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

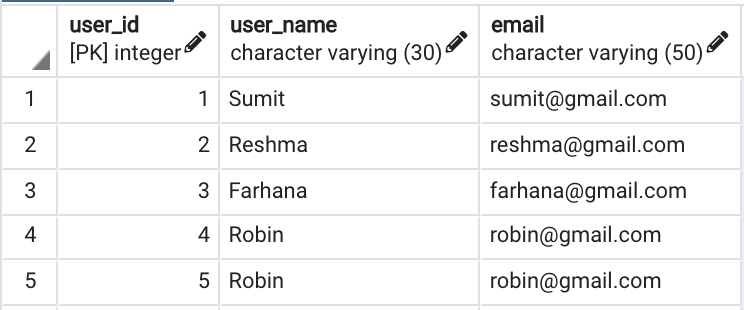
1. **Write a SQL Query to fetch all the duplicate records in a table.**

**Table Name**: TEMP\_USERS

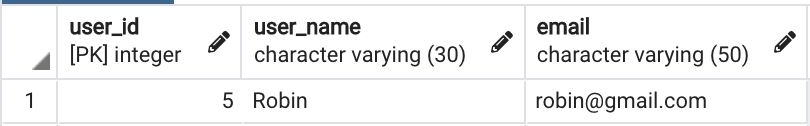
**Note**: Record is considered duplicate if a user name is present more than once.

**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 (

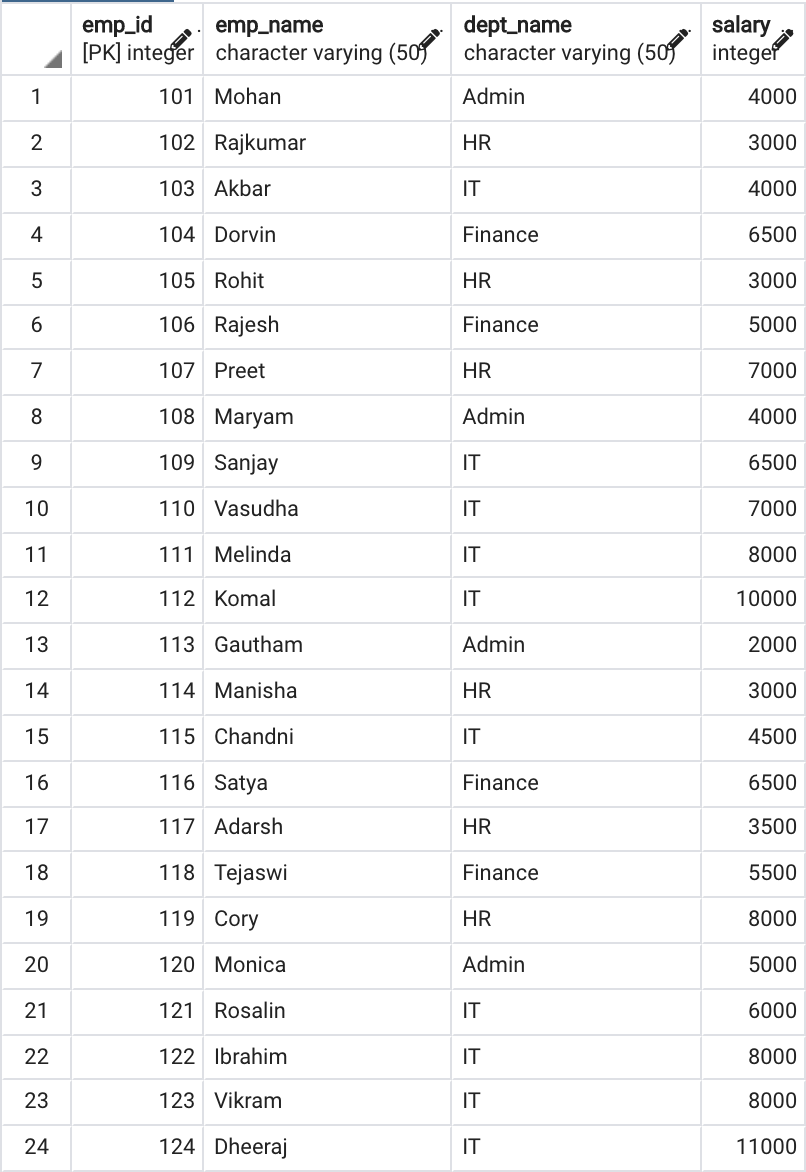
select \*, ROW\_NUMBER() over (partition by user\_name order by user\_id) rn from temp\_user) x

where x.rn > 1;

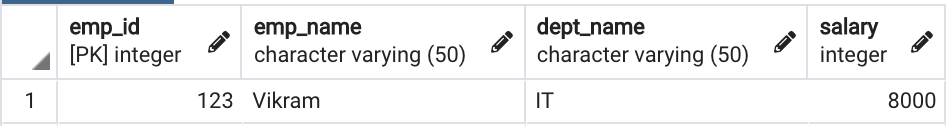
**2. Write a SQL query to fetch the second last record from employee table.**

**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);

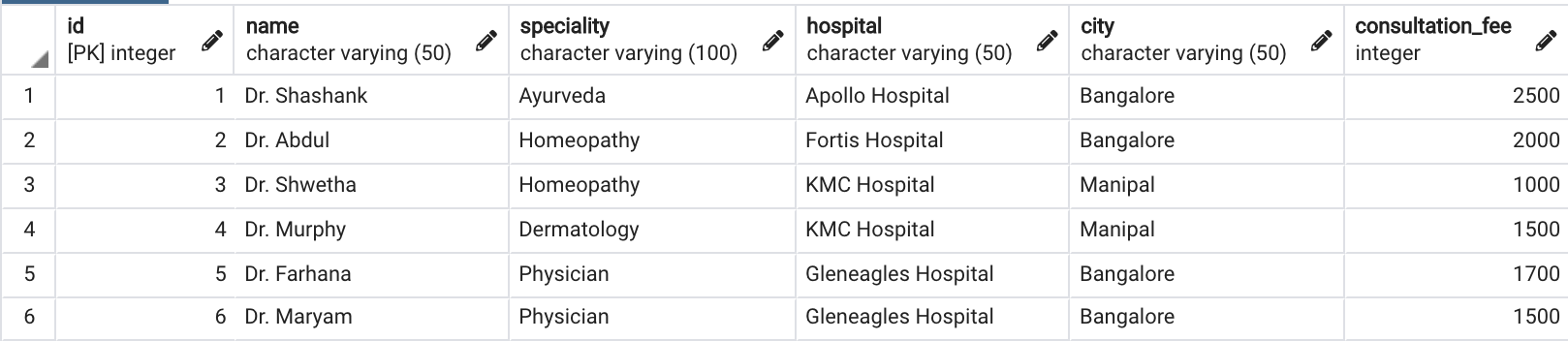
**4. From the doctors table, fetch the details of doctors who work in the same hospital but in different specialty.**

**Table Name**: DOCTORS

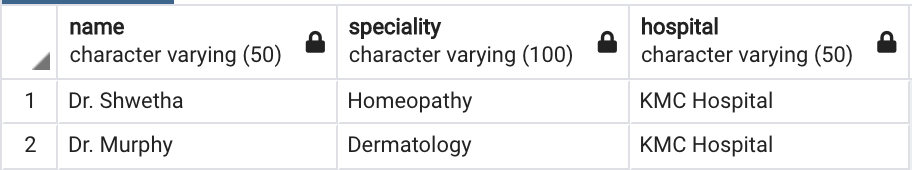
**Approach**: Use self join to solve this problem. Self join is when you join a table to itself.

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

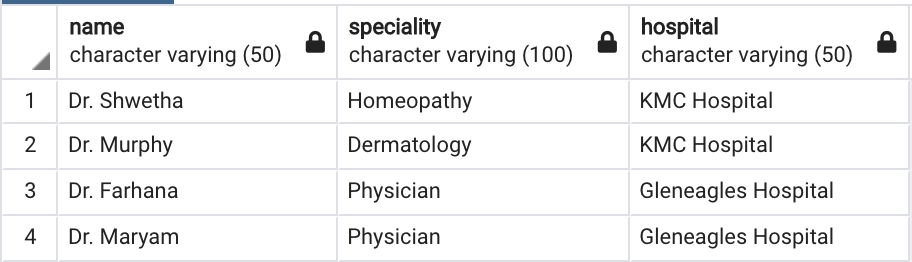
DOCTORS



**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

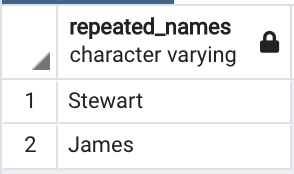
**Approach**: We need to fetch users who have appeared 3 or more times consecutively in login details table. There is a window function which can be used to fetch data from the following record. Use that window function to compare the user name in current row with user name in the next row and in the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

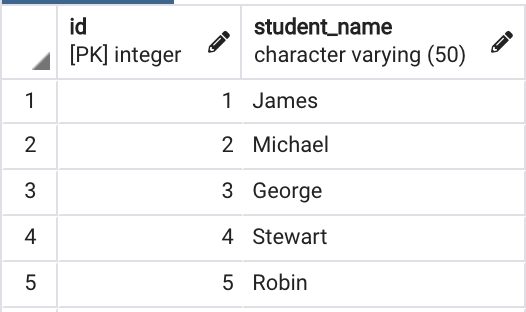
**Note**: If there are no adjacent student then the student name should stay the same.

**Table Name**: STUDENTS

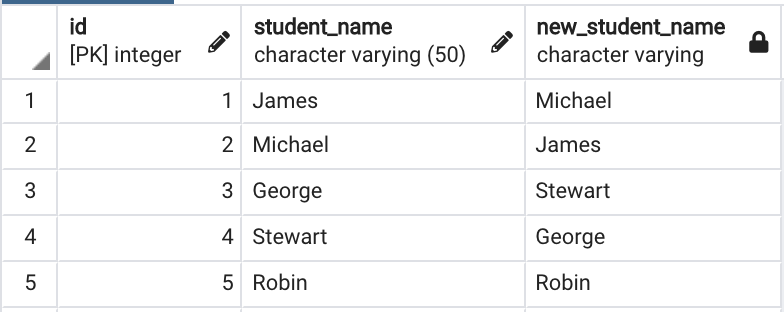
**Approach**: Assuming id will be a sequential number always. If id is an odd number then fetch the student name from the following record. If id is an even number then fetch the student name from the preceding record. Try to figure out the window function which can be used to fetch the preceding the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

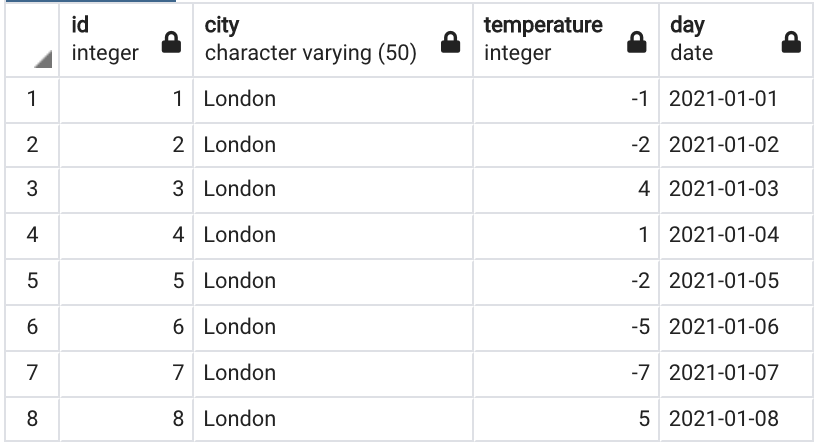
**7. From the weather table, fetch all the records when London had extremely cold temperature for 3 consecutive days or more.**

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

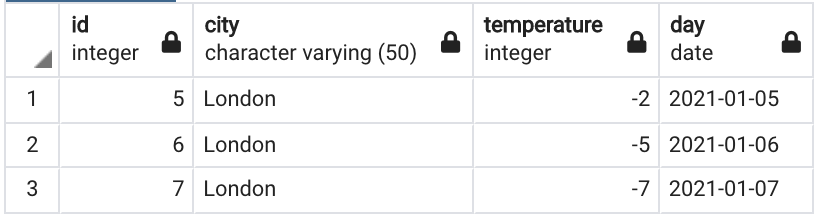
**Table Name**: WEATHER

**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;

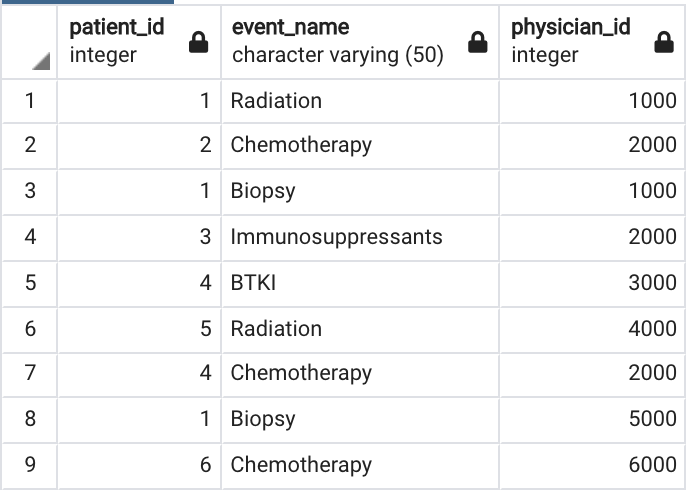
**8. From the following 3 tables (event\_category, physician\_speciality, patient\_treatment), write a SQL query to get the histogram of specialties of the unique physicians who have done the procedures but never did prescribe anything.**

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

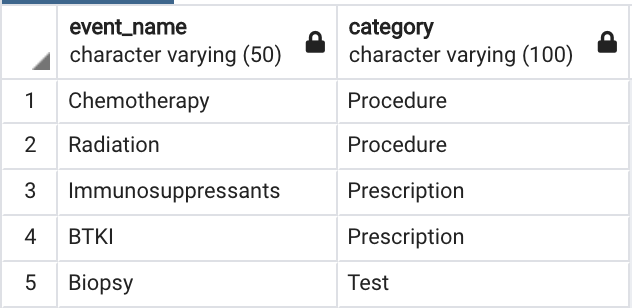
**Approach**: Using the patient treatment and event category table, identify all the physicians who have done “Prescription”. Have this recorded in a sub query.

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.

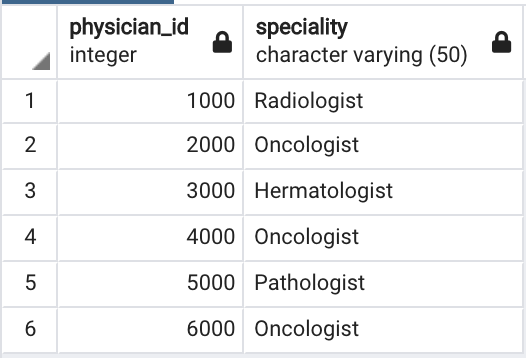
PATIENT\_TREATMENT



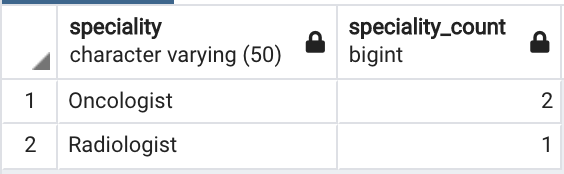
EVENT\_CATEGORY



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.**

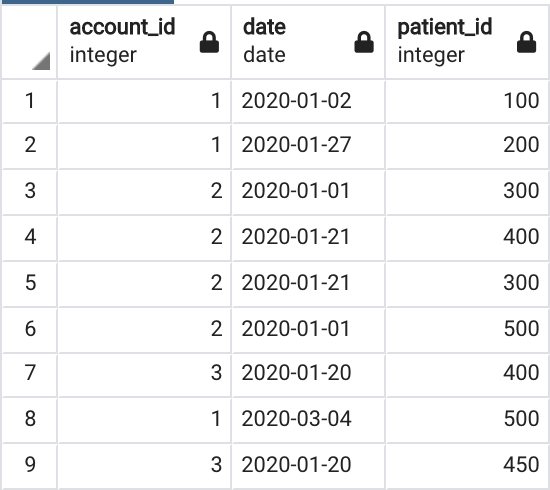
**Note**: Prefer the account id with the least value in case of same number of unique patients

**Table Name**: PATIENT\_LOGS

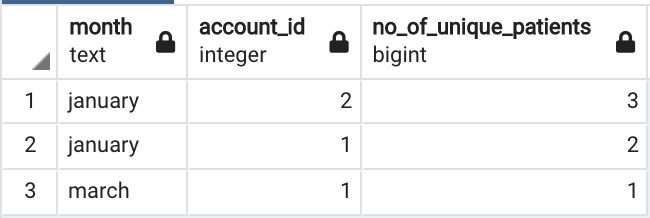
**Approach**: First convert the date to month format since we need the output specific to each month. Then group together all data based on each month and account id so you get the total no of patients belonging to each account per month basis.

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

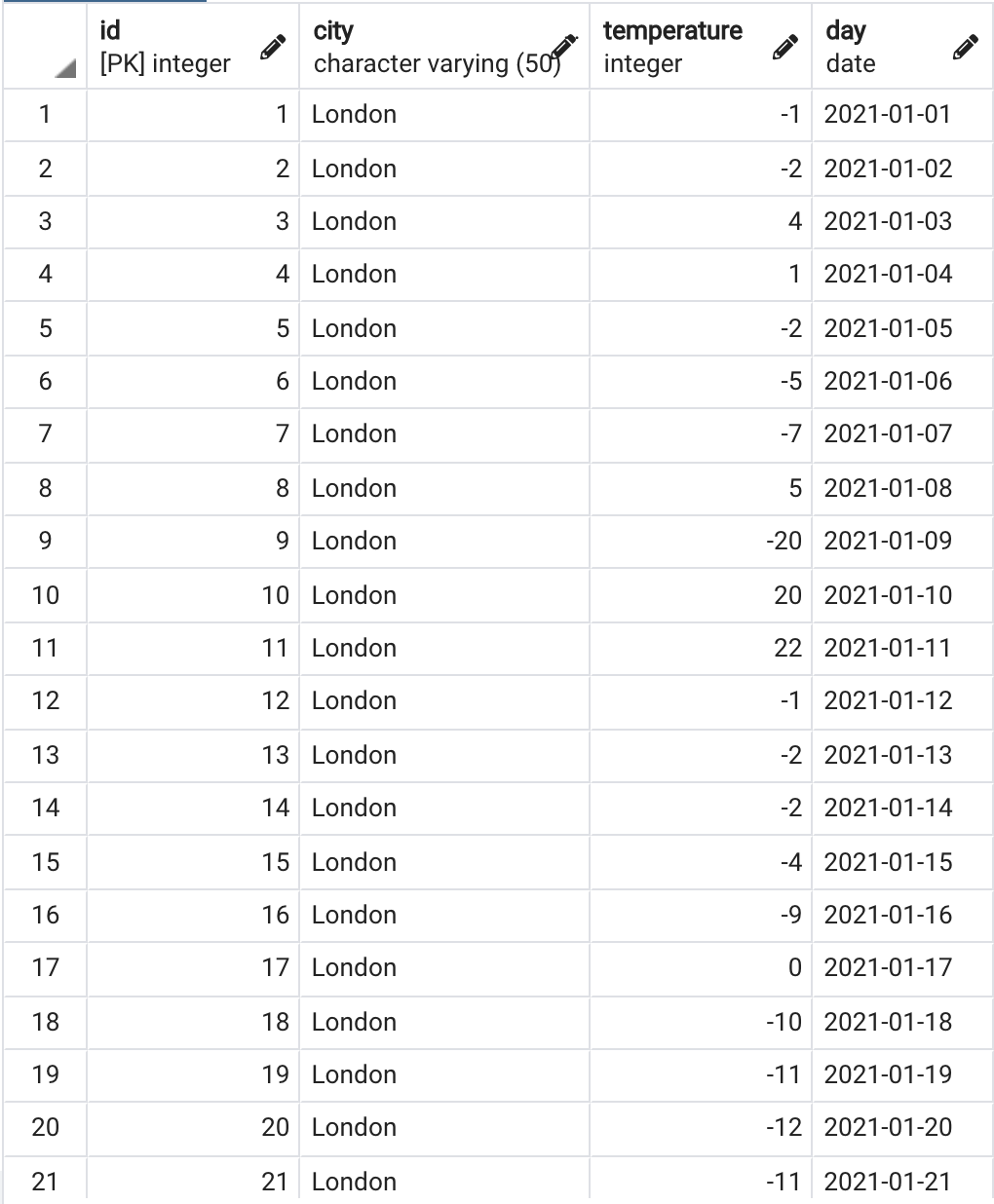
**10. SQL Query to fetch “N” consecutive records from a table based on a certain condition**

Note: Write separate queries to satisfy following scenarios:

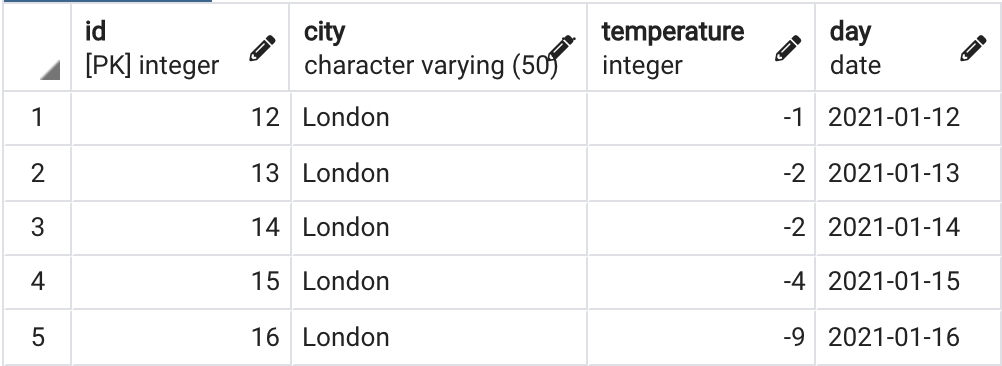
10a. when the table has a primary key

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)

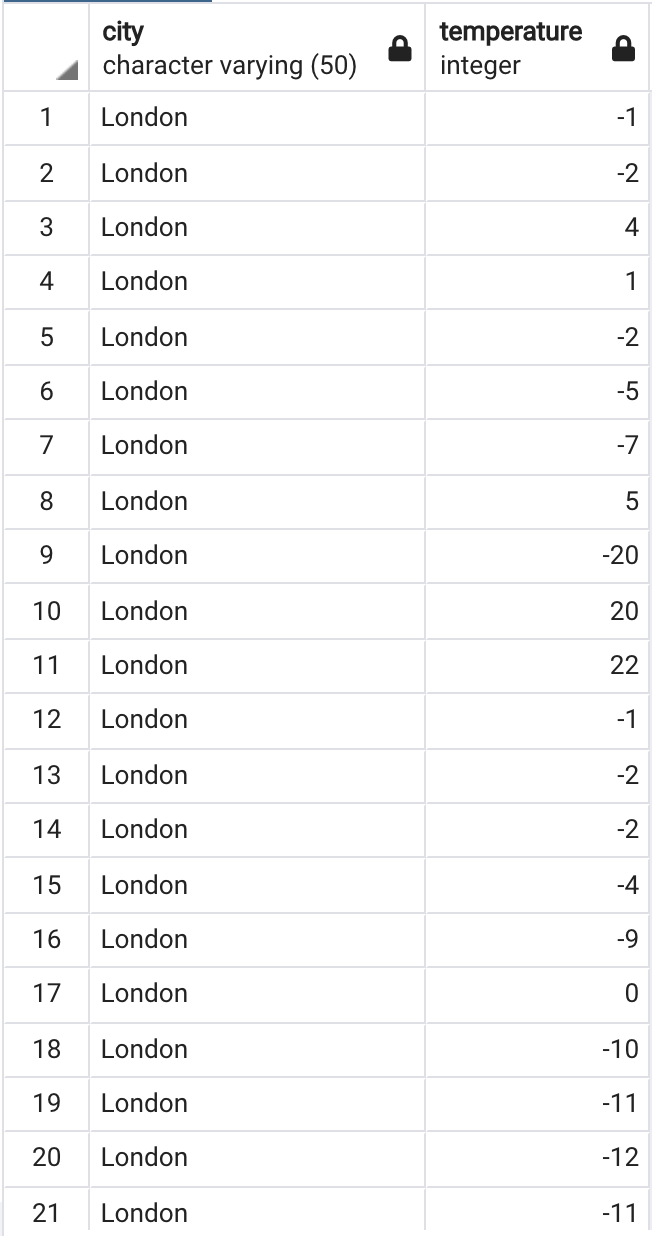
select id, city, temp, recorded\_day from T12

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

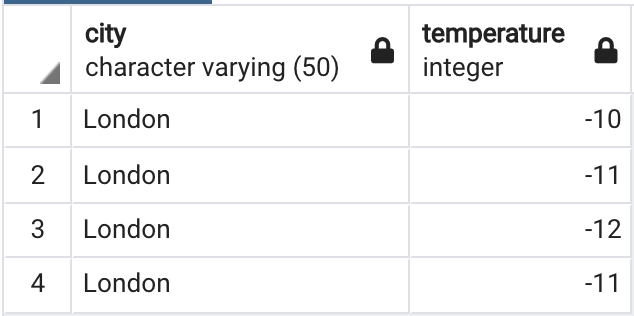
10b. When table does not have a primary key

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

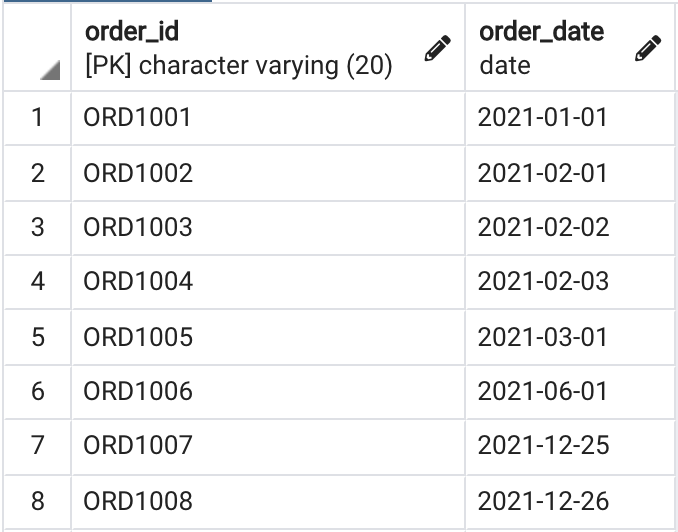
from t2

where t2.cnt = 5;

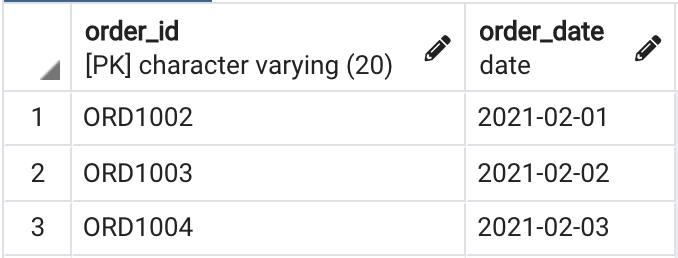
10c. Query logic based on data field

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:

Employee attendence 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)

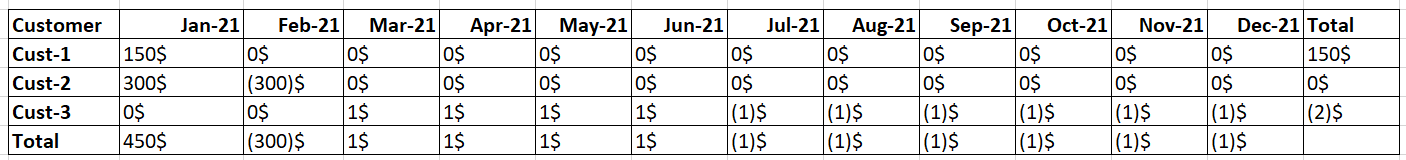
select H2.emp\_name EMP\_NAME, E2.EMP\_NAME MANAGER\_NAME, H2.DESIGNATION, H2.LVL from rec\_emp\_manger H2

join emp\_details E2 on

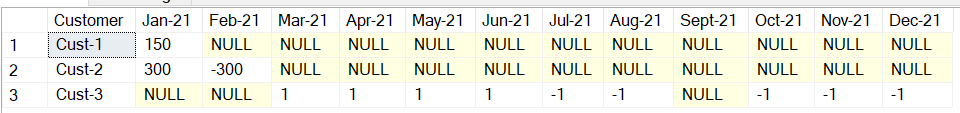
E2.id = H2.manager\_id;

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

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)