Tech TFQ Youtuber/https://techtfq.com/blog/learn-how-to-write-sql-queries-practice-complex-sql-queries

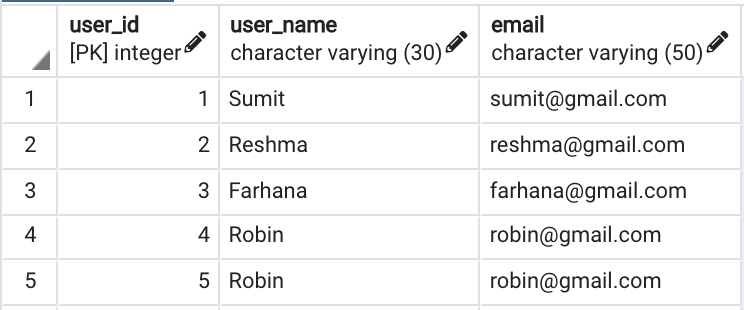
1. **Write a SQL Query to fetch all the duplicate records in a table.**

**Table Name**: TEMP\_USERS

**Note**: Record is considered duplicate if a user name is present more than once.

**Approach**: Partition the data based on user name and then give a row number to each of the partitioned user name. If a user name exists more than once then it would have multiple row numbers. Using the row number which is other than 1, we can identify the duplicate records.

TEMP\_USERS



**Expected Output**



Solution Query:

select user\_id, user\_name, email from (

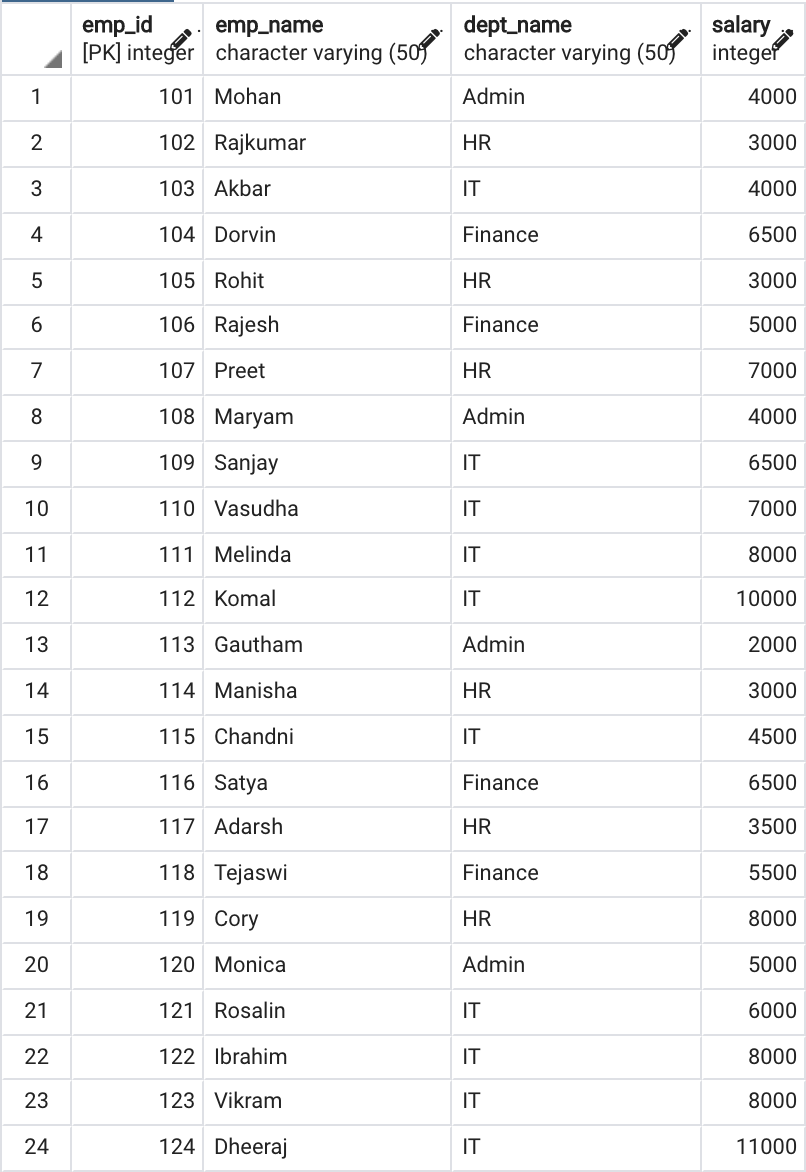
select \*, ROW\_NUMBER() over (partition by user\_name order by user\_id) rn from temp\_user) x

where x.rn > 1;

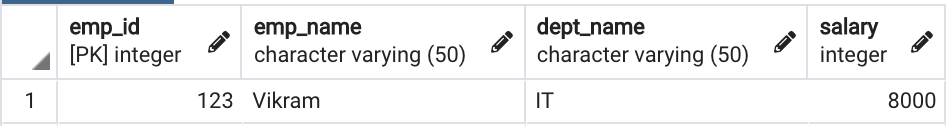
**2. Write a SQL query to fetch the second last record from employee table.**

**Table Name**: EMPLOYEE

**Approach**: Using window function sort the data in descending order based on employee id. Provide a row number to each of the record and fetch the record having row number as 2.

EMPLOYEE  


**Expected Output**



Solution Query:

Select emp\_id, emp\_name, dept\_name, salary from (

Select \*, row\_number() over( order by emp\_id desc) rn from EMPLOYEE ) x where x.rn = 2;

**3. Write a SQL query to display only the details of employees who either earn the highest salary or the lowest salary in each department from the employee table.**

**Table Name**: EMPLOYEE

**Approach**: Write a sub query which will partition the data based on each department and then identify the record with maximum and minimum salary for each of the partitioned department. Finally, from the main query fetch only the data which matches the maximum and minimum salary returned from the sub query.

EMPLOYEE

A table of numbers and letters

Description automatically generated

**Expected Output**

A table with numbers and letters

Description automatically generated

Solution Query:

select emp\_id, emp\_name, e.dept\_name, e.salary, x.max\_sal, x.min\_sal from employee e,

(select dept\_name, min(salary) min\_sal, max(salary) max\_sal from employee group by dept\_name ) x

where x.dept\_name = e.dept\_name and e.salary in (x.min\_sal, x.max\_sal);

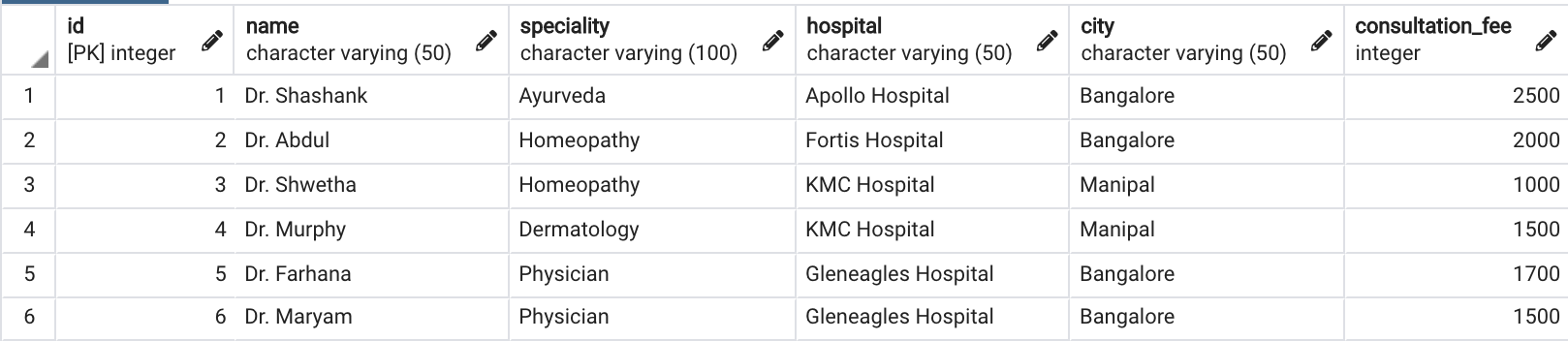
**4. From the doctors table, fetch the details of doctors who work in the same hospital but in different specialty.**

**Table Name**: DOCTORS

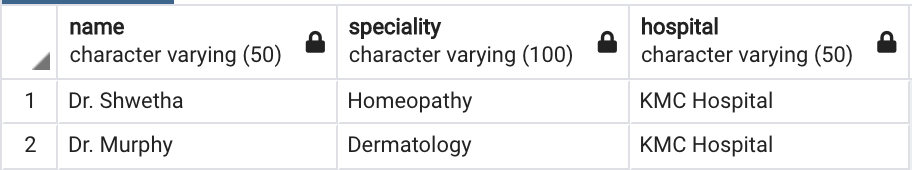
**Approach**: Use self join to solve this problem. Self join is when you join a table to itself.

**Additional Query**: Write SQL query to fetch the doctors who work in same hospital irrespective of their specialty.

DOCTORS



**Expected Output:** Same hospital different speciality



**Expected Output:** Same hospital irrespective of speciality



Solution Query:

select D1.\* FROM DOCTORS D1, DOCTORS D2 where D1.HOSPITAL=D2.HOSPITAL and D1.SPECIALITY <> D2.SPECIALITY;

or

SELECT D1.\* FROM DOCTORS D1 JOIN DOCTORS D2 ON D1.HOSPITAL=D2.HOSPITAL and D1.SPECIALITY <> D2.SPECIALITY;

SELECT \* FROM DOCTORS D1,

(SELECT HOSPITAL, COUNT(HOSPITAL) CNT FROM DOCTORS GROUP BY HOSPITAL HAVING COUNT(HOSPITAL) > 1 ) D2

where D1.HOSPITAL = D2.HOSPITAL;

**5. From the login\_details table, fetch the users who logged in consecutively 3 or more times.**

**Table Name**: LOGIN\_DETAILS

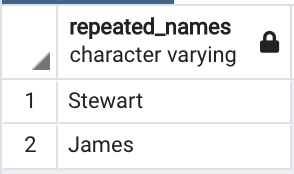
**Approach**: We need to fetch users who have appeared 3 or more times consecutively in login details table. There is a window function which can be used to fetch data from the following record. Use that window function to compare the user name in current row with user name in the next row and in the row following the next row. If it matches then fetch those records.

LOGIN\_DETAILS

A screenshot of a computer

Description automatically generated

**Expected Output**

  
Solution:

with cte as

(select \*,

case when user\_name = lead(user\_name) over (order by login\_id) -- here we compare the current record user name with next record user name

and user\_name = lead(user\_name, 2) over (order by login\_id) -- here we gave offset 2 in lead function to check next to next record

then user\_name -- if both the above conditions are true we get user name else null in repeated\_users columns

else null

end as repeated\_users

from LOGIN\_DETAILS )

select distinct user\_name from cte where repeated\_users is not null;

**6. From the students table, write a SQL query to interchange the adjacent student names.**

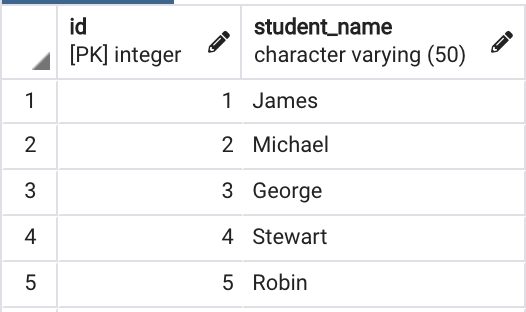
**Note**: If there are no adjacent student then the student name should stay the same.

**Table Name**: STUDENTS

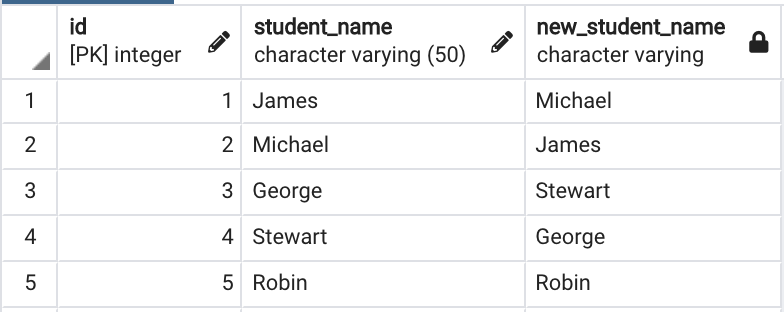
**Approach**: Assuming id will be a sequential number always. If id is an odd number then fetch the student name from the following record. If id is an even number then fetch the student name from the preceding record. Try to figure out the window function which can be used to fetch the preceding the following record data.

If the last record is an odd number then it wont have any adjacent even number hence figure out a way to not interchange the last record data.

STUDENTS



**Expected Output**



**Solution**:

select \*,

case when id%2 <> 0

then lead(student\_name,1,student\_name) over (order by id)

else lag(student\_name) over (order by id)

end as adjacent\_student

from STUDENTS;

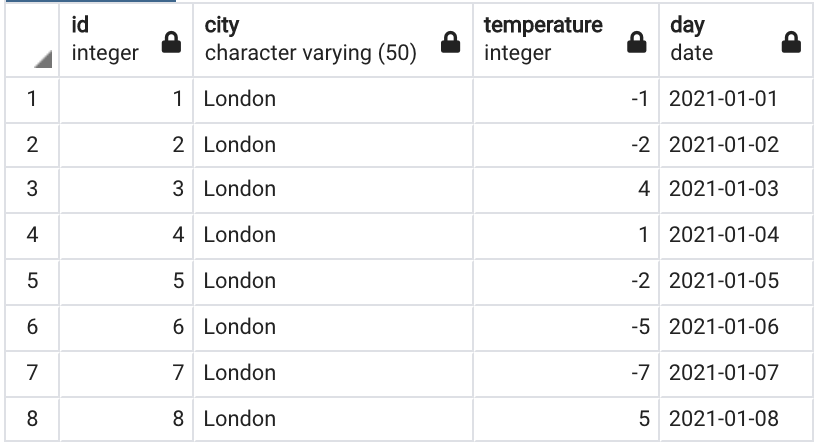
**7. From the weather table, fetch all the records when London had extremely cold temperature for 3 consecutive days or more.**

**Note**: Weather is considered to be extremely cold when its temperature is less than zero.

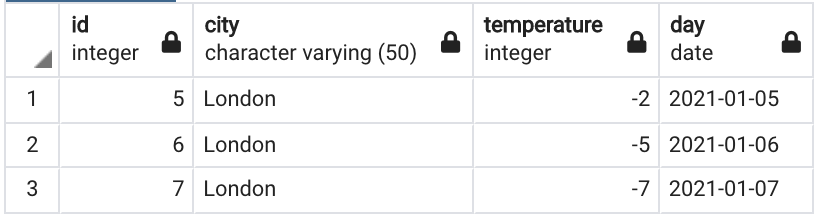
**Table Name**: WEATHER

**Approach**: First using a sub query identify all the records where the temperature was very cold and then use a main query to fetch only the records returned as very cold from the sub query. You will not only need to compare the records following the current row but also need to compare the records preceding the current row. And may also need to compare rows preceding and following the current row. Identify a window function which can do this comparison pretty easily.

WEATHER



**Expected Output**



SOLUTION:

with cte as

(select \*,

case

when temperature < 0 --current day should be less than 0

and lead (temperature, 1) over (order by day) < 0 -- next day should be less than 0 as well

and lead(temperature, 2) over (order by day) < 0 --next to next day also should be less than 0

then 'YES'

when temperature < 0 --current day should be less than 0

and lag (temperature) over (order by day) < 0 --previous day should be less than 0

and lead(temperature) over (order by day) < 0 --next day should be less than 0

then 'YES'

when temperature < 0 --current day should be less than 0

and lag (temperature, 1) over (order by day) < 0 --previous day should be less than 0

and lag(temperature, 2) over (order by day) < 0 --2 days prior also shouls be less than 0

then 'YES'

else null

end extreme\_day

from weather)

select id, city, temperature, day from cte where extreme\_day = 'YES'-- dump all the days which are extreme weather set of 3 days;

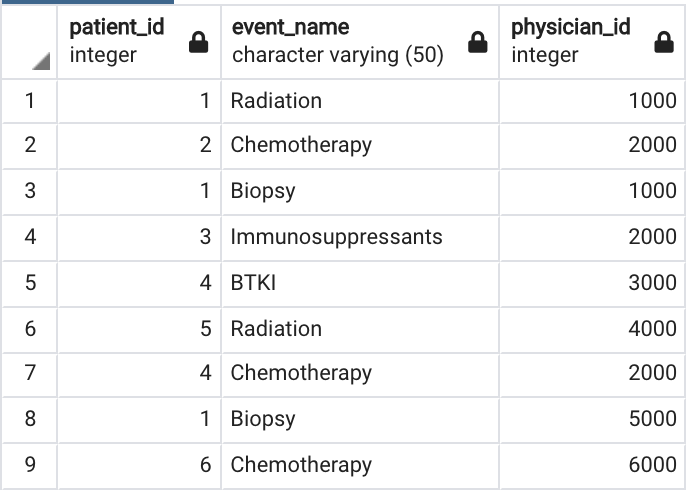
**8. From the following 3 tables (event\_category, physician\_speciality, patient\_treatment), write a SQL query to get the histogram of specialties of the unique physicians who have done the procedures but never did prescribe anything.**

**Table Name**: EVENT\_CATEGORY, PHYSICIAN\_SPECIALITY, PATIENT\_TREATMENT

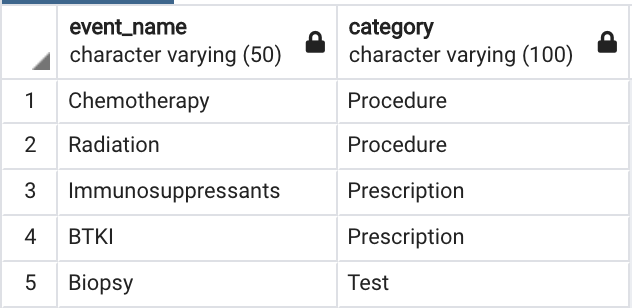
**Approach**: Using the patient treatment and event category table, identify all the physicians who have done “Prescription”. Have this recorded in a sub query.

Then in the main query join the patient treatment, event category and physician speciality table to identify all the physician who have done “Procedure”. From these physicians, remove those physicians you got from sub query to return the physicians who have done Procedure but never did Prescription.

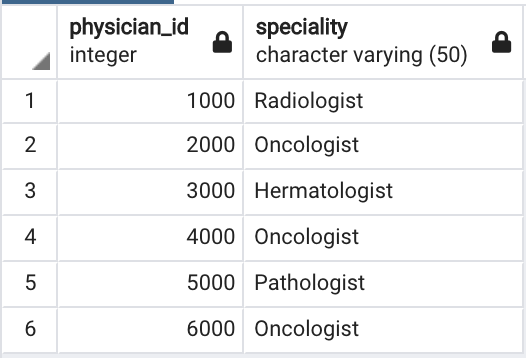
PATIENT\_TREATMENT



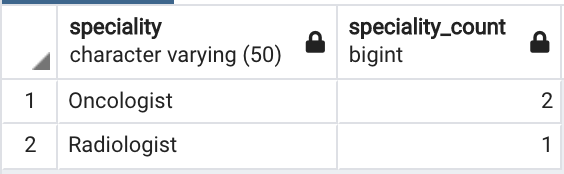
EVENT\_CATEGORY



PHYSICIAN\_SPECIALITY



**Expected Output**



Solution:

with cte3 as

(select PT.physician\_id from PATIENT\_TREATMENT PT join EVENT\_CATEGORY EC

on PT.event\_name = EC.event\_name

and EC.category = 'Prescription')

select PS.speciality, count(1)

from PHYSICIAN\_SPECIALITY PS

join PATIENT\_TREATMENT PT on PS.physician\_id = PT.physician\_id

join EVENT\_CATEGORY EC on PT.event\_name = EC.event\_name

and EC.category = 'Procedure'

and PS.physician\_id not in (select physician\_id from cte3) group by PS.speciality ;

**9. Find the top 2 accounts with the maximum number of unique patients on a monthly basis.**

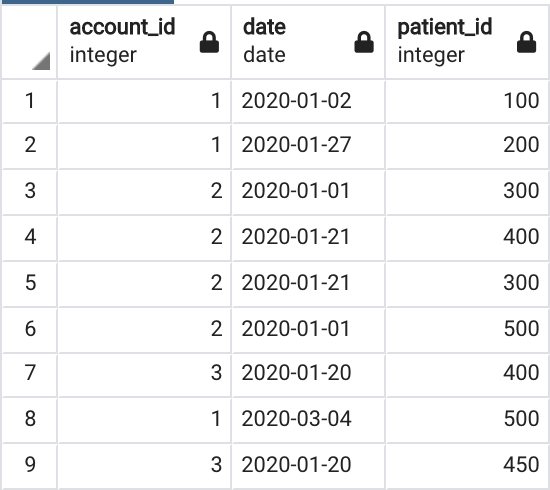
**Note**: Prefer the account id with the least value in case of same number of unique patients

**Table Name**: PATIENT\_LOGS

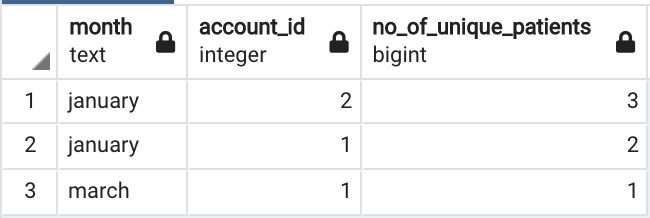
**Approach**: First convert the date to month format since we need the output specific to each month. Then group together all data based on each month and account id so you get the total no of patients belonging to each account per month basis.

Then rank this data as per no of patients in descending order and account id in ascending order so in case there are same no of patients present under multiple account if then the ranking will prefer the account if with lower value. Finally, choose upto 2 records only per month to arrive at the final output.

PATIENT\_LOGS



**Expected Output**



Solution:

select mnth, account\_id, cnt from

(select X.account\_id, X.mnth, count(X.patient\_id) cnt, RANK() over (partition by mnth order by count(X.patient\_id) desc, account\_id ) rnk

from

(select account\_id, MONTH(treatment\_date) mnth, patient\_id

from PATIENT\_LOGS

group by MONTH(treatment\_date), account\_id, patient\_id )

X group by account\_id, mnth ) Z

where rnk <=2 order by mnth, account\_id desc

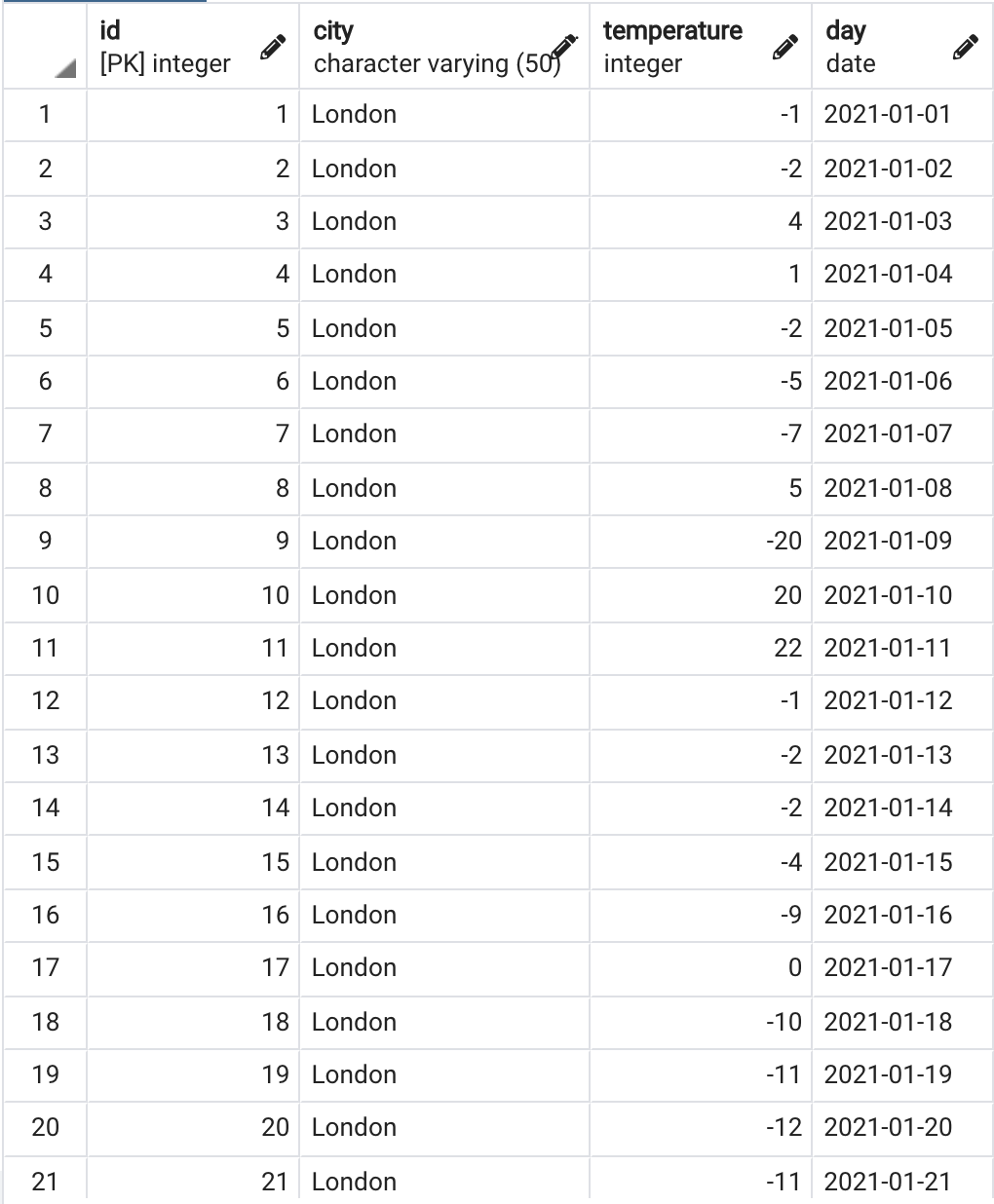
**10. SQL Query to fetch “N” consecutive records from a table based on a certain condition**

Note: Write separate queries to satisfy following scenarios:

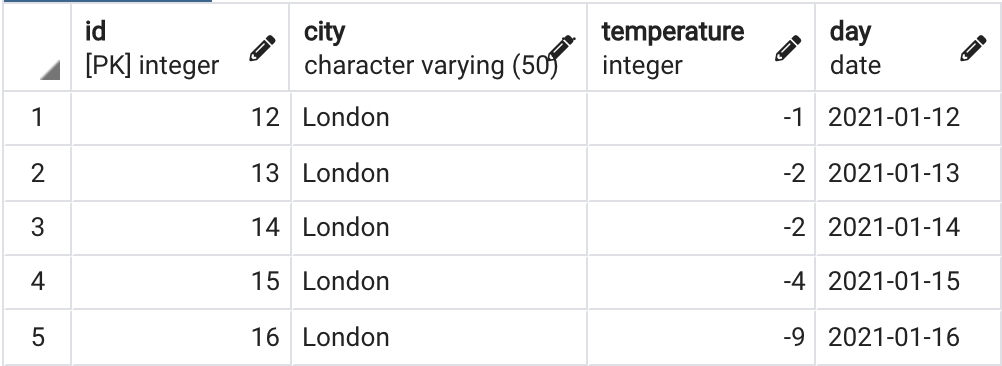
10a. when the table has a primary key

Table Name: WEATHER

**WEATHER table Data**



**10a: Expected Output**



Solution:

/\*

1) we need to eliminate the days where temp is > 0 and consider only <0 records

select \* from WEATHER1 where temp <0;

2) assign a row muner for each record in the table order by id since id is a primary key and in sequence

select \*,ROW\_NUMBER() over (order by id) from WEATHER1 where temp <0

3) the difference between consecutive days of id and row\_number will remain same as ids are in sequence

select \*,id - ROW\_NUMBER() over (order by id) from WEATHER1 where temp <0

4) put the above data in a subquery

4) now partition the records by number of difference of id and row\_number count and give an alian like cnt

5) now fetch cnt = N based on N value

\*/

with T11 as

(select \*,id - ROW\_NUMBER() over (order by id) difference from WEATHER1

where temp <0),

T12 as

(select \*, count(\*) over (partition by difference order by difference ) cnt from T11)

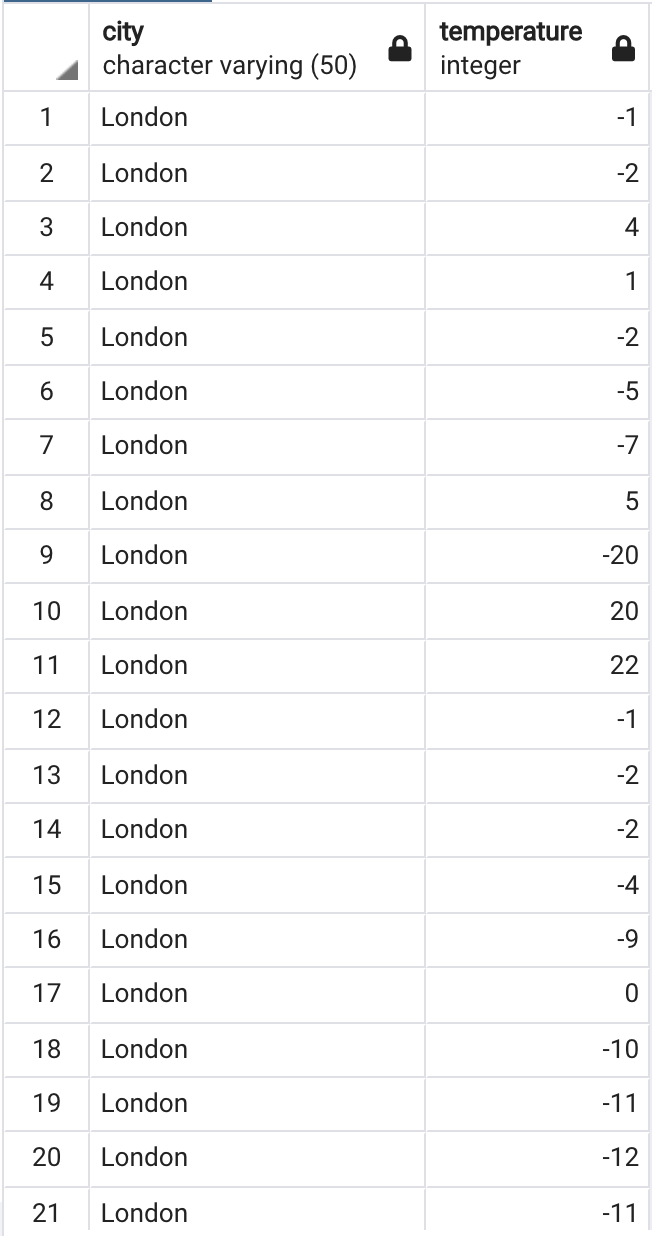
select id, city, temp, recorded\_day from T12

where cnt = X; --X can be any consecutive value…

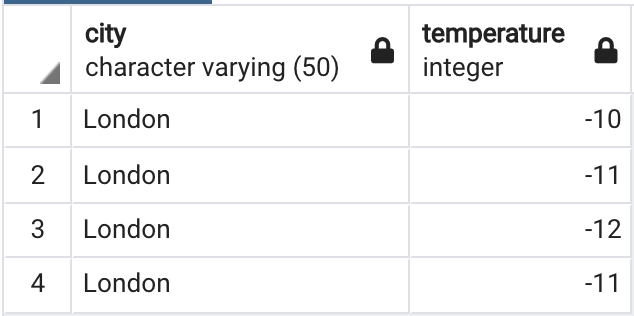
10b. When table does not have a primary key

Table Name: VW\_WEATHER

**VW\_WEATHER table data**



**10b: Expected Output**



Solution:

/\*

1) Lets create view without id column and only non-zero records

create view weather\_vw as select city, temp from WEATHER1

2) Now that the view doesnt have id/PK column, create a subquery with clause add a row\_num column, it will work as id

3) proceed as 10.a

\*/

create view weather\_vw as select city, temp from WEATHER1;

select \* from weather\_vw;

with

w as

(select \*, row\_number() over (order by city) as id

from weather\_vw),

t1 as

(select \*, id - row\_number() over (order by id) as diff

from w

where w.temp < 0),

t2 as

(select \*,

count(\*) over (partition by diff order by diff) as cnt

from t1)

select city, temp, id, cnt

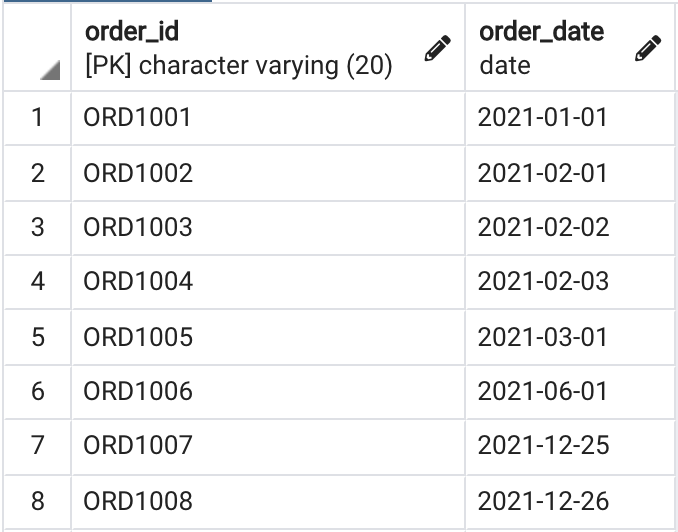
from t2

where t2.cnt = 5;

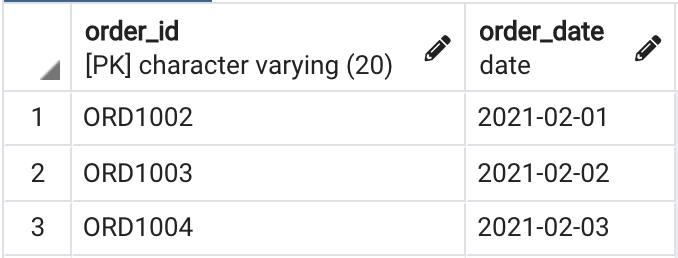
10c. Query logic based on data field

Table Name: ORDERS

**ORDERS table data**



**10c: Expected Output**



Solution:

/\* 1) since the data doesnt have an id column, lets add one by using Row\_num function, and put it in an alias

select \*, ROW\_NUMBER () over (order by ORDER\_ID) id from ORDERS

2) Now, subtract the ROW\_NUM function from order\_date to get a unique difference like we did in previous queries.

(select \*, ROW\_NUMBER () over (order by ORDER\_ID) id, DATEADD(day,-ROW\_NUMBER () over (order by ORDER\_ID),ORDER\_DATE) difference from ORDERS)

3) create a count based on each unique partition in another subquery

4) finally fetch the records based on count X

\*/

select \* from orders;

with

a1 as

(select \*, ROW\_NUMBER () over (order by ORDER\_ID) id, DATEADD(day,-ROW\_NUMBER () over (order by ORDER\_ID),ORDER\_DATE) difference from ORDERS),

a2 as

(select \*, count(order\_id) over (partition by difference) cnt from a1 )

select \* from a2

where cnt = 3;

/\* Amazon and Big4 interview qn:

11. Employee attendance register: need a from and to dates of all status log, for example if employee is present

Between 1st to 3rd of Jan, and then absent on 4th... we should have two records showing like below

EMPLOYEE FROM\_DATE TO\_DATE STATUS

A1 2024-01-01 2024-01-03 PRESENT

A1 2024-01-04 2024-01-04 ABSENT

A1 2024-01-05 2024-01-06 PRESENT

A1 2024-01-07 2024-01-09 ABSENT

etc \*/

create table employee\_1 (employee varchar(2), dates date, status varchar(10));

insert into employee\_1 values('A1', '2024-01-01', 'PRESENT');

insert into employee\_1 values('A1', '2024-01-02', 'PRESENT');

insert into employee\_1 values('A1', '2024-01-03', 'PRESENT');

insert into employee\_1 values('A1', '2024-01-04', 'ABSENT');

insert into employee\_1 values('A1', '2024-01-05', 'PRESENT');

insert into employee\_1 values('A1', '2024-01-06', 'PRESENT');

insert into employee\_1 values('A1', '2024-01-07', 'ABSENT');

insert into employee\_1 values('A1', '2024-01-08', 'ABSENT');

insert into employee\_1 values('A1', '2024-01-09', 'ABSENT');

insert into employee\_1 values('A1', '2024-01-10', 'PRESENT');

insert into employee\_1 values('A2', '2024-01-06', 'PRESENT');

insert into employee\_1 values('A2', '2024-01-07', 'PRESENT');

insert into employee\_1 values('A2', '2024-01-08', 'ABSENT');

insert into employee\_1 values('A2', '2024-01-09', 'PRESENT');

insert into employee\_1 values('A2', '2024-01-10', 'ABSENT');

select \* from employee\_1;

with sq1 as

(select \*, ROW\_NUMBER() over (partition by status, employee order by dates) id

from employee\_1),

sq2 as

(select \*, DATEADD(day,-id,dates) diff from sq1)

select employee, min(dates) from\_date, max(dates) to\_date, status from sq2

group by diff, employee, status order by employee, from\_date;

--alterante approach--

with sq1 as

(select \*, ROW\_NUMBER() over (partition by status, employee order by dates) id

from employee\_1),

sq2 as

(select \*, DATEADD(day,-id,dates) diff from sq1),

sq3 as

--select \* from sq2;

(select \*,

FIRST\_VALUE(dates) over (partition by diff, employee order by employee) as FROM\_DATE,

--FIRST\_VALUE is a default window function, which returen first value of argument we passed based on partition

LAST\_VALUE(dates) over (partition by diff, employee order by employee

range between unbounded preceding and unbounded following) as TO\_DATE --this line is a standard syntax to get last value to avoid default frame clause

from SQ2)

select distinct employee, FROM\_DATE, TO\_DATE, STATUS

from SQ3 order by employee, FROM\_DATE;

/\*--------------------------Window functions--------------------------------\*/

select \* from employee;

--display max sal by each dept

select dept\_name, max(salary) max\_sal from employee group by dept\_name;

--here we didnt display the rest of the values... in case we need to show the max sal for each record,

--thats when window functions come into picture. see example below.

select \*, max(salary) over (partition by dept\_name) from employee;

--there will be partitions created based on the column specified (in this case its dept\_name)

--and aggregate function(in this case its max function) will be executed on each partition

--row\_num, rank, dense\_rank, lead, lag

--are functions, which just assign a value for each record which has been detected/identified by the arguments

--mentioned in over clause, thats why we dont pass any argument inside the function name unline aggregate functions.

select \*, ROW\_NUMBER() over (order by emp\_id) rownum from employee; --order by is mandatory in row\_number function in sql server,

--in this case it can be used as a PK to uniquely identify every record in the table.

--we can create windows for each partition and then given row\_numbers as well

select \*, ROW\_NUMBER() over (partition by dept\_name order by emp\_id) rownum from employee;

--lets say use case where they need first two employees joined in each dept, here row\_number is useful.

select \* from

(select \*, row\_number() over (partition by dept\_name order by emp\_id) rownum from employee ) x

where x.rownum < 3;

--RANK ; duplicate values will have same rank and next number will be skipped.

--Fetch top 3 sal in each dept

select \*, rank() over (partition by dept\_name order by salary desc) ranks from employee; -- will assign ranks to

--each employee based on salary, we just need to pick top 3 salaries from above

select \* from

(select \*, rank() over (partition by dept\_name order by salary desc) ranks from employee) y

where y.ranks <=3;

--similarly faq is find nth max salary from table, lets see how we can build it below.

select \* from

(select \*, rank() over (order by salary desc) ranks from employee) y

where y.ranks = 15; -- here 4 can be replaced by any rank within table

--dense\_rank... same as rank but when duplicate values encountered, next rank will not be skipped...

--lets see all rank, dense\_rank, and row\_number in one example

--you may observe the difference at records with salary 2000 below.

select \*,

rank() over (order by salary desc) ranks,

dense\_rank() over (order by salary desc) dense,

row\_number() over (order by salary desc) rownum

from employee;

--lead and lag

--sal of current employee is higher/lower than next/prev employee

--LEAD: will always have last record null, it has 4 arguments

-- 1) on which column the function need to perform

-- 2) what is the skip value... by default 1, e.g: if we give 2, it will compare the following second record.

-- 3) default value to replace the last record

-- 4) over function in general to define windows and orders

select \*, lead(salary,1,9999) over (order by emp\_id) next\_emp\_sal from employee;

--similarly check per dept by creating paritions

select \*, lead(salary) over (partition by dept\_name order by emp\_id) next\_emp\_sal from employee;

--lag is also similar but as name suggest it will check with previous record and display in window...

select \*, lag(salary) over (partition by dept\_name order by emp\_id) next\_emp\_sal from employee;

--lets see how we can display some meaningful summary based on lead, lag for salary comparision

select \*,

case

when salary < lead(salary) over (partition by dept\_name order by emp\_id)

then 'Lower Salary than next employee'

when salary > lead(salary) over (partition by dept\_name order by emp\_id)

then 'Higher Salary than next employee'

when salary = lead(salary) over (partition by dept\_name order by emp\_id)

then 'Equal Salary as next employee'

else NULL

end next\_emp\_sal

from employee;

/\*--------------------------------------------------------------------------\*/

/\* Windows functions advanced topics \*/

CREATE TABLE product

(

product\_category varchar(255),

brand varchar(255),

product\_name varchar(255),

price int

);

INSERT INTO product VALUES

('Phone', 'Apple', 'iPhone 12 Pro Max', 1300),

('Phone', 'Apple', 'iPhone 12 Pro', 1100),

('Phone', 'Apple', 'iPhone 12', 1000),

('Phone', 'Samsung', 'Galaxy Z Fold 3', 1800),

('Phone', 'Samsung', 'Galaxy Z Flip 3', 1000),

('Phone', 'Samsung', 'Galaxy Note 20', 1200),

('Phone', 'Samsung', 'Galaxy S21', 1000),

('Phone', 'OnePlus', 'OnePlus Nord', 300),

('Phone', 'OnePlus', 'OnePlus 9', 800),

('Phone', 'Google', 'Pixel 5', 600),

('Laptop', 'Apple', 'MacBook Pro 13', 2000),

('Laptop', 'Apple', 'MacBook Air', 1200),

('Laptop', 'Microsoft', 'Surface Laptop 4', 2100),

('Laptop', 'Dell', 'XPS 13', 2000),

('Laptop', 'Dell', 'XPS 15', 2300),

('Laptop', 'Dell', 'XPS 17', 2500),

('Earphone', 'Apple', 'AirPods Pro', 280),

('Earphone', 'Samsung', 'Galaxy Buds Pro', 220),

('Earphone', 'Samsung', 'Galaxy Buds Live', 170),

('Earphone', 'Sony', 'WF-1000XM4', 250),

('Headphone', 'Sony', 'WH-1000XM4', 400),

('Headphone', 'Apple', 'AirPods Max', 550),

('Headphone', 'Microsoft', 'Surface Headphones 2', 250),

('Smartwatch', 'Apple', 'Apple Watch Series 6', 1000),

('Smartwatch', 'Apple', 'Apple Watch SE', 400),

('Smartwatch', 'Samsung', 'Galaxy Watch 4', 600),

('Smartwatch', 'OnePlus', 'OnePlus Watch', 220);

--FIRST\_VALUE : function extracts very first record value from a partition mentioned in the function.

--E.g: write a query to display the most expensive product under each category (corresponding to each record).

select \*,

first\_value(product\_name) over (partition by product\_category order by price desc) most\_exp\_prod

from product;

--LAST VALUE: function extracts very last record value from a partition mentioned in the function.

--write a query to display the least expensive product under each category (corresponding to each record).

select \*,

last\_value(product\_name) over (partition by product\_category order by price desc) least\_exp\_prod

from product;

A screenshot of a computer

Description automatically generated

/\*writing the query like first\_value is not going to help here .... because of frame clause

FRAME Clause: when data is divided into partition, when controller read data in each partition, it treats them in frames.

each frame can be defined by combination of considering from first record in the partition to end of the last record

in the parition. Then on that frame entire last\_value function will be applied.

condition to built tha frame is by using "range between unbounded preceding and unbounded following", this will tell controller

that use the entire partition from first record to last.

defaule behavior condition is "range between unbounded preceeding and current row following"... in our result set, since we

didnt mention any condition inside last\_value over clause, it has taken default and as every new record is being added

into the frame, it is showing new least\_exp\_prod.

We can change keyword from 'range' to 'row', which will give different results when duplicate values present in data.

Instead of unbounded, we can give numeric value, and controller will consider only those number of records.

Lets see below with range condition to tell the FRAME to use entire parition

\*/

select \*,

last\_value(product\_name) over (partition by product\_category order by price desc

range between unbounded preceding and unbounded following) least\_exp\_prod

from product;

--lets display both most expensive and least expensive in same query

select \*,

first\_value(product\_name) over (partition by product\_category order by price desc) most\_exp\_prod,

last\_value(product\_name) over (partition by product\_category order by price desc

range between unbounded preceding and unbounded following) least\_exp\_prod

from product;

A screenshot of a computer

Description automatically generated

--here if we are using same OVER() function multiple times in a query, it can be shorted by below alternate approach

select \*,

first\_value(product\_name) over W most\_exp\_prod,

last\_value(product\_name) over W least\_exp\_prod

from product

window W as (partition by product\_category order by price desc

range between unbounded preceding and unbounded following) ;

--another alternate approach to get the results using just FIRST\_VALUE

select \*,

first\_value(product\_name) over (partition by product\_category order by price desc) most\_exp\_prod,

first\_value(product\_name) over (partition by product\_category order by price ) least\_exp\_prod

from product;

--NTH\_Value: Similar to FIRST and LAST Value, will result Nth value in the partition/table

--second most expensive product

--in sql server its not there, and nth\_value also depends on frame clause

select \*,

first\_value(product\_name) over W most\_exp\_prod,

last\_value(product\_name) over W least\_exp\_prod,

nth\_value(product\_name, 2) over W second\_most\_exp\_product

from product

window W as (partition by product\_category order by price desc

range between unbounded preceding and unbounded following) ;

--NTILE: used to group data in certain partition and treat it as a bucket based on single numric argument we pass;

--mostlikely SQL will create equal number of buckets, if not will distribute rows to one bucket at a time.

--write a query to categorize phones in expensive, mid-range and cheper phones buckets

select \* ,

NTILE(3) OVER (order by price )

from product ;

with CTE as

(select \* ,

NTILE(3) OVER (order by price ) TILE

from product)

select \*,

case

when TILE = 1 then 'cheper phone'

when TILE = 2 then 'mid range phone'

when TILE = 3 then 'expensive phone'

end phone\_Category

from CTE

where product\_category='Phone';

/\*CUME\_DIST: (cumulative distribution)

value --> 1 <= CUME\_DIST >0

FORMULA = current rownum (or rownum with value same as current row num) /Total num of rows

basically if each row position will be divided by total number of rows.\*/

--Query to fetch all records which constitutes first 30% of table based on price

with X as

(select \*,

CUME\_DIST() over (order by price desc) cum\_dist, -- will display in fractions, we need to convert into numeric by below

round(CUME\_DIST() over (order by price desc)\*100, 2) cum\_dist\_percentage

from product )

select product\_name, concat(cum\_dist\_percentage,'%' )

from X

where cum\_dist\_percentage <= 30 ;

/\*PERCENT\_RANK (relative rank of the current row/percentage ranking)

value --> 1 <= PERCENT\_RANK > 0

FORMULA: Current row num -1 / Total number of rows -1

\*/

--Query to find out how much percentage more expensive is 'Galaxy Z fold 3' when compared to all other products

WITH K as

(select \*,

PERCENT\_RANK() over (order by price ) percent\_rank, -- this displays in fractions, we need to convert to numbers

round(PERCENT\_RANK() over (order by price )\*100, 2) percent\_rank\_num

from product)

select percent\_rank\_num from K where product\_name = 'Galaxy Z Fold 3';

/\*--------------------------------------------------------------------------\*/

/\* Recursive Queries

Syntax:

with [Recursive] --recursive key word is optional in some of the databases

CTE\_NAME as

(

select query -- this is non recursive query or base query

union [all] -- union all is optional in some of the databases

select query (recursive query using CTE\_NAME [with a termination condition])

)

select \* from CTE\_NAME

\*/

--Q1. Return numbers 1 to 10 without using any built in functions

WITH

CTE AS

( select 1 num

union all

select num+1 from CTE

where num < 10

)

select \* from CTE;

--Q2. Find the hierarchy of employees under a given manager

create table emp\_details (id int primary key, emp\_name varchar(20), manager\_id int, salary int, designation varchar(20));

insert into emp\_details values (1, 'Shripadh', NULL , 10000, 'CEO');

insert into emp\_details values (2, 'Satya', 5 , 1400, 'Software Engineer');

insert into emp\_details values (3, 'Jia', 5 , 500, 'Data Analyst');

insert into emp\_details values (4, 'David', 5 , 1800, 'Data Scientist');

insert into emp\_details values (5, 'Michael', 7 , 3000, 'Manager');

insert into emp\_details values (6, 'Arvind', 7 , 2400, 'Architect');

insert into emp\_details values (7, 'Asha', 1 , 4200, 'CTO');

insert into emp\_details values (8, 'Maryam', 1 , 3500, 'Manager');

insert into emp\_details values (9, 'Reshma', 8 , 2000, 'Business Analyst');

insert into emp\_details values (10, 'Akshay', 8 , 2500, 'Java Developer');

select \* from emp\_details

where manager\_id = 7;

with rec\_manager\_emp as

(select id, emp\_name, manager\_id, designation,1 lvl from emp\_details where emp\_name = 'Asha'

union all

select E.id, E.emp\_name, E.manager\_id, E.designation, H.lvl+1 from rec\_manager\_emp H

join emp\_details E

on H.id = E.manager\_id)

select \* from rec\_manager\_emp;

--Q3. Find the hierarchy of managers for a given employee

with rec\_emp\_manger as

(select id, emp\_name, manager\_id, designation, 1 lvl from emp\_details where emp\_name = 'David'

union all

select E.id, E.emp\_name, E.manager\_id, E.designation, H.lvl+1 from rec\_emp\_manger H

join emp\_details e

on H.manager\_id = e.id)

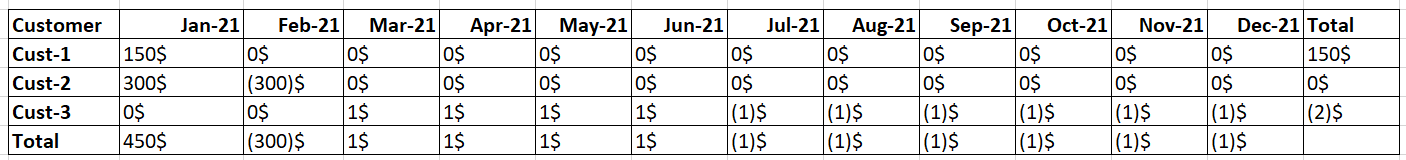
select H2.emp\_name EMP\_NAME, E2.EMP\_NAME MANAGER\_NAME, H2.DESIGNATION, H2.LVL from rec\_emp\_manger H2

join emp\_details E2 on

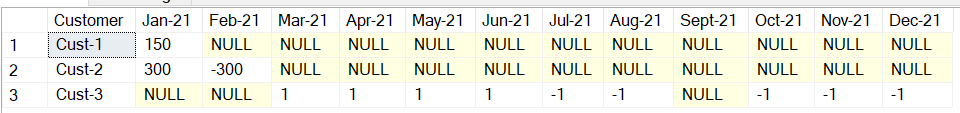
E2.id = H2.manager\_id;

/\*--------------------------------------------------------------------------\*/

Expected O/P:



--Output at this step looks like below.



select \* from

(select customer\_id as Customer,

FORMAT(sales\_date, 'MMM-yy') as sales\_date,

cast(replace(amount,'$','') as int) as amount

from sales\_data) as sales\_data --(since sql server needs alias for a subquery

pivot

(--aggregate function :here we want to sum the amount so

sum(amount)

--for columns values to be selected --(in this case its like "Jan-21", "Feb-21", "Mar-21", ...etc)

for sales\_date in ( [Jan-21],[Feb-21],[Mar-21],[Apr-21],[May-21],[Jun-21],

[Jul-21],[Aug-21],[Sept-21],[Oct-21],[Nov-21],[Dec-21] )

) as pivot\_table

-- at this point we need to build the same data for TOTAL record so join the above data with additional data by union

UNION

select \* from

(select 'Total' as Customer,

FORMAT(sales\_date, 'MMM-yy') as sales\_date,

cast(replace(amount,'$','') as int) as amount

from sales\_data) as sales\_data --(since sql server needs alias for a subquery

pivot

(--aggregate function :here we want to sum the amount so

sum(amount)

--for columns values to be selected --(in this case its like "Jan-21", "Feb-21", "Mar-21", ...etc)

for sales\_date in ( [Jan-21],[Feb-21],[Mar-21],[Apr-21],[May-21],[Jun-21],

[Jul-21],[Aug-21],[Sept-21],[Oct-21],[Nov-21],[Dec-21] )

) as pivot\_table

--once the total is added, the o/p looks like below

A screenshot of a calendar

Description automatically generated

--now we need to remove nulls and treat them as 0s, it can by done by coalesce in sql server; we need to build it

--from existing result set so, lets give it an alias name inside with clause now.

with pivot\_data as

(

select \* from

(select customer\_id as Customer,

FORMAT(sales\_date, 'MMM-yy') as sales\_date,

cast(replace(amount,'$','') as int) as amount

from sales\_data) as sales\_data --(since sql server needs alias for a subquery

pivot

(--aggregate function :here we want to sum the amount so

sum(amount)

--for columns values to be selected --(in this case its like "Jan-21", "Feb-21", "Mar-21", ...etc)

for sales\_date in ( [Jan-21],[Feb-21],[Mar-21],[Apr-21],[May-21],[Jun-21],

[Jul-21],[Aug-21],[Sep-21],[Oct-21],[Nov-21],[Dec-21] )

) as pivot\_table

-- at this point we need to build the same data for TOTAL record so join the above data with additional data by union

UNION

select \* from

(select 'Total' as Customer,

FORMAT(sales\_date, 'MMM-yy') as sales\_date,

cast(replace(amount,'$','') as int) as amount

from sales\_data) as sales\_data --(since sql server needs alias for a subquery

pivot

(--aggregate function :here we want to sum the amount so

sum(amount)

--for columns values to be selected --(in this case its like "Jan-21", "Feb-21", "Mar-21", ...etc)

for sales\_date in ( [Jan-21],[Feb-21],[Mar-21],[Apr-21],[May-21],[Jun-21],

[Jul-21],[Aug-21],[Sep-21],[Oct-21],[Nov-21],[Dec-21] )

) as pivot\_table

),

final\_data as

(

select customer,

coalesce([Jan-21],0) as 'Jan-21',

coalesce([Feb-21],0) as 'Feb-21',

coalesce([Mar-21],0) as 'Mar-21',

coalesce([Apr-21],0) as 'Apr-21',

coalesce([May-21],0) as 'May-21',

coalesce([Jun-21],0) as 'Jun-21',

coalesce([Jul-21],0) as 'Jul-21',

coalesce([Aug-21],0) as 'Aug-21',

coalesce([Sep-21],0) as 'Sep-21',

coalesce([Oct-21],0) as 'Oct-21',

coalesce([Nov-21],0) as 'Nov-21',

coalesce([Dec-21],0) as 'Dec-21'

from pivot\_data

)

select \* from final\_data ;

O/P: looks like below

A screenshot of a calendar

Description automatically generated

--now we need to come up with Total column and add $, and () for -ve values; this can be done in final\_data extraction itself.

--base query is ready, put it in syntax

with pivot\_data as

(

select \* from

(select customer\_id as Customer,

FORMAT(sales\_date, 'MMM-yy') as sales\_date,

cast(replace(amount,'$','') as int) as amount

from sales\_data) as sales\_data --(since sql server needs alias for a subquery

pivot

(--aggregate function :here we want to sum the amount so

sum(amount)

--for columns values to be selected --(in this case its like "Jan-21", "Feb-21", "Mar-21", ...etc)

for sales\_date in ( [Jan-21],[Feb-21],[Mar-21],[Apr-21],[May-21],[Jun-21],

[Jul-21],[Aug-21],[Sep-21],[Oct-21],[Nov-21],[Dec-21] )

) as pivot\_table

-- at this point we need to build the same data for TOTAL record so join the above data with additional data by union

UNION

select \* from

(select 'Total' as Customer,

FORMAT(sales\_date, 'MMM-yy') as sales\_date,

cast(replace(amount,'$','') as int) as amount

from sales\_data) as sales\_data --(since sql server needs alias for a subquery

pivot

(--aggregate function :here we want to sum the amount so

sum(amount)

--for columns values to be selected --(in this case its like "Jan-21", "Feb-21", "Mar-21", ...etc)

for sales\_date in ( [Jan-21],[Feb-21],[Mar-21],[Apr-21],[May-21],[Jun-21],

[Jul-21],[Aug-21],[Sep-21],[Oct-21],[Nov-21],[Dec-21] )

) as pivot\_table

),

final\_data as

(

select customer,

coalesce([Jan-21],0) as 'Jan-21',

coalesce([Feb-21],0) as 'Feb-21',

coalesce([Mar-21],0) as 'Mar-21',

coalesce([Apr-21],0) as 'Apr-21',

coalesce([May-21],0) as 'May-21',

coalesce([Jun-21],0) as 'Jun-21',

coalesce([Jul-21],0) as 'Jul-21',

coalesce([Aug-21],0) as 'Aug-21',

coalesce([Sep-21],0) as 'Sep-21',

coalesce([Oct-21],0) as 'Oct-21',

coalesce([Nov-21],0) as 'Nov-21',

coalesce([Dec-21],0) as 'Dec-21'

from pivot\_data

)

select customer,

case when [Jan-21] < 0 then concat('(',[Jan-21]\*-1,'$)') else concat ([Jan-21],'$') end as 'Jan-21',

case when [Feb-21] < 0 then concat('(',[Feb-21]\*-1,'$)') else concat ([Feb-21],'$') end as 'Feb-21',

case when [Mar-21] < 0 then concat('(',[Mar-21]\*-1,'$)') else concat ([Mar-21],'$') end as 'Mar-21',

case when [Apr-21] < 0 then concat('(',[Apr-21]\*-1,'$)') else concat ([Apr-21],'$') end as 'Apr-21',

case when [May-21] < 0 then concat('(',[May-21]\*-1,'$)') else concat ([May-21],'$') end as 'May-21',

case when [Jun-21] < 0 then concat('(',[Jun-21]\*-1,'$)') else concat ([Jun-21],'$') end as 'Jun-21',

case when [Jul-21] < 0 then concat('(',[Jul-21]\*-1,'$)') else concat ([Jul-21],'$') end as 'Jul-21',

case when [Aug-21] < 0 then concat('(',[Aug-21]\*-1,'$)') else concat ([Aug-21],'$') end as 'Aug-21',

case when [Sep-21] < 0 then concat('(',[Sep-21]\*-1,'$)') else concat ([Sep-21],'$') end as 'Sep-21',

case when [Oct-21] < 0 then concat('(',[Oct-21]\*-1,'$)') else concat ([Oct-21],'$') end as 'Oct-21',

case when [Nov-21] < 0 then concat('(',[Nov-21]\*-1,'$)') else concat ([Nov-21],'$') end as 'Nov-21',

case when [Dec-21] < 0 then concat('(',[Dec-21]\*-1,'$)') else concat ([Dec-21],'$') end as 'Dec-21'

from final\_data ;

O/P: looks like below:

A screenshot of a calendar

Description automatically generated

we are yet to get the Total Column and its corresponding sum for each customer,

adding the below code at the end give result like below

case when customer = 'Total' then ''

else [Jan-21]+[Feb-21]+[Mar-21]+[Apr-21]+[May-21]+[Jun-21]+[Jul-21]+[Aug-21]+[Sep-21]+[Oct-21]+[Nov-21]+[Dec-21]

end as 'Total'

from final\_data ;

A screenshot of a calendar

Description automatically generated

We still have no $ value and -ve values in Total column, so to eliminate then use below code

case when customer = 'Total' then ''

else case when [Jan-21]+[Feb-21]+[Mar-21]+[Apr-21]+[May-21]+[Jun-21]+[Jul-21]+[Aug-21]+[Sep-21]+[Oct-21]+[Nov-21]+[Dec-21] < 0

then CONCAT('(', ([Jan-21]+[Feb-21]+[Mar-21]+[Apr-21]+[May-21]+[Jun-21]+[Jul-21]+[Aug-21]+[Sep-21]+[Oct-21]+[Nov-21]+[Dec-21])\*-1 ,')$')

else

CONCAT([Jan-21]+[Feb-21]+[Mar-21]+[Apr-21]+[May-21]+[Jun-21]+[Jul-21]+[Aug-21]+[Sep-21]+[Oct-21]+[Nov-21]+[Dec-21],'$')

end

end as 'Total'

from final\_data ;

this will give output as below: --corrected -ve, $ mistakes for rest of the columns as well

A screenshot of a calendar

Description automatically generated

Final query looks like below.

--base query is ready, put it in syntax

with pivot\_data as

(

select \* from

(select customer\_id as Customer,

FORMAT(sales\_date, 'MMM-yy') as sales\_date,

cast(replace(amount,'$','') as int) as amount

from sales\_data) as sales\_data --(since sql server needs alias for a subquery

pivot

(

sum(amount)

for sales\_date in ( [Jan-21],[Feb-21],[Mar-21],[Apr-]21],[May-21],[Jun-21],

[Jul-21],[Aug-21],[Sep-21],[Oct-21],[Nov-21],[Dec-21] )

) as pivot\_table

-- at this point we need to build the same data for TOTAL record so join the above data with additional data by union

UNION

select \* from

(select 'Total' as Customer,

FORMAT(sales\_date, 'MMM-yy') as sales\_date,

cast(replace(amount,'$','') as int) as amount

from sales\_data) as sales\_data --(since sql server needs alias for a subquery

pivot

(--aggregate function :here we want to sum the amount so

sum(amount)

--for columns values to be selected --(in this case its like "Jan-21", "Feb-21", "Mar-21", ...etc)

for sales\_date in ( [Jan-21],[Feb-21],[Mar-21],[Apr-21],[May-21],[Jun-21],

[Jul-21],[Aug-21],[Sep-21],[Oct-21],[Nov-21],[Dec-21] )

) as pivot\_table

),

final\_data as

(

select customer,

coalesce([Jan-21],0) as 'Jan-21',

coalesce([Feb-21],0) as 'Feb-21',

coalesce([Mar-21],0) as 'Mar-21',

coalesce([Apr-21],0) as 'Apr-21',

coalesce([May-21],0) as 'May-21',

coalesce([Jun-21],0) as 'Jun-21',

coalesce([Jul-21],0) as 'Jul-21',

coalesce([Aug-21],0) as 'Aug-21',

coalesce([Sep-21],0) as 'Sep-21',

coalesce([Oct-21],0) as 'Oct-21',

coalesce([Nov-21],0) as 'Nov-21',

coalesce([Dec-21],0) as 'Dec-21'

from pivot\_data

)

select customer,

case when [Jan-21] < 0 then concat('(',[Jan-21]\*-1,')$') else concat ([Jan-21],'$') end as 'Jan-21',

case when [Feb-21] < 0 then concat('(',[Feb-21]\*-1,')$') else concat ([Feb-21],'$') end as 'Feb-21',

case when [Mar-21] < 0 then concat('(',[Mar-21]\*-1,')$') else concat ([Mar-21],'$') end as 'Mar-21',

case when [Apr-21] < 0 then concat('(',[Apr-21]\*-1,')$') else concat ([Apr-21],'$') end as 'Apr-21',

case when [May-21] < 0 then concat('(',[May-21]\*-1,')$') else concat ([May-21],'$') end as 'May-21',

case when [Jun-21] < 0 then concat('(',[Jun-21]\*-1,')$') else concat ([Jun-21],'$') end as 'Jun-21',

case when [Jul-21] < 0 then concat('(',[Jul-21]\*-1,')$') else concat ([Jul-21],'$') end as 'Jul-21',

case when [Aug-21] < 0 then concat('(',[Aug-21]\*-1,')$') else concat ([Aug-21],'$') end as 'Aug-21',

case when [Sep-21] < 0 then concat('(',[Sep-21]\*-1,')$') else concat ([Sep-21],'$') end as 'Sep-21',

case when [Oct-21] < 0 then concat('(',[Oct-21]\*-1,')$') else concat ([Oct-21],'$') end as 'Oct-21',

case when [Nov-21] < 0 then concat('(',[Nov-21]\*-1,')$') else concat ([Nov-21],'$') end as 'Nov-21',

case when [Dec-21] < 0 then concat('(',[Dec-21]\*-1,')$') else concat ([Dec-21],'$') end as 'Dec-21',

case when customer = 'Total' then ''

else case when [Jan-21]+[Feb-21]+[Mar-21]+[Apr-21]+[May-21]+[Jun-21]+[Jul-21]+[Aug-21]+[Sep-21]+[Oct-21]+[Nov-21]+[Dec-21] < 0

then CONCAT('(', ([Jan-21]+[Feb-21]+[Mar-21]+[Apr-21]+[May-21]+[Jun-21]+[Jul-21]+[Aug-21]+[Sep-21]+[Oct-21]+[Nov-21]+[Dec-21])\*-1 ,')$')

else

CONCAT([Jan-21]+[Feb-21]+[Mar-21]+[Apr-21]+[May-21]+[Jun-21]+[Jul-21]+[Aug-21]+[Sep-21]+[Oct-21]+[Nov-21]+[Dec-21],'$')

end

end as 'Total'

from final\_data ;

/\*--------------------------------------------------------------------------\*/

/\*Further Pivot Syntax:

SELECT column\_list

FROM (table\_expression/subquery ) alias

PIVOT

(

aggregate\_function(aggregate\_column)

FOR pivot\_column

IN( pivot\_column\_values )

) [AS] pivot\_table\_alias

[ORDER BY column\_list];

\*/

CREATE TABLE BookSales

(BookType VARCHAR(20), SalesYear INT, BookSales MONEY);

INSERT INTO BookSales VALUES('Fiction', 2014, 11201);

INSERT INTO BookSales VALUES('Fiction', 2014, 12939);

INSERT INTO BookSales VALUES('Fiction', 2013, 10436);

INSERT INTO BookSales VALUES('Fiction', 2013, 9346);

INSERT INTO BookSales VALUES('Nonfiction', 2014, 7214);

INSERT INTO BookSales VALUES('Nonfiction', 2014, 5800);

INSERT INTO BookSales VALUES('Nonfiction', 2013, 8922);

INSERT INTO BookSales VALUES('Nonfiction', 2013, 7462);

select \* from BookSales;

Actual Data:

A screenshot of a computer

Description automatically generated

Expected Data:

A screenshot of a computer

Description automatically generated

Solution:

CREATE TABLE BookSales

(BookType VARCHAR(20), SalesYear INT, Sales INT);

INSERT INTO BookSales VALUES('Fiction', 2014, 11201);

INSERT INTO BookSales VALUES('Fiction', 2014, 12939);

INSERT INTO BookSales VALUES('Fiction', 2013, 10436);

INSERT INTO BookSales VALUES('Fiction', 2013, 9346);

INSERT INTO BookSales VALUES('Nonfiction', 2014, 7214);

INSERT INTO BookSales VALUES('Nonfiction', 2014, 5800);

INSERT INTO BookSales VALUES('Nonfiction', 2013, 8922);

INSERT INTO BookSales VALUES('Nonfiction', 2013, 7462);

select \* from BookSales;

select \* from (

select BookType,

SalesYear,

Sales

From BookSales) as book\_sales

pivot

(

sum(Sales)

for SalesYear in ([2013], [2014])

) as pivot\_table

Output:

A screenshot of a computer

Description automatically generated

--trying with case statement, and trying to print output like above example

with base\_query as --there is no need to build base query, but as a standard practice doing it.

(select BookType,

SalesYear,

Sales

From BookSales),

final\_query as

(select BookType,

sum(case when SalesYear = 2014 then Sales else 0 end) '2014',

sum(case when SalesYear = 2013 then Sales else 0 end) '2013'

--sum(Sales) as 'Total'

from base\_query

group by BookType

Union

select 'Total' as BookType,

sum(case when SalesYear = 2014 then Sales else 0 end) '2014',

sum(case when SalesYear = 2013 then Sales else 0 end) '2013'

--'' as 'Total'

from base\_query )

select

BookType, [2014], [2013]

,case when BookType = 'Total' then '' else (concat('',[2014]+[2013])) end as 'Total'

from final\_query;

Output:

A screenshot of a computer

Description automatically generated

/\* working on dupilcate data removel \*/

create table cars (id int, model varchar(20), brand varchar(20), color varchar(20), make int);

insert into cars values (1, 'Model S', 'Tesla', 'Blue', 2018),

(2, 'EQS', 'Mercedes-Benz', 'Black', 2022),

(3, 'iX', 'BMW', 'Red', 2022),

(4, 'ioniq 5', 'Hyundai', 'White', 2021),

(5, 'Model S', 'Tesla', 'Silver', 2018),

(6, 'ioniq 5', 'Hyundai', 'Green', 2021);

select \* from cars order by model, brand;

/\* Scenario-1: Data duplicated based on SOME of the columns ,

write a query to delete duplicate data from cars table.

duplicate record is identified based on model and brand

\*/

--> Solution 1:- Delete using unique identifier

delete

--select \*

from cars where id in (

select max(id) from cars group by model, brand having count(\*) > 1);

--> Solution 2:- Using Self join

delete

--select \*

from cars where id in (

select c2.id from cars c1 join cars c2 on c1.model = c2.model and c1.brand = c2.brand and c1.id<c2.id)

--> Solution 3:- Window Function

delete

--select \*

from cars where id in (select id from (

select \*, row\_number() over (partition by model, brand order by make) rn from cars ) x where x.rn>1);

--> Solution 4:- Using Min function-- this even deletes multiple records

delete

--select \*

from cars where id not in (

select min(id) from cars group by model, brand );

--> Solution 5:- Using Backup Table , not very effective for prod tables, table availabitlity and permissions may be lost

create table cars\_backup from

select \* from cars where id in (select min(id) from cars group by model, brand ));

--drop table cars ;

alter table rename cars\_backup rename to cars;

--> Solution 6:- Using backup table, without dropping actual table, same as above, not very effective.

create table cars\_backup from

select \* from cars where id in (select min(id) from cars group by model, brand ));

truncate table cars;

insert into cars select \* from cars\_backup ;

/\*Scenario -2: Data is duplicated based on all columns

write a query to delete a duplicate entry for a car in cars table \*/

insert into cars values (1, 'Model S', 'Tesla', 'Blue', 2018),

(4, 'ioniq 5', 'Hyundai', 'White', 2021) ;

select \* from cars order by id;

--> Solution 1:- Delete using CTID

select \*, rowid from cars;

delete from cars where rowid in (select max(rowid) from cars group by model, brand having count(\*)>1);

--this is similar to delete using MIN fn or unique id, only difference is that rowid is specific to Oracle,

--and CTID can be used in postgre sql, but no such column in SQLServer and MYSQL. Solution-2 will work across DBs.

--> Solution 2:- By creating temporary unique id column

alter table cars add row\_num int IDENTITY (1,1); --for sql server

--alter table cars add column row\_num int generated always as IDENTITY; for other dbs

select \* from cars;

delete from cars

where row\_num in (

select max(row\_num) from

cars group by model, brand having count(row\_num)> 1

);

--dropping the additionally created column after delete

alter table cars drop column row\_num;

--> Solution 3:- by creating a backup table

create table cars\_bkp as

select distinct \* from cars;

drop cars;

alter table rename cars\_bkp to cars ;

--> Solution 4:- by creating a backup table but not dropping the original table

create table cars\_bkp as

select distinct \* from cars;

truncate table cars;

insert into cars select \* from cars\_bkp;

drop table cars\_bkp;

/\*--------------------------------------------------------------------------\*/

/\* find actual distance each car travelled corresponding to each day \*/

create table cars\_distance (cars varchar(10), days varchar(10), cumulative\_distance int);

insert into cars\_distance values ('car1','day1',50),

('car1','day2',100),

('car1','day3',200),

('car2','day1',0),

('car3','day1',0),

('car3','day2',50),

('car3','day3',50),

('car3','day4',100);

/\* Expected O/P:

cars days cumulative\_distace actual\_distance

car1 day1 50 50

car1 day2 100 50

car1 day3 200 100

car2 day1 0 0

car3 day1 0 0

car3 day2 50 50

car3 day3 50 0

car3 day4 100 100

\*/

select \*, cumulative\_distance - lag(cumulative\_distance,1,0) over(partition by cars order by days) actual\_distance

from cars\_distance;

A screenshot of a computer

Description automatically generated

/\*--------------------------------------------------------------------------\*/

/\*--------------------------------------------------------------------------\*/

/\*input to output

display given data in below format

Source Destination Distance

Bangalore Hyderabad 400

Chennai Pune 400

Mumbai Delhi 400

\*/

Create table src\_dest\_table (source varchar(20), destination varchar(20), distance int);

insert into src\_dest\_table values ('Bangalore','Hyderabad',400),

('Hyderabad','Bangalore',400),

('Chennai','Pune',400),

('Pune','Chennai',400),

('Mumbai','Delhi',400),

('Delhi','Mumbai',400);

select source, destination, distance from (

select \*, ntile(2) over (ORDER BY source) route\_group from src\_dest\_table) s

where route\_group = 1;

--here we got only one set but not exact output source and destination for Mumbai to Delhi

with cte as

(select \*, ROW\_NUMBER () over (order by distance) rn from src\_dest\_table)

select a.source, a.destination, a.distance from cte as a join cte b

on a.rn < b.rn

and a.source = b.destination

and a.destination = b.source;

/\*--------------------------------------------------------------------------\*/

/\* ungroup the given input data

expected output is that each item should be displayed total\_count times along with its id, in separate record\*/

create table travel\_items (id int, item\_name varchar(20), total\_count int);

insert into travel\_items values (1, 'water bottle', 2),

(2,'tent', 1), (3, 'apple', 4);

insert into travel\_items values (4, 'chairs',6);

with cte as

(

select id, item\_name, total\_count, 1 iter from travel\_items

union all

select id,item\_name, total\_count, iter+1 iter from cte where total\_count > iter)

select id, item\_name, iter from cte order by id;

A table with numbers and letters

Description automatically generated

/\* Derive IPL Matches

There are 10 IPL teams

1) Write a query such that each team plays with every other team only once

2) Write a query such that each team plays with every other team twice \*/

create table ipl\_teams(team\_code varchar(5), team\_name varchar(50));

insert into ipl\_teams values ('MI', 'Mumbai Indians'),

('CSK', 'Chennai SuperKings'),

('KKR', 'Kolkatha Knight Riders'),

('SRG', 'Sunrisers Hyderabad'),

('DC', 'Delhi Capitals'),

('PBKS', 'Punjab Kings'),

('GT', 'Gujrat Titans'),

('RCB', 'Royal Challengers Bengalooru'),

('RR', 'Rajasthan Royals'),

('LSG', 'Lucknow Super Giants');

--Solution-1: one time with each other, total 45 teams

select \* from ipl\_teams order by team\_name;

with cte1 as

(select team\_code, team\_name,

row\_number() over (order by team\_code) rn from ipl\_teams)

select concat(t1.team\_name, ' Vs ', t2.team\_name) matches from cte1 t1 join cte1 t2 on

t1.rn > t2.rn

order by matches ;

--Solution-2: two times with each other, total 10 teams, so 90 mathes.

select concat(t1.team\_name, ' Vs ', t2.team\_name) matches from ipl\_teams t1 join ipl\_teams t2 on

t1.team\_name != t2.team\_name

order by matches ;

--alternate approach based on query 1

with cte1 as

(select team\_code, team\_name,

row\_number() over (order by team\_code) rn from ipl\_teams)

select concat(t1.team\_name, ' Vs ', t2.team\_name) matches from cte1 t1 join cte1 t2 on

t1.rn <> t2.rn

order by matches ;

/\*Pizza Delivery Status

A pizza company is taking orders from customers, and each pizza ordered is added to their database as a separate order

Each order has an associated status "Created", "Submitted", or "Delivered"

An order's final status is calculated based on status as follows:

1) When all orders of a customer have a status of "Delivered" then Final status of the customer is "Completed"

2) when a customer has some orders "Delivered" and some orders "Submitted" then final status is "WIP"

3) when all custumers orders are "submitted" then final status is "Awaiting progress"

4) Otherwise final status is "Awaiting Submission"

Write a query of each customer and his final\_status order by customer name

\*/

create table pizza\_delivery (cust\_name varchar(20), order\_id varchar(10), status varchar(20));

insert into pizza\_delivery values ('John','J1','DELIVERED'),

('John','J2','DELIVERED'),

('Smith','S1','SUBMITTED'),

('David','D1','SUBMITTED'),

('David','D2','DELIVERED'),

('David','D3','CREATED'),

('Krish','K1','CREATED');

select \* from pizza\_delivery;

select distinct cust\_name, 'COMPLETED' as Final\_Status from pizza\_delivery T1

where T1.status = 'DELIVERED'

and not exists

( select \* from pizza\_delivery T2 where status in ('SUBMITTED','CREATED')

and T1.cust\_name = T2.cust\_name)

union

select distinct cust\_name, 'IN PROGRESS' as Final\_Status from pizza\_delivery T1

where T1.status = 'DELIVERED'

and exists

( select \* from pizza\_delivery T2 where status in ('SUBMITTED','CREATED')

and T1.cust\_name = T2.cust\_name)

union

select distinct cust\_name, 'AWAITING PROGRESS' as Final\_Status from pizza\_delivery T1

where T1.status = 'SUBMITTED'

and not exists

( select \* from pizza\_delivery T2 where status in ('DELIVERED','CREATED')

and T1.cust\_name = T2.cust\_name)

union

select distinct cust\_name, 'AWAITING SUBMISSION' as Final\_Status from pizza\_delivery T1

where T1.status = 'CREATED'

and not exists

( select \* from pizza\_delivery T2 where status in ('DELIVERED','SUBMITTED')

and T1.cust\_name = T2.cust\_name)

Finding Average distance of routed between two cities:

A screenshot of a data sheet

Description automatically generated

create table src\_dest\_table\_1 (source varchar(20), destination varchar(20), distance int);

insert into src\_dest\_table\_1 values

('A','B',21),

('B','A',18),

('A','B',19),

('C','D',15),

('C','D',17),

('D','C',16),

('D','C',18);

with cte as

(select

row\_number () over (order by source) id, source, destination, sum(distance) total\_distance,count(\*) route\_count

from src\_dest\_table\_1

group by source, destination)

select c1.source, c1.destination,

cast ((c1.total\_distance+c2.total\_distance)\*1.0/(c1.route\_count+c2.route\_count) as decimal(10,2)) average\_distance

from cte c1 join cte c2

on c1.id < c2.id

and c1.source = c2.destination

Google Interview Question:

create table google\_user\_activity (username varchar(20), activity varchar(20), startdate date, enddate date);

insert into google\_user\_activity values

('Amy','Travel','2020-02-12','2020-02-20'),

('Amy','Dancing','2020-02-21','2020-02-23'),

('Amy','Travel','2020-02-24','2020-02-28'),

('Joe','Travel','2020-02-11','2020-02-18'),

('Adam','Travel','2020-03-10','2020-03-20'),

('Adam','Dancing','2020-03-21','2020-03-23'),

('Adam','Singing','2020-03-24','2020-03-28'),

('Adam','Music','2020-03-29','2020-03-31');

In the above table we need to fin second most activity for each user, if user has only activity, then return that one activity instead of second most recent activity.

Solution-1:

with cte as

(select row\_number() over (partition by username order by startdate) row\_id, count(\*) over(partition by username ) group\_count,

\* from google\_user\_activity)

select distinct cte.username, cte.activity, cte.startdate, cte.enddate

from cte where row\_id = case when group\_count=1 then 1 else group\_count-1 end

Solution-2:

with cte as

(select row\_number() over (partition by username order by startdate) row\_id, count(\*) over(partition by username ) group\_count,

\* from google\_user\_activity),

cte2 as (select case when group\_count=1 then row\_id

else group\_count-1

end id, username from cte)

select distinct cte.username, cte.activity, cte.startdate, cte.enddate

from cte , cte2 where cte.row\_id = cte2.id and cte.username = cte2.username ;

create table account\_transactions (account\_num varchar(20), transaction\_date date, debit\_credit varchar(10), amount int);

insert into account\_transactions values

('acc\_1','2024-01-20','credit',100),

('acc\_1','2024-01-21','credit',500),

('acc\_1','2024-01-22','credit',300),

('acc\_1','2024-01-23','credit',200),

('acc\_2','2024-01-20','credit',500),

('acc\_2','2024-01-21','credit',1100),

('acc\_2','2024-01-22','debit',1000),

('acc\_3','2024-01-20','credit',1000),

('acc\_4','2024-01-20','credit',1500),

('acc\_4','2024-01-21','debit',500),

('acc\_5','2024-01-20','credit',900);

--write a query to display account\_no and transaction date when the account has reached 1000.

--Include only those accounts whose current value > 1000

with cte1 as

(select account\_num,

sum(case when debit\_credit = 'credit' then amount else amount\*-1 end) current\_value from account\_transactions

group by account\_num),

cte2 as

(select account\_num, transaction\_date, amount, sum(amount) over (partition by account\_num order by transaction\_date

range between unbounded preceding and current row) cumil\_amount

from account\_transactions

where account\_num in (select account\_num from cte1 where current\_value >= 1000))

select account\_num, min(transaction\_date) from cte2 where cumil\_amount >= 1000

group by account\_num

A screenshot of a computer

Description automatically generated

create table billing(cust\_id int, cust\_name varchar(20), billing\_id varchar(4), billing\_creation\_date date, billing\_amount varchar(10));

insert into billing values

(1,'A','id1','10-10-2020','100$'),

(1,'A','id2','11-11-2020','150$'),

(1,'A','id3','12-11-2021','100$'),

(2,'B','id4','10-11-2019','150$'),

(2,'B','id5','11-11-2020','200$'),

(2,'B','id6','12-11-2021','250$'),

(3,'C','id7','01-01-2018','100$'),

(3,'C','id8','05-01-2019','250$'),

(3,'C','id9','06-01-2021','300$');

--Display average billing amount for each customer between 2019 and 2021,

--assume 0$ billing amount if nothing is billed for a particular year of that customer

with cte1 as

(select cust\_id, cust\_name, year(billing\_creation\_date) billing\_year,

cast(replace(billing\_amount,'$','') as int) as billing\_amount from billing),

cte2 as

(select cust\_id, cust\_name,

sum (case when billing\_year = 2019 then billing\_amount else 0 end) bill\_2019,

sum (case when billing\_year = 2020 then billing\_amount else 0 end) bill\_2020,

sum (case when billing\_year = 2021 then billing\_amount else 0 end) bill\_2021,

count (case when billing\_year = 2019 then billing\_amount else NULL end) count\_2019,

count (case when billing\_year = 2020 then billing\_amount else NULL end) count\_2020,

count (case when billing\_year = 2021 then billing\_amount else NULL end) count\_2021

from cte1

group by cust\_id, cust\_name )

select cust\_id, cust\_name,

concat(cast ((bill\_2019+bill\_2020+bill\_2021)\*1.0 /(case when count\_2019 = 0 then 1 else count\_2019 end

+ case when count\_2020 = 0 then 1 else count\_2020 end

+ case when count\_2021 = 0 then 1 else count\_2021 end ) as decimal (10,2) ),'$') average\_count

from cte2

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

select \* from position

select \* from employez

insert into employez values

(1,'John Smith',1),

(2,'Jane Doe',2),

(3,'Michael Brown',2),

(4,'Emily Johnson',2),

(5,'William Lee',3),

(6,'Jessica Clark',3),

(7,'Christopher Harris',3),

(8,'Olivia Wilson',3),

(9,'Daniel Martinez',3),

(10,'Sophia Miller',3);

with cte1 as

(select id, title, groups, levels, payscale, totalpost, count(id) over(partition by title order by title) filled\_postitions

from position

union all

select id, title, groups, levels, payscale,totalpost, filled\_postitions+1

from cte1

where filled\_postitions < totalpost

),

cte2 as

(select id, title, groups, levels, payscale, row\_number () over (partition by title order by id) group\_num from cte1 ),

cte3 as

(select \*, row\_number () over (partition by position order by name) group\_num from employez),

cte4 as

(select cte2.id, title, groups, levels, payscale, coalesce(name, 'Vacant') name from cte2 left join cte3

on cte2.group\_num = cte3.group\_num

and cte2.id = cte3.position)

select title, groups, levels, payscale, name from cte4 order by id

A screenshot of a computer

Description automatically generated

A close up of text

Description automatically generated

create table candidates (id int, gender varchar(1), age int, party varchar(20));

insert into candidates values

(1,'M',55,'Democratic'),

(2,'M',51,'Democratic'),

(3,'F',62,'Democratic'),

(4,'M',60,'Republic'),

(5,'F',61,'Republic'),

(6,'F',58,'Republic');

create table results (constituency\_id int, candidate\_id int, votes int);

insert into results values

(1,1,847529),

(1,4,283409),

(2,2,293841),

(2,5,393485),

(3,3,429084),

(3,6,303890);

/\*Expected Output:

Democratic 2

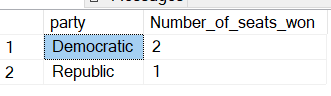
Republic 1 \*/

with cte1 as

(select r.\*,c.party, rank() over (partition by constituency\_id order by votes desc) position from results r

join candidates c on r.candidate\_id = c.id)

select party, sum(position) Number\_of\_seats\_won from cte1 where position=1 group by party



A white text on a white background

Description automatically generated

create table customers (id int, first\_name varchar(20), last\_name varchar(20));

insert into customers values

(1,'Carolyn','O''Lunny'),

(2,'Matteo','Husthwaite'),

(3,'Malessa','Rowseby');

create table campaigns (id int, customer\_id int, name varchar(30));

insert into campaigns values

(2,1,'Overcoming Challenges'),

(4,1,'Business Rules'),

(3,2,'YUI'),

(1,3,'Quantitative Finance'),

(5,3,'MMC');

create table events (campaign\_id int, status varchar(10));

insert into events values

(1,'Success'),

(1,'Success'),

(2,'Success'),

(2,'Success'),

(2,'Success'),

(2,'Success'),

(2,'Success'),

(3,'Success'),

(3,'Success'),

(3,'Success'),

(4,'Success'),

(4,'Success'),

(4,'Failure'),

(4,'Failure'),

(5,'Failure'),

(5,'Failure'),

(5,'Failure'),

(5,'Failure'),

(5,'Failure'),

(5,'Failure'),

(4,'Success'),

(5,'Success'),

(5,'Success'),

(1,'Failure'),

(1,'Failure'),

(1,'Failure'),

(2,'Failure'),

(3,'Failure');

/\*Expected Output:

event\_type Customer Campaign Total

Success Carolyn O'Lunny Business Rules, Overcoming Challenges 8

Failure Malessa Rowesby MMC, Quantitative Finance 9

\*/

select \* from customers;

select \* from campaigns;

select \* from events;

with cte1 as

(select

e.campaign\_id,

e.status event\_type, concat(c.first\_name,' ', c.last\_name) Customer, cp.name Campaign

,row\_number() over (partition by concat(c.first\_name,' ', c.last\_name),e.status order by campaign\_id ) Total\_Campaigns

from events e

join campaigns cp on e.campaign\_id=cp.id

join customers c on cp.customer\_id=c.id),

cte2 as

(select \*,max(Total\_Campaigns) over (partition by customer, event\_type order by Customer) max from cte1 ),

cte3 as

(select max(total\_campaigns) maxx,event\_type from cte1

group by event\_type),

cte4 as

(select distinct cte2.event\_type, cte2.customer, cte2.campaign, cte2.max total

from cte2 join cte3 on cte2.max = cte3.maxx )

select event\_type, customer, string\_agg(campaign, ',') campaign, total from cte4

group by event\_type, customer,total

order by event\_type desc ;

----------------------------

with cte1 as

(select

e.status event\_type, concat(c.first\_name,' ', c.last\_name) Customer, cp.name Campaign

--,row\_number() over (partition by concat(c.first\_name,' ', c.last\_name),e.status order by campaign\_id ) Total\_Campaigns

,count(1) as total

from events e

join campaigns cp on e.campaign\_id=cp.id

join customers c on cp.customer\_id=c.id

group by e.status, concat(c.first\_name,' ', c.last\_name), cp.name ),

cte2 as

(select event\_type, customer, STRING\_AGG (Campaign,',') campaign,sum(total) total

from cte1

group by event\_type, customer)

select event\_type, customer, campaign, total from

(select \*, rank() over (partition by event\_type order by total desc) rnk from cte2 ) a

where a.rnk = 1 order by event\_type desc ;

A screenshot of a computer

Description automatically generated

A screenshot of a white text

Description automatically generated

A screenshot of a computer

Description automatically generated

with cte1 as

(select distinct r.\*, concat(c.first\_name,' ',c.last\_name) candidate\_name

, count(1) over (partition by concat(c.first\_name,' ',c.last\_name), state order by concat(c.first\_name,' ',c.last\_name) ) total\_count

from results\_tab r join candidates\_tab c

on c.id= r.candidate\_id),

cte2 as

(select candidate\_name, string\_agg (concat(state,' (', total\_count,')'),',') states ,

rank() over (partition by candidate\_name order by total\_count desc) wins

from cte1

group by candidate\_name,total\_count)

select candidate\_name,

string\_agg(case when wins = 1 then states else '' end,'') '1st\_place',

string\_agg(case when wins = 2 then states else '' end,'') '2nd\_place',

string\_agg(case when wins = 3 then states else '' end,'') '3rd\_place'

from cte2

group by candidate\_name

create table persons (person\_id varchar(2), parent\_id varchar(2), person\_status varchar(10));

insert into persons values

('A' ,'X','Alive'),

('B' ,'Y','Dead'),

('X' ,'X1','Alive'),

('Y' ,'Y1','Alive'),

('X1','X2','Alive'),

('Y1','Y2','Dead');

select \* from persons ;

--print the number of persons with grand parent alive.

with cte as

(select p1.\*, p2.parent\_id grand\_parent\_id from persons p1 join persons p2

on p1.parent\_id = p2.person\_id)

select count(\*) Total\_Persons\_with\_Grand\_Parents\_Alive

--cte.grand\_parent\_id, cte.person\_id, p.person\_status grand\_parent\_status, cte.person\_status

from cte join persons p

on cte.grand\_parent\_id = p.person\_id

where p.person\_status='Alive'

---------

select count(1) Total\_Persons\_with\_Grand\_Parents\_Alive

--c.\*, p.parent\_id, g.parent\_id, g.person\_status grand\_parent\_id

from persons c

join persons p on c.parent\_id = p.person\_id

join persons g on p.parent\_id = g.person\_id

where g.person\_status = 'Alive'

------------

--Display sum(runs) per each over

--actual challenge is to generate data for 20 overs randomly

create table match\_score

(

balls int,

runs int

);

-- Formula for range between 2 numbers. Here we need between 1 & 6. 1 being low number and 6 begin the high number.

-- floor(RANDOM()\*(high\_no - low\_no + 1)) + low\_no;

INSERT INTO match\_score

SELECT value, floor(rand(value\*21378)\*6-1)+1

from generate\_series (1,120);

--since in sqlserver the rand value is coming same number for all 120 entries...

--added 'seed' as value\*21378 (21378 is a random number to get one digit before decimal value

--then selected high\_num as 6, low\_num as 1... and used floor to round off decimals into numbers

select \* from match\_score

SELECT RUNS,COUNT(1) FROM match\_score GROUP BY RUNS;

UPDATE match\_score

SET RUNS = floor(rand()\*(2-1))+1

WHERE RUNS = 5;

SELECT sum(runs) FROM match\_score ;

with cte as

(SELECT \*, ntile(20) over (order by balls) overs FROM match\_score)

select overs, sum(runs) from cte group by overs;

13/Nov/2024

Print ‘Active’ if at least one of the status is ‘Active’, Print ‘Inactive’ if all the entries of a parent are ‘Inactive’.

create table parent\_status (parent\_id int, child\_id int, stats varchar(10));

insert into parent\_status values

(1,3,'Active'),

(1,4,'Inactive'),

(1,5,'Active'),

(1,6,'Inactive'),

(2,7,'Active'),

(2,8,'Inactive'),

(3,9,'Inactive'),

(4,10,'Inactive'),

(4,11,'Active'),

(5,12,'Inactive'),

(5,13,'Inactive');

select \*, rank () over (partition by parent\_id order by stats) rnk

from parent\_status;

with cte1 as

(select distinct parent\_id from parent\_status where stats = 'Active')

select distinct p.parent\_id, case when p.parent\_id in (select parent\_id from cte1) then 'Active' else 'Inactive' end status

from parent\_status p

select distinct p.parent\_id,

case when p.parent\_id in

(select p2.parent\_id from parent\_status p2 where p2.stats='Active')

then 'Acitve' else 'Inactive' end status from parent\_status p

select parent\_id, min(stats) from parent\_status group by parent\_id;

select distinct parent\_id, stats from

(select parent\_id, stats, rank() over (partition by parent\_id order by stats) rnk from parent\_status) a

where a.rnk=1;

A screenshot of a computer

Description automatically generated

create table lift (id int, capacity int);

insert into lift values (1, 300), (2,350);

create table lift\_passengers (passenger\_name varchar(30), weight\_kg int, lift\_id int);

insert into lift\_passengers values

('Rahul',85,1),

('Adarsh',73,1),

('Riti',95,1),

('Dheeraj',80,1),

('Vimal',83,2),

('Neha',77,2),

('Priti',73,2),

('Himanshi',85,2);

--print comma separated list of passengers who can be accommodated in each lift without exceeding its capacity.

--The passenger list should be order by their weight in increasing order.

select lift\_id, string\_agg(passenger\_name, ', ') passengers from

(select p.\*, l.capacity, sum(weight\_kg) over (partition by lift\_id order by weight\_kg

range between unbounded preceding and current row ) cumulative\_weight

from lift\_passengers p join lift l on p.lift\_id = l.id) k

where k.cumulative\_weight <= k.capacity

group by lift\_id

--it gives the same result without unbounded clause

A screenshot of a computer

Description automatically generated

A screenshot of a chart

Description automatically generated

/\*For pairs of brands in the same year (e.g.: apple/samsung/2020 and samsung/apple/2020)

if custom1 = custom3 and custom2 = custom4 then keep only one pair

if custom1 != custom3 and custom2 != custom4 then keep both the pairs

For brands that do not have pairs, keep all the records \*/

create table brands (brand1 varchar (15), brand2 varchar(15), year int, custom1 int, custom2 int, custom3 int, custom4 int);

insert into brands values

('apple','samsung',2020, 1,2,1,2),

('samsung','apple',2020, 1,2,1,2),

('samsung','apple',2021, 1,2,5,3),

('apple','samsung',2021, 5,3,1,2),

('goole','',2021,5,9,null,null),

('oneplus','nothing',2022,5,9,6,3);

with cte as

(select case when brand1>brand2 then concat(brand1,brand2,year) else concat(brand2,brand1,year) end pair, \* from brands ),

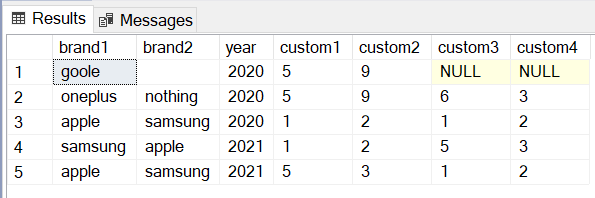
cte1 as

(select \*, lead(pair) over (partition by pair order by pair) next\_rec from cte),

cte2 as

(select \*, case when next\_rec is not NULL and custom1= custom3 and custom2=custom4 then 1 else 0 end dup\_flag from cte1)

select brand1, brand2, year, custom1, custom2, custom3, custom4 from cte2 where dup\_flag !=1 order by year



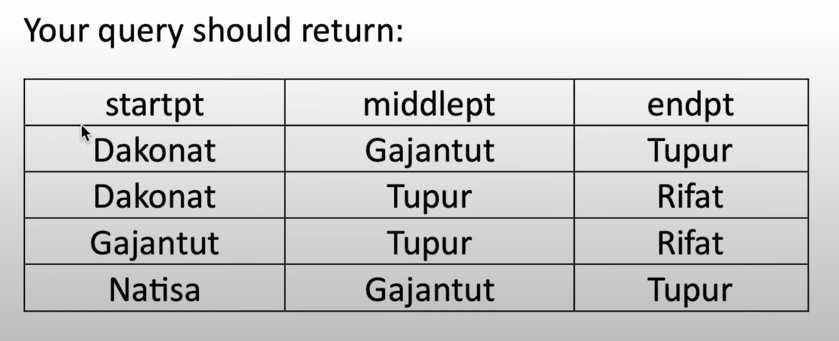
2024-11-21:

A close up of text

Description automatically generated

A white background with black text

Description automatically generated



create table mountain\_huts

(

id integer not null unique,

name varchar(40) not null unique,

altitude integer not null

);

insert into mountain\_huts values (1, 'Dakonat', 1900);

insert into mountain\_huts values (2, 'Natisa', 2100);

insert into mountain\_huts values (3, 'Gajantut', 1600);

insert into mountain\_huts values (4, 'Rifat', 782);

insert into mountain\_huts values (5, 'Tupur', 1370);

drop table if exists trails;

create table trails

(

hut1 integer not null,

hut2 integer not null

);

insert into trails values (1, 3);

insert into trails values (3, 2);

insert into trails values (3, 5);

insert into trails values (4, 5);

insert into trails values (1, 5);

**Solution:**

with cte1 as

(select t1.hut1, m1.name source, m1.altitude altitude1, t1.hut2, m2.name destination, m2.altitude altitude2,

case when m1.altitude > m2.altitude then 1 else 0 end altitude\_flag

from mountain\_huts m1 join

trails t1 on m1.id = t1.hut1

join mountain\_huts m2 on m2.id = t1.hut2 ),

--Now we order the source and destination based on source always being on higher altitude, if not we swap the values.

cte2 as

(select case when altitude\_flag =1 then hut1 else hut2 end hut1,

case when altitude\_flag=1 then source else destination end source,

case when altitude\_flag=1 then altitude1 else altitude2 end altitude1,

case when altitude\_flag=1 then hut2 else hut1 end hut2,

case when altitude\_flag=1 then destination else source end destination,

case when altitude\_flag=1 then altitude2 else altitude1 end altitude2

from cte1 )

select c1.source source, c1.destination mid\_point, c2.destination end\_point

from cte2 c1 join cte2 c2 on c1.destination = c2.source

order by c1.source

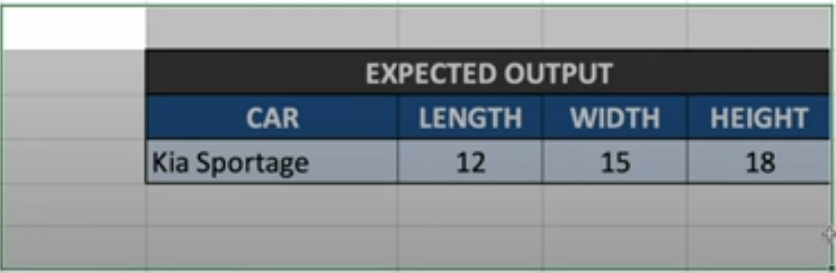
A screenshot of a computer

Description automatically generated

2024-11-22:

A screenshot of a calculator

Description automatically generated



create table footer (id int, car varchar(15), length int, breadth int, height int);

insert into footer values

(1,'Hyundai',15,6,NULL),

(2,'',NULL,NULL,20),

(3,'',12,8,15),

(4,'Toyota',NULL,15,NULL),

(5,'Kia',NULL,NULL,18);

--Solution-1:

select \* from

(select top 1 car from footer where car is not null order by id desc) car

cross join

(select top 1 length from footer where length is not null order by id desc) length

cross join (select top 1 breadth from footer where breadth is not null order by id desc ) breadth

cross join (select top 1 height from footer where height is not null order by id desc ) height ;

Solution-2:

with cte as

(select id, car, length,

case when length is not null then 1 else 0 end length\_flag ,

sum(case when length is not null then 1 else 0 end) over (order by id) length\_segment,

breadth,

case when breadth is not null then 1 else 0 end breadth\_flag ,

sum(case when breadth is not null then 1 else 0 end) over (order by id) breadth\_segment,

height,

case when height is not null then 1 else 0 end height\_flag ,

sum(case when height is not null then 1 else 0 end) over (order by id) height\_segment

from footer)

select top 1 --\*

car, first\_value(length) over (partition by length\_segment order by id) new\_length,

first\_value(breadth) over (partition by breadth\_segment order by id) new\_breadth,

first\_value(height) over (partition by height\_segment order by id) new\_height

from cte

order by id desc

A screenshot of a computer

Description automatically generated

12/02/2024:

A screenshot of a computer

Description automatically generated

drop table if exists salary;

create table salary

(

emp\_id int,

emp\_name varchar(30),

base\_salary int

);

insert into salary values(1, 'Rohan', 5000);

insert into salary values(2, 'Alex', 6000);

insert into salary values(3, 'Maryam', 7000);

drop table if exists income;

create table income

(

id int,

income varchar(20),

percentage int

);

insert into income values(1,'Basic', 100);

insert into income values(2,'Allowance', 4);

insert into income values(3,'Others', 6);

drop table if exists deduction;

create table deduction

(

id int,

deduction varchar(20),

percentage int

);

insert into deduction values(1,'Insurance', 5);

insert into deduction values(2,'Health', 6);

insert into deduction values(3,'House', 4);

drop table if exists emp\_transaction;

create table emp\_transaction

(

emp\_id int,

emp\_name varchar(50),

trns\_type varchar(20),

amount numeric

);

insert into emp\_transaction

select s.emp\_id, s.emp\_name, x.trns\_type

, case when x.trns\_type = 'Basic' then round(base\_salary \* (cast(x.percentage as decimal)/100),2)

when x.trns\_type = 'Allowance' then round(base\_salary \* (cast(x.percentage as decimal)/100),2)

when x.trns\_type = 'Others' then round(base\_salary \* (cast(x.percentage as decimal)/100),2)

when x.trns\_type = 'Insurance' then round(base\_salary \* (cast(x.percentage as decimal)/100),2)

when x.trns\_type = 'Health' then round(base\_salary \* (cast(x.percentage as decimal)/100),2)

when x.trns\_type = 'House' then round(base\_salary \* (cast(x.percentage as decimal)/100),2) end as amount

from salary s

cross join (select income as trns\_type, percentage from income

union

select deduction as trns\_type, percentage from deduction) x;

select \* from salary;

select \* from income;

select \* from deduction;

select \* from emp\_transaction;

select employee, sum(basic) basic, sum(allowance) allowance, sum(others) others, sum(basic+allowance+others) gross

, sum(Insurance) Insurance, sum(Health) Health, sum(House) House, sum(Insurance+Health+House) Total\_Deductions,

sum(basic+allowance+others) - sum(Insurance+Health+House) NetPay from

(select emp\_name EMPLOYEE,

sum(coalesce(case when trns\_type = 'Basic' then amount end,0)) BASIC ,

sum(coalesce(case when trns\_type = 'Allowance' then amount end,0)) ALLOWANCE

, sum(coalesce(case when trns\_type = 'Others' then amount end,0)) OTHERS

,sum(coalesce(case when trns\_type = 'Insurance' then amount end,0)) Insurance ,

sum(coalesce(case when trns\_type = 'Health' then amount end,0)) Health

, sum(coalesce(case when trns\_type = 'House' then amount end,0)) House

from emp\_transaction

group by emp\_name, trns\_type, amount ) a

group by employee

,

(case when trns\_type = 'Basic' then amount end +

case when trns\_type = 'Allowance' then amount end + case when trns\_type = 'Allowance' then amount end )

GROSS

group by emp\_name, trns\_type

A screenshot of a computer

Description automatically generated

12/03/2024:

A screenshot of a computer

Description automatically generated

create table input\_marks (test\_id int, marks int);

insert into input\_marks values

(100,55),

(101,55),

(102,60),

(103,58),

(104,40),

(105,50);

--Output-1:

select test\_id, marks from (

select \*, lag(marks) over (order by test\_id) prev\_marks from input\_marks ) a

where marks > prev\_marks or prev\_marks is NULL

select test\_id, marks from (

select \*, lag(marks,1,0) over (order by test\_id) prev\_marks from input\_marks ) a

where marks > prev\_marks

--Output-2:

select test\_id, marks from (

select \*, lag(marks,1,marks) over (order by test\_id) prev\_marks from input\_marks ) a

where marks > prev\_marks

A screenshot of a computer

Description automatically generated