in', 'like', or 'comment') in the current month and last month.

Input Tables : user_actions

	users_id	event_id	event_type	event_date
1	445	7765	sign-in	2022-05-31 12:00:00.000
2	445	3634	like	2022-06-05 12:00:00.000
3	648	3124	like	2022-06-18 12:00:00.000
4	648	2725	sign-in	2022-06-22 12:00:00.000
5	648	8568	comment	2022-07-03 12:00:00.000
6	445	4363	sign-in	2022-07-05 12:00:00.000
7	445	2425	like	2022-07-06 12:00:00.000
8	445	2484	like	2022-07-22 12:00:00.000
9	648	1423	sign-in	2022-07-26 12:00:00.000
10	445	5235	comment	2022-07-29 12:00:00.000

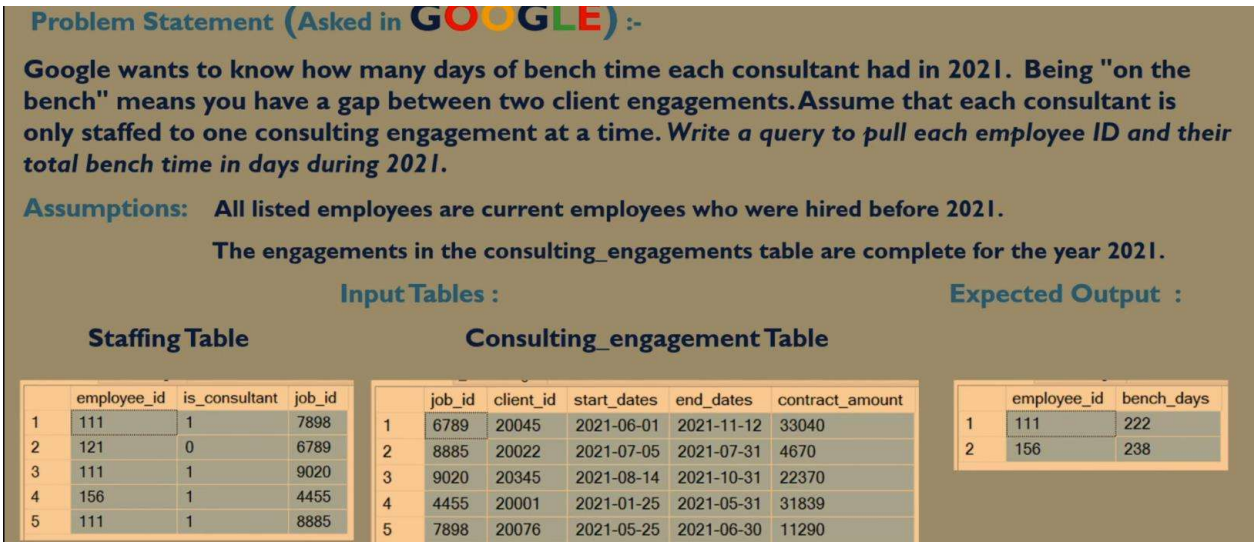
Output Table :

	months	monthly_active_users
1	7	2

```
With CTE_Users As (
Select users_id,MONTH(event_Date) As Months From user_actions where event_type in ('sign-in','like','comment')
group by users_id , MONTH(event_Date)
),
CTE_Active_Users As (
Select A.users_id,B.Months, A.Months As Months_1 From CTE_Users As A inner join CTE_Users As B ON A.users_id =B.users_id and B
and A.Months =B.Months -1
)
Select Months,count(users_id) As monthly_active_users From CTE_Active_Users
Group by Months
```

```
WITH monthly_activity AS (
  SELECT
    users_id,
    DATE_FORMAT(event_date, '%Y-%m-01') AS month_start
  FROM user_actions
  WHERE event_type IN ('sign-in', 'like', 'comment')
  GROUP BY users_id, DATE_FORMAT(event_date, '%Y-%m-01')
),
retained_users AS (
  SELECT
    users_id,
    month_start,
    LAG(month_start) OVER (
      PARTITION BY users_id
      ORDER BY month_start
    ) AS prev_month
  FROM monthly_activity
)
```

```
SELECT
  MONTH(month_start) AS month,
  COUNT(DISTINCT users_id) AS monthly_active_users
FROM retained_users
WHERE prev_month = DATE_SUB(month_start, INTERVAL 1 MONTH)
GROUP BY MONTH(month_start)
ORDER BY month;
```



	employee_id	is_consultant	job_id
1	111	1	7898
2	121	0	6789
3	111	1	9020
4	156	1	4455
5	111	1	8885

	job_id	client_id	start_dates	end_dates	contract_amount
1	6789	20045	2021-06-01	2021-11-12	33040
2	8885	20022	2021-07-05	2021-07-31	4670
3	9020	20345	2021-08-14	2021-10-31	22370
4	4455	20001	2021-01-25	2021-05-31	31839
5	7898	20076	2021-05-25	2021-06-30	11290

	employee_id	bench_days
1	111	222
2	156	238

```
WITH emp_engagements AS (
  SELECT
    s.employee_id,
    c.start_dates,
    c.end_dates,
    LAG(c.end_dates) OVER (
      PARTITION BY s.employee_id
      ORDER BY c.start_dates
    ) AS prev_end_date
  FROM staffing s
  JOIN consulting_engagements c
    ON s.job_id = c.job_id
  WHERE s.is_consultant = 1
),
bench_calc AS (
  SELECT
    employee_id,
    CASE
      WHEN prev_end_date IS NOT NULL
```

```
            AND DATEDIFF(start_dates, prev_end_date) > 1
        THEN DATEDIFF(start_dates, prev_end_date)  - 1
        ELSE 0
      END AS bench_days
  FROM emp_engagements
)

SELECT
  employee_id,
  SUM(bench_days) AS bench_days
FROM bench_calc
GROUP BY employee_id
ORDER BY employee_id;
```

Problem Statement (Asked in GOOGLE) :-

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.

Input Tables : `search_frequency`

	searches	num_users
1	1	2
2	4	1
3	2	2
4	3	3
5	6	1
6	5	3

Expected Output :

	Median
1	3.5

SQLQuery59.sql - su...(SUNIL\Sunil (76))*

```
4   --When No of Data points are odd Eg. 1,1,2,3,4  = 2
5   --When No of Data points are even Eg. 1,1,2,3 -> 1+2/2 =1.5
6   -- percentile_cont
7
8   With CTE_REC_Median As (
9   Select searches, num_users, 1 As temp From search_frequency -- Base query
10  UNION ALL
11  Select  searches, num_users,temp + 1 From CTE_REC_Median where temp + 1 <= num_users
12  )
13  Select ROUND(percentile_cont(.5) within group ( order by searches) over()),1) As Median From CTE_REC_Median order by searches,tem
14
15
16
17
18
19
20
21
```

150 %

Results | Messages

	searches	num_users	temp
1	1	2	1
2	1	2	2
3	2	2	1
4	2	2	2
5	3	3	1
6	3	3	2

Query executed successfully. sunil (15.0 RTM) SUNIL\Sunil (76) master 00:00:00 14 rows

Problem Statement :-

Given below is Stocks Table which consists of DateKey, StockName and Price.

Write a SQL query to derive another column MarketPrice where it is going to forward fill NULL values with the last Non NULL value

Input Tables : Stocks

	DateKey	StocksName	Price
1	2023-01-01	Infosys	1400
2	2023-01-02	Infosys	NULL
3	2023-01-03	Infosys	1450
4	2023-01-04	Infosys	NULL
5	2023-01-05	Infosys	NULL
6	2023-01-05	Infosys	NULL
7	2023-01-01	Reliance	2300
8	2023-01-02	Reliance	NULL
9	2023-01-03	Reliance	NULL
10	2023-01-04	Reliance	2375
11	2023-01-05	Reliance	2400

Expected Output :

	DateKey	StocksName	Price	MarketPrice
1	2023-01-01	Infosys	1400	1400
2	2023-01-02	Infosys	NULL	1400
3	2023-01-03	Infosys	1450	1450
4	2023-01-04	Infosys	NULL	1450
5	2023-01-05	Infosys	NULL	1450
6	2023-01-05	Infosys	NULL	1450
7	2023-01-01	Reliance	2300	2300
8	2023-01-02	Reliance	NULL	2300
9	2023-01-03	Reliance	NULL	2300
10	2023-01-04	Reliance	2375	2375
11	2023-01-05	Reliance	2400	2400

SQLQuery1.sql - sun...(SUNIL\Sunil (66))*

```
1  -- Solution
2  -- Select * From Stocks
3  With CTE_Rank As (
4  Select DateKey,StocksName,Price,
5  COUNT(Price) OVER (Partition by StocksName Order by DateKey) As Rnk
6  From Stocks
7  )
8  Select DateKey,StocksName,Rnk,Price,
9  FIRST_VALUE(Price) OVER(Partition by StocksName,Rnk Order by DateKey) As MarketPrice
10 From CTE_Rank
```

150 %

Results | Messages

	DateKey	StocksName	Price	MarketPrice
1	2023-01-01	Infosys	1400	1400
2	2023-01-02	Infosys	NULL	1400
3	2023-01-03	Infosys	1450	1450
4	2023-01-04	Infosys	NULL	1450
5	2023-01-05	Infosys	NULL	1450
6	2023-01-05	Infosys	NULL	1450
7	2023-01-01	Reliance	2300	2300
8	2023-01-02	Reliance	NULL	2300
9	2023-01-03	Reliance	NULL	2300
10	2023-01-04	Reliance	2375	2375
11	2023-01-05	Reliance	2400	2400
12	2023-01-06	Reliance	NULL	2400

Problem Statement :-

Given below is Device Table which consists of Device_id and Locations.

Write a SQL query to get the output as shown below :-

Input Tables : `Device`

	Device_id	Locations
1	12	Bangalore
2	12	Bangalore
3	12	Bangalore
4	12	Bangalore
5	12	Hosur
6	12	Hosur
7	13	Hyderabad
8	13	Hyderabad
9	13	Secunderabad
10	13	Secunderabad

Expected Output :

	Device_id	no_of_location	max_signal_location	no_of_signals
1	12	2	Bangalore	6
2	13	2	Secunderabad	5

```
with cte as (
 select device_id,
count(DISTINCT locations) as no_of_location,   locations,
count(locations) as cnt_siginal_location,
count(locations) as no_of_signal,
dense_rank() over(partition by device_id order by count(locations) desc) as rn
from Device
group by device_id, locations )
select device_id,
sum(no_of_location) total_location, locations,
sum(no_of_signal) as total_signal
from cte
group by device_id
```

Problem Statement :- Players Table indicates the group of each player. Match table contains details of the match played. The winner in each group is the player who scored the maximum total points within the group. In the case of a tie, the lowest player_id wins.

You can assume that, in each match, players belong to the same group.

Write an SQL query to find the winner in each group.

Input Tables : Players & Matches

Expected Output :

	player_id	group_id
1	15	1
2	25	1
3	30	1
4	45	1
5	10	2
6	35	2
7	50	2
8	20	3

	match_id	first_player	second_player	first_score	second_score
1	1	15	45	3	0
2	2	30	25	1	2
3	3	30	15	2	0
4	4	40	20	5	2
5	5	35	50	1	1

	player_id	group_id
1	15	1
2	35	2
3	40	3

```sql
With CTE_All_Players As (
Select first_player As Player, first_score As Scores From Matches
UNION ALL
Select second_player As Player, second_score As Scores From Matches
),
CTE_Total_Points As (
Select Player, SUM(Scores) As TotScores From CTE_All_Players
Group by Player
)
Select * From CTE_Total_Points
```

SQLQuery62.sql - su...(SUNIL\Sunil (64))*

```sql
Select Player, SUM(Scores) As TotScores From CTE_All_Players
Group by Player
),
CTE_Rnk As (

Select player_id,TotScores,p.group_id,
ROW_NUMBER() OVER(Partition by group_id Order By TotScores desc,player_id asc ) As Rnk
From CTE_Total_Points As C INNER JOIN Players As P ON C.Player = P.player_id
)
Select player_id,group_id From CTE_Rnk where Rnk=1
```

150 %

Results | Messages

	player_id	TotScores	group_id	Rnk
1	15	3	1	1
2	30	3	1	2
3	25	2	1	3
4	45	0	1	4
5	35	1	2	1
6	50	1	2	2
7	40	5	3	1
8	20	2	3	2

Problem Statement :- **Staff Table has two columns namely Id and Name.**

Write an SQL query to get the expected output as shown below :-

Input Tables : **Staff_Tbl**

	Id	Name
1	1	Staff1
2	2	Staff2
3	3	Staff3
4	4	Staff4
5	5	Staff5
6	6	Staff6
7	7	Staff7

Expected Output :

	Results
1	1 Staff1, 2 Staff2
2	3 Staff3, 4 Staff4
3	5 Staff5, 6 Staff6
4	7 Staff7, 8 Staff8

```
1   -- Solution----
2   Select * From Staff_Tbl
3   --concat two columns with space in between
4   --merge two-two records with with comma and space as a separator
5
6   With CTE_Staff As (
7   Select CONCAT(Id,' ', Name) As Name,
8   NTILE(4) OVER (Order by ID ) As Grp
9   From Staff_Tbl )
10  Select STRING_AGG(Name,', ') As Results From CTE_Staff
11  Group by Grp
12
```

150 %

Results | Messages

	Results
1	1 Staff1, 2 Staff2
2	3 Staff3, 4 Staff4
3	5 Staff5, 6 Staff6
4	7 Staff7, 8 Staff8

```
with new as (
select ceiling ( cast (id as decimal) / 2 ) as id,
Name from Staff_Tbl)
select id, STRING_AGG(Name,' ,') as grouped from new group by id
```

Problem Statement :- **There are two tables namely Job_Positions and Job_Employees**

Write an SQL query to get the expected output as shown in next slide. Basically it should display not filled posts as 'Vacant' along with the details of the employee:-

Input Tables :

Job_Positions

	id	title	groups	levels	payscale	totalpost
1	1	General manager	A	I-15	10000	1
2	2	Manager	B	I-14	9000	5
3	3	Asst. Manager	C	I-13	8000	10

Job_Employees

	id	name	position_id
1	1	John Smith	1
2	2	Jane Doe	2
3	3	Michael Brown	2
4	4	Emily Johnson	2
5	5	William Lee	3
6	6	Jessica Clark	3
7	7	Christopher Harris	3
8	8	Olivia Wilson	3
9	9	Daniel Martinez	3
10	10	Sophia Miller	3

Expected Output :

	id	Title	Groups	levels	payscale	Employee_Name
1	1	General manager	A	I-15	10000	John Smith
2	2	Manager	B	I-14	9000	Jane Doe
3	2	Manager	B	I-14	9000	Michael Brown
4	2	Manager	B	I-14	9000	Emily Johnson
5	2	Manager	B	I-14	9000	Vacant
6	2	Manager	B	I-14	9000	Vacant
7	3	Asst. Manager	C	I-13	8000	William Lee
8	3	Asst. Manager	C	I-13	8000	Jessica Clark
9	3	Asst. Manager	C	I-13	8000	Christopher Harris
10	3	Asst. Manager	C	I-13	8000	Olivia Wilson
11	3	Asst. Manager	C	I-13	8000	Daniel Martinez
12	3	Asst. Manager	C	I-13	8000	Sophia Miller
13	3	Asst. Manager	C	I-13	8000	Vacant
14	3	Asst. Manager	C	I-13	8000	Vacant

Problem Statement :- The relationship between the LIFT and LIFT_PASSENGERS Table is such that multiple passengers can attempt to enter the same lift, but the total weight of the passengers in a lift cannot exceed the lift's capacity.

Write an SQL query that produces a comma separated list of passengers who can be accommodated in each lift without exceeding the lift's capacity. The passengers in the list should be ordered by their weight in increasing order

Input Tables :

Lift_Passengers:

	Passenger_Name	Weight_Kg	Lift_Id
1	Mark	85	1
2	Antony	73	1
3	David	95	1
4	Mary	80	1
5	John	83	2
6	Robert	77	2
7	Maria	73	2
8	Susan	85	2

Lift

	Id	Capacity_Kg
1	1	300
2	2	350

OUTPUT Table

	Lift_Id	Passengers
1	1	Antony, Mary, Mark
2	2	Maria, Robert, John, Susan

```
2
3  Select * From Lift_Passengers
4  Select * From lift
5
6  With CTE_Passengers As (
7  Select *,
8  SUM(weight_kg) OVER(Partition by Id Order by weight_kg) As Running_Wt,
9  Case when SUM(weight_kg) OVER(Partition by Id Order by weight_kg) <= Capacity_Kg Then 1 Else 0 End As Flag
10 From Lift_Passengers As LP INNER JOIN lift As L ON LP.Lift_Id= L.Id
11 )
12 Select Id, STRING_AGG(Passenger_Name,', ') within group (Order by Weight_Kg) As Passengers From CTE_Passengers
13 where Flag=1
14 Group by Id
15
```

36 %

Results | Messages

	Passenger_Name	Weight_Kg	Lift_Id	Id	Capacity_Kg	Running_Wt	Flag
1	Antony	73	1	1	300	73	1
2	Mary	80	1	1	300	153	1
3	Mark	85	1	1	300	238	1
4	Maria	73	2	2	350	73	1
5	Robert	77	2	2	350	150	1
6	John	83	2	2	350	233	1
7	Susan	85	2	2	350	318	1

Problem Statement :- There is a table called User_Transaction which contains information about XYZ company users transactions for different products.

Write a query to calculate the Year-on-Year growth rate for the total spend of each product.

Input Tables :

User_Transaction :

	transaction_id	product_id	spend	transaction_date
1	1	101	50.00	2021-01-15 00:00:00.000
2	2	101	75.00	2022-02-20 00:00:00.000
3	3	101	100.00	2023-03-10 00:00:00.000
4	4	101	200.00	2023-01-12 00:00:00.000
5	5	101	150.00	2024-01-10 00:00:00.000
6	6	102	200.00	2022-02-25 00:00:00.000
7	7	102	250.00	2023-05-30 00:00:00.000
8	8	102	300.00	2024-02-14 00:00:00.000
9	9	103	350.00	2022-06-18 00:00:00.000
10	10	103	300.00	2023-08-20 00:00:00.000
11	11	104	450.00	2022-09-25 00:00:00.000
12	12	104	500.00	2022-10-30 00:00:00.000

OUTPUT Table

	Years	product_id	curr_year_spend	Prev_year_spend	YOY_Growth_Rate
1	2021	101	50.00	NULL	NULL
2	2022	101	75.00	50.00	50.000000
3	2023	101	300.00	75.00	300.000000
4	2024	101	150.00	300.00	-50.000000
5	2022	102	200.00	NULL	NULL
6	2023	102	250.00	200.00	25.000000
7	2024	102	300.00	250.00	20.000000
8	2022	103	350.00	NULL	NULL
9	2023	103	300.00	350.00	-14.290000
10	2022	104	950.00	NULL	NULL

```
7  With CTE_Current As (
8  Select YEAR(transaction_date) As Years,product_id,
9  SUM(spend) As curr_year_spend
10 From user_transaction
11 Group by product_id,YEAR(transaction_date) --order by product_id
12 ),
13 CTE_Previous As (
14 Select Years, product_id, curr_year_spend,
15 LAG(curr_year_spend) OVER (Partition by product_id Order by Years) As Prev_year_spend
16 From CTE_Current)
17 Select Years, product_id, curr_year_spend,Prev_year_spend,
```

136 %

Results | Messages

	Years	product_id	curr_year_spend	Prev_year_spend	YOY_Growth_Rate
1	2021	101	50.00	NULL	NULL
2	2022	101	75.00	50.00	50.000000
3	2023	101	300.00	75.00	300.000000
4	2024	101	150.00	300.00	-50.000000
5	2022	102	200.00	NULL	NULL
6	2023	102	250.00	200.00	25.000000
7	2024	102	300.00	250.00	20.000000
8	2022	103	350.00	NULL	NULL
9	2023	103	300.00	350.00	-14.290000
10	2022	104	950.00	NULL	NULL

Problem Statement :- **There is a table called Stationary which contains columns Date_Sold, Product and Amount_Sold. Write a query to report the difference between the number of Notebook and Pen sold each day and corresponding status as shown below.**

Clause	Status
If Notebook is sold more than Pen	Notebook is sold more
If Pen is sold more than Notebook	Pen is sold more
If neither Notebook nor Pen is sold	Nothing is sold
If both Notebook and Pen are sold equally	Both Notebook and Pen sold equally

Stationary :

	Date_Sold	Product	Amount_Sold
1	2022-06-01	Notebook	6
2	2022-06-01	Pen	18
3	2022-06-02	Pen	14
4	2022-06-02	Notebook	15
5	2022-06-03	Pen	15
6	2022-06-03	Notebook	6
7	2022-06-04	Notebook	16
8	2022-06-04	Pen	NULL
9	2022-06-05	Notebook	NULL
10	2022-06-05	Pen	NULL
11	2022-06-06	Notebook	16
12	2022-06-06	Pen	16
13	2022-06-07	Notebook	NULL
14	2022-06-07	Pen	NULL
15	2022-06-08	Notebook	NULL
16	2022-06-08	Pen	10

OUTPUT Table :

	Date_Sold	Diffference	Status
1	2022-06-01	12	Pen is sold more
2	2022-06-02	1	Notebook is sold more
3	2022-06-03	9	Pen is sold more
4	2022-06-04	16	Notebook is sold more
5	2022-06-05	0	Nothing is sold
6	2022-06-06	0	Both Notebook and Pen sold equally
7	2022-06-07	0	Nothing is sold
8	2022-06-08	10	Pen is sold more
9	2022-06-09	0	Nothing is sold

```
WITH cte AS(
SELECT
 DATE_SOLD
,SUM(CASE WHEN PRODUCT = 'Notebook' THEN AMOUNT_SOLD ELSE 0 END) AS notebook
,SUM(CASE WHEN PRODUCT = 'Pen' THEN AMOUNT_SOLD ELSE 0 END ) AS PEN
FROM Stationery
GROUP BY DATE_SOLD
)
SELECT
DATE_SOLD
,CASE WHEN NOTEBOOK > PEN THEN NOTEBOOK - PEN
    WHEN NOTEBOOK < PEN THEN PEN - NOTEBOOK
ELSE 0
END AS difference
,CASE WHEN NOTEBOOK > PEN THEN 'Notebook is sold more'
    WHEN PEN > NOTEBOOK THEN 'Pen is sold more'
    WHEN PEN  = 0 AND NOTEBOOK = 0 THEN 'Nothing to sold'
    WHEN PEN  = NOTEBOOK   THEN 'Both notebook and pen sold equally'
END AS Status
FROM cte
ORDER BY 1;
```

Problem Statement :-

A vaccine is administered in two doses. It is best if the doses are given between 48 and 72 days apart, inclusive. Write a sql query to return the percentage of beneficiaries who received both doses within the recommended time period, rounded to the nearest integer. Dose Table has columns namely Dose_id, Beneficiary_id, Dose_type and Vaccination_Date as shown below:

Input Table :

Doses

	Dose_id	Beneficiary_id	Dose_type	Vaccination_date
1	2193	750	first	2021-05-15
2	2194	750	second	2021-07-05
3	2195	751	first	2021-06-01
4	2196	751	second	2021-07-31
5	2197	752	first	2021-06-10
6	2198	752	second	2021-07-30
7	2199	753	first	2021-06-15
8	2200	753	second	2021-09-01
9	2201	754	first	2021-04-18
10	2202	754	second	2021-06-10

OUTPUT Table :

	percentage_within_recommended_period
1	67

```
WITH CTE AS (
SELECT Vaccination_date AS first_dose,
LAG(Vaccination_date) OVER (PARTITION BY Beneficiary_id ORDER BY Dose_type) AS second_dose
FROM Doses )
SELECT ROUND(CAST(COUNT(CASE WHEN DATEDIFF(DAY, second_dose, first_dose) BETWEEN 48
AND 72 THEN 1 END) * 100.0 / COUNT(*) AS DECIMAL(4,2)), 2) AS
percentage_Within_recommended_period
FROM CTE
WHERE second_dose IS NOT NULL;
```

Problem Statement :- **You are working with a table called Orders that tracks customer orders with their order dates and amounts. Write a query to find each customer's latest order amount along with the amount of the second latest order.**

Your output should be like customer_id, lastest_order_amount, second_lastest_order_amount

Input Table :

Orders

	order_id	customer_id	order_date	order_amount
1	1	101	2024-01-10	150.00
2	2	101	2024-02-15	200.00
3	3	101	2024-03-20	180.00
4	4	102	2024-01-12	200.00
5	5	102	2024-02-25	250.00
6	6	102	2024-03-10	320.00
7	7	103	2024-01-25	400.00
8	8	103	2024-02-15	420.00

OUTPUT Table :

	customer_id	latest_order_amount	second_latest_order_amount
1	101	180.00	200.00
2	102	320.00	250.00
3	103	420.00	400.00

```sql

-- You are working with a table called orders (order_id, customer_id, order_date, order_amount) that tracks customer orders with their order dates
-- Write a query to find each customer's latest order amount along with the amount of the second latest order.
-- Your output should be like - customer_id, lastest_order_amount, second_lastest_order_amount
-- Solution
Select * From Orders;
With CTE_Orders As (
Select customer_id,order_date,order_amount As lastest_order_amount,
LEAD(order_amount) OVER(Partition by customer_id Order by order_date desc) As second_lastest_order_amount,
ROW_NUMBER() OVER(Partition by customer_id Order by order_date desc) As Rnk
From Orders
)
Select customer_id, lastest_order_amount, second_lastest_order_amount
From CTE_Orders
WHERE Rnk=1

```