\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Practicle\_Questions\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



1. **How to select UNIQUE records from a table using a SQL Query?**

select distinct EMPLOYEE\_ID,NAME,SALARY from employee

select EMPLOYEE\_ID,NAME,SALARY from employee group by EMPLOYEE\_ID

1. **How to delete DUPLICATE records from a table using a SQL Query?**

select \* from employee

select \*,count(\*) from employee group by emp\_id,name,salary having count(\*)>1

delete from employee where emp\_id not in (select min(emp\_id) from employee group by name,salary).

1. **How to read TOP 5 records from a table using a SQL query?**

Consider below table DEPARTMENTS as the source data.



select \* from departments order by Department\_ID asc limit 5

select \* from departments limit 5

1. **How to read LAST 5 records from a table using a SQL query?**

select \* from departments order by Department\_ID desc limit 5

## here last 5 records in ascending order

(select \* from departments order by Department\_ID desc limit 5) order by Department\_ID

1. **How to find the employee with second MAX Salary using a SQL query?**

select max(salary) as second\_max\_salary from employee where salary not in (select max(salary) from employee).

## If we need all the columns too

with

temp as(

select max(salary) as salary from employee where salary not in

(select max(salary) as salary from employee)

)

select a.\* from employee a join temp b on a.salary = b.salary

1. **How to find the employee with third MAX Salary using a SQL query without using Analytic Functions?**

Consider the same EMPLOYEES table as source discussed in previous question

SELECT name, salary FROM Employee e1

WHERE 3-1 =

(SELECT COUNT(DISTINCT salary) FROM Employee e2 WHERE e2.salary > e1.salary)

## with functions

Select Salary from employees order by Salary DESC limit 2,1;

1. **Assume you have the below tables on sessions that users have, and a users table. Write a query to get the active user count of daily cohorts.**



By definition, daily cohorts are active users from a particular day. First, we can use a subquery to get the sessions of new users by day using an inner join with users. This is to filter for only active users by a particular join date for the cohort. Then we can get a distinct count to return the active user count:

with new\_users\_by\_date as(

select s.\* from sessions s join users u on s.user\_id = u.user\_id s.date = u.date)

select date,count(distinct user\_id) as active\_user\_count from new\_users\_by\_date group by date order by asc

1. **Assume you are given the below table on transactions from users for purchases. Write a query to get the list of customers where their earliest purchase was at least $50.**



we can also use the RANK() window function to get the ordering of purchase by customer, and then use that subquery to filter on customers where the first purchase (rank one) is at least 50 dollars. Note that this requires the subquery to include spend as well.

with purchase\_rank as

( select user\_id,spend,rank() over ( partition by user\_id order by transaction\_date asc)

as rank\_ from user\_transactions)

select user\_id,spend from purchase\_rank where rank =1 and spend>=50

1. **Assume you are given the below table on transactions from users. Write a query to get the number of users and total products bought per latest transaction date where each user is bucketed into their latest transaction date.**



First, we need to get the latest transaction date for each user, along with the number of products they have purchased. This can be done in a subquery where we GROUP BY user\_id and take a COUNT(DISTINCT product\_id) to get the number of products they have purchased, and a MAX(transaction\_date) to get the latest transaction date (while casting to a date). Then, using this subquery, we can simply do an aggregation by the transaction date column in the previous subquery, while doing a COUNT() on the number of users, and a SUM() on the number of products:

with latest\_transaction as(

select user\_id, count(distinct product\_id) as product\_num,

max(transaction\_date::Date) as transactions from user\_transaction

group by user\_id)

select user\_transaction, count(user\_id) as num\_users,

sum(num\_products) as total\_products from latest\_transaction group by 1

1. **Assume you are given the below tables on users and their time spent on sending and opening Snaps. Write a query to get the breakdown for each age breakdown of the percentage of time spent on sending versus opening snaps.**

### activities Table:

| **Column Name** | **Type** |
| --- | --- |
| activity\_id | integer |
| user\_id | integer |
| activity\_type | string ('send', 'open', 'chat') |
| time\_spent | float |
| activity\_date | Datetime |

### age\_breakdown Table:

| **Column Name** | **Type** |
| --- | --- |
| user\_id | integer |
| age\_bucket | string ('21-25', '26-30', '31-25') |

We can get the breakdown of total time spent on each activity by each user by filtering out for the activity\_type and taking the sum of time spent. In doing this, we want to do an outer join with the age bucket to get the total time by age bucket for both activity types. This results in the below two subqueries. Then, we can use these two subqueries to sum them by joining on the appropriate age bucket and take the proportion for send time and the proportion for open time per age bucket :

## send activity\_type total time spent on each activity by each user

with send\_timespent as(

select age\_breakdown.age\_bucket, sum(activities.time\_spent) from age\_breakdown

left join on age\_breakdown.user\_id = activities.user\_id

where activity\_type = 'send'

group by 1)

## open activity\_type total time spent on each activity by each user

open open\_timespent as(

select age\_breakdown.age\_bucket, sum(activities.time\_spent) from age\_breakdown

left join on age\_breakdown.user\_id = activities.user\_id

where activity\_type = 'open'

group by 1)

## finally for percentage of time spent on sending versus opening snaps.

select a.age\_bucket,

s.send\_timespent /(s.send\_timespent + o.open\_timespent) as pct\_send,

o.open\_timespent / (s.send\_timespent + o.open\_timespent) as pct\_open,

from age\_breakdown a

left join send\_timespent s on a.age\_bucket = s.age\_bucket

left join open\_timespent o on a.age\_bucket = o.age\_bucket

group by 1

1. **Assume you are given the below table on reviews from users. Define a top-rated place as a business whose reviews only consist of 4 or 5 stars. Write a query to get the number and percentage of businesses that are top-rated places.**



First, we need to get the places where the reviews are all 4 or 5 stars. We can do this using a HAVING clause, instead of a WHERE clause since the reviews need to all be 4 stars or above. For the HAVING condition, we can use a CASE statement that filters for 4 or 5 stars and then take a SUM over them. This can then be compared with the total row count of the particular business\_id reviews to ensure that the count of top reviews matches with the total review count. With the relevant businesses, we can then do an outer join with the original table on business\_id to get a COUNT of distinct business\_id matches, and then the percentage by comparing the COUNT from the top places with the overall COUNT of business\_id :-

with top\_places as( select business\_id from reviews group by 1 having

sum(

case

when review\_stars >=4 then 1 else 0 end) = count(\*)

)

select count(distinct t.business\_id) as top\_places,

count(distinct t.business\_id/count(r.business\_id) as top\_places\_pct

from reviews r

left join top\_places t

on r.business\_id = t.business\_id)

**12.**





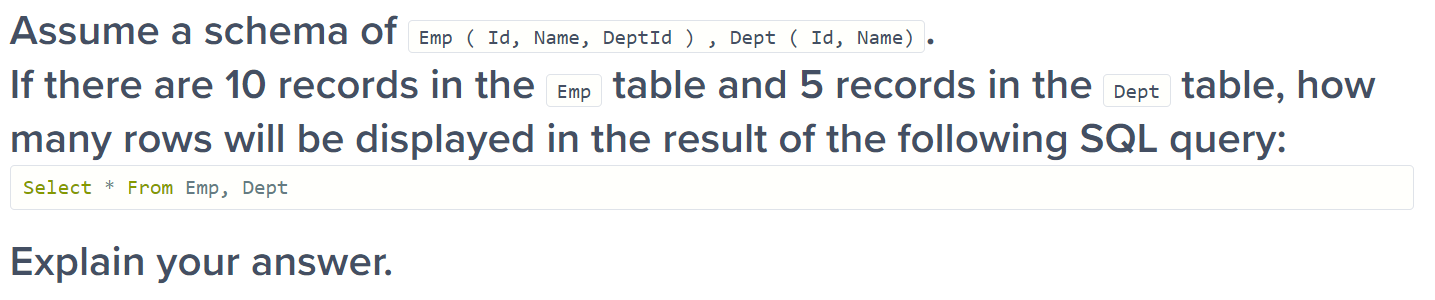


**13.** 



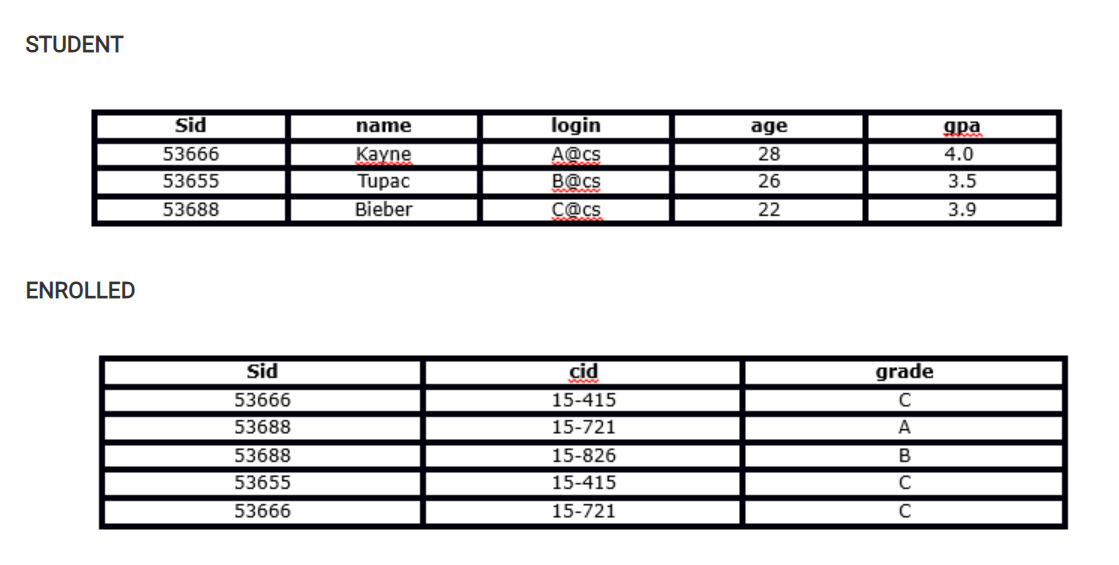


**14.**

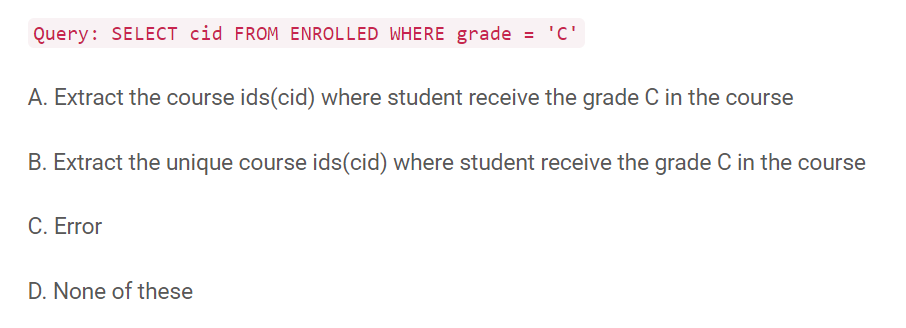


The query will result in 50 rows as a “cartesian product” or “cross join”, which is the default whenever the ‘where’ clause is omitted.

**15.**



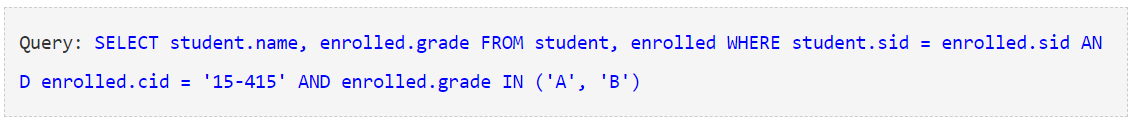
Which of the following is the correct outcome of the SQL query below?



Solution: A

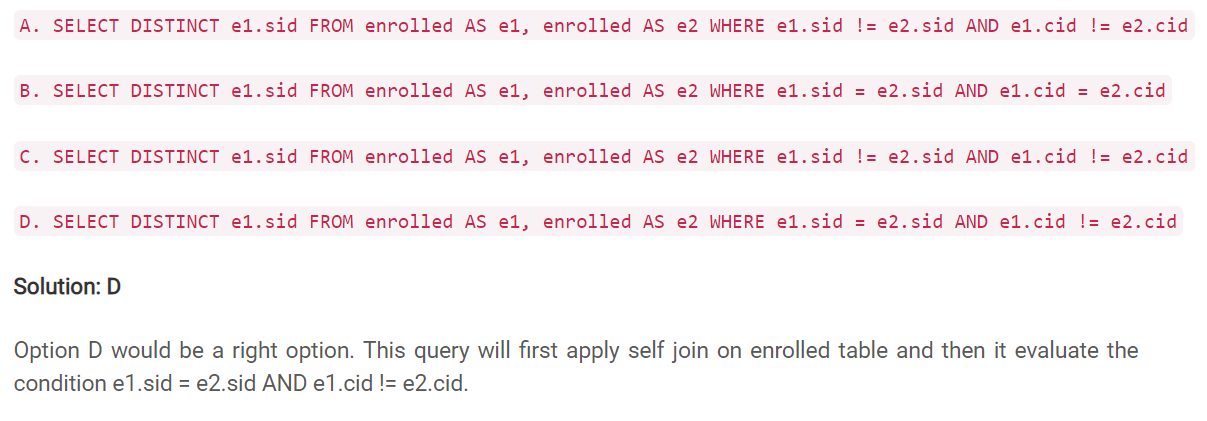
The query will extract the course ids where student receive the grade “C” in the course.

**16.** **What is the correct outcome of the SQL query below?**

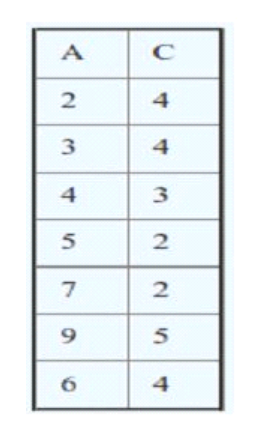


Nothing will be there in output as the above query first joined the ENROLLED and STUDENT tables then it will evaluate the where condition and then it will return the name, grade of the students, those took 15-415 and got a grade ‘A’ or ‘B’ in the course

**17.** Which of the following query will find all the unique students who have taken more than one course?

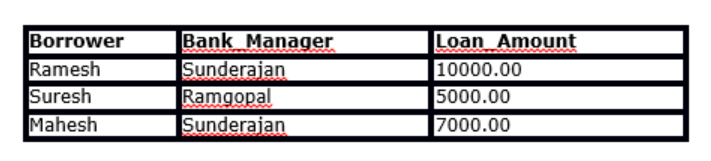


**18**. What are the tuples additionally deleted to preserve reference integrity when the rows (2,4) are deleted from the below table. Suppose you are using ‘ON DELETE CASCADE’.

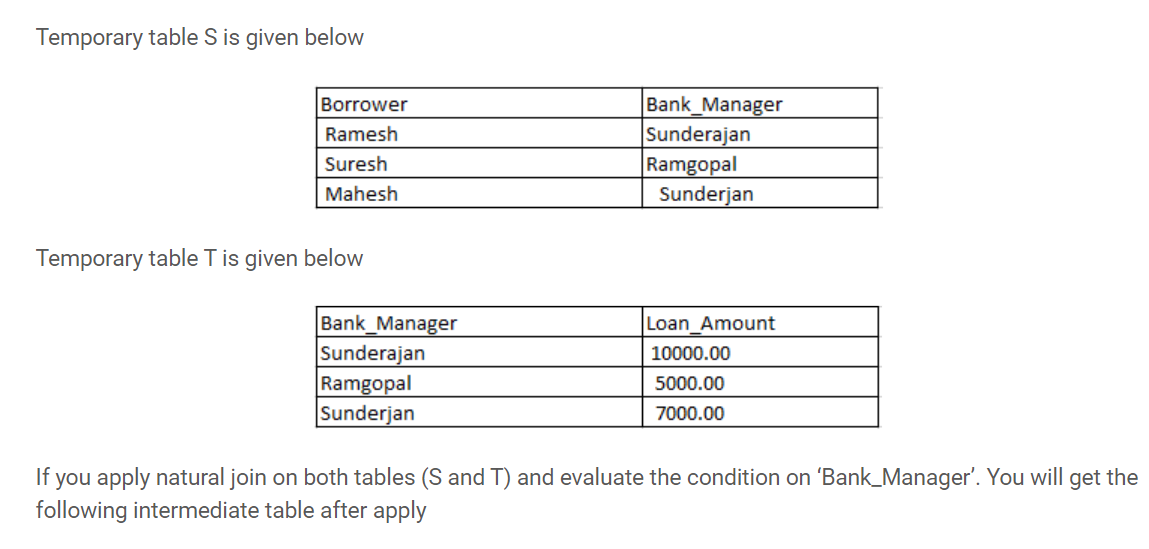


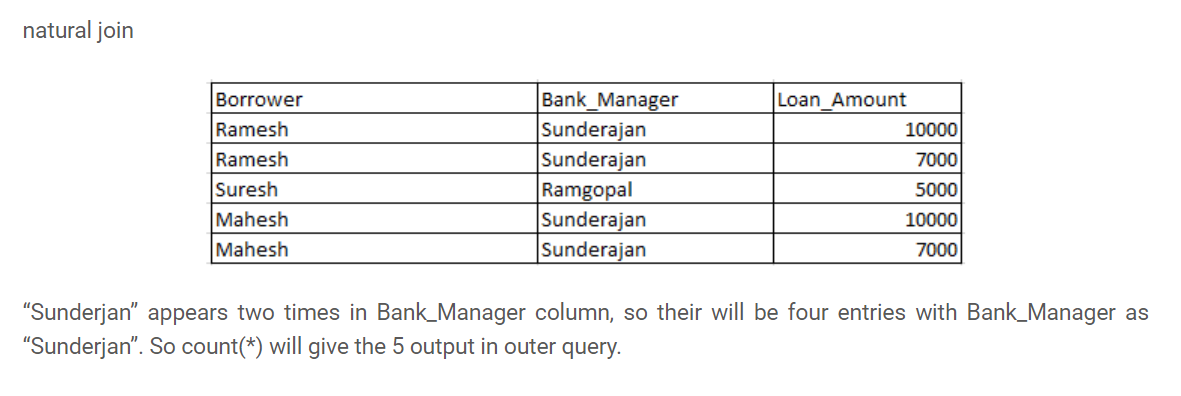
When (2,4) is deleted. Since C is a foreign key referring A with delete on cascade, all entries with value 2 in C must be deleted. So (5, 2) and (7, 2) are deleted. As a result of this 5 and 7 are deleted from A which causes (9, 5) to be deleted.

**19.** Suppose you have a table “Loan\_Records”.



SELECT Count(\*)  FROM  ( (SELECT Borrower, Bank\_Manager FROM   Loan\_Records) AS S NATURAL JOIN (SELECT Bank\_Manager, Loan\_Amount FROM   Loan\_Records) AS T );





**20.What will be the output of the below query?**

Query: SELECT Company, AVG(Salary) FROM AV1 HAVING AVG(Salary) > 1200 GROUP BY Company WHERE Salary > 1000 ;

**There won’t be any output and error will be there.**

**The order should always be like**

**Where**

**Group by**

**Having**

**21. SQL Query to find the second highest salary of Employee**

**select max(salary) as Second\_max\_salary from employees where salary not in (select max(salary) from employees)**

**22. SQL Query to find Max Salary from each department.**

**select department\_id,max(salary) from employees group by department\_id**

**These questions become more interesting if the Interviewer will ask you to print the department name instead of the department id**

**## when we have the department\_name in the same table that as of the employee**

**select job\_id,max(salary) from employees group by job\_id**

**## when we have the department name in another table we need to join the Employee table with Department using the foreign key DeptID, make sure you do**[**LEFT or RIGHT OUTER JOIN**](https://javarevisited.blogspot.com/2013/05/difference-between-left-and-right-outer-join-sql-mysql.html)**to include departments without any employee as well.**

**Select Deptname,max(salary) from employee e left join department d on e. DeptID = d. DeptID group by Deptname.**

**23.** Write SQL Query to display the current date?

SQL has built-in function called GetDate() which returns the current timestamp.

**Select GetDate()**

|  |  |
| --- | --- |
| **Works in:** | SQL Server (starting with 2008), Azure SQL Database, Azure SQL Data Warehouse, Parallel Data Warehouse |

**## In mysql we have another query:-**

**SELECT CURDATE()**

**24.** Write an SQL Query to print the name of the distinct employee who joined between 13-JAN-01 to 03-JAN-06.

select distinct FIRST\_NAME,employee\_id from employees where HIRE\_DATE between ‘13-JAN-01’ and ‘03-JAN-06’.

**25.** Write an SQL Query to find an employee whose salary is equal to or greater than 10000.

select FIRST\_NAME from employees where salary >= 10000

**26.** Write SQL Query to find duplicate rows in a employees? and then write SQL query to delete them?

select FIRST\_NAME,count(\*) as count from employees group by EMPLOYEE\_ID having count(EMPLOYEE\_ID) >1

**select FIRST\_NAME,LAST\_NAME,count(\*) as count from employees group by LAST\_NAME having count(LAST\_NAME) >1**

**#### For deletion of records**

**delete from employees where EMPLOYEE\_ID not in (select max(EMPLOYEE\_ID) from employees group by EMPLOYEE\_ID)**

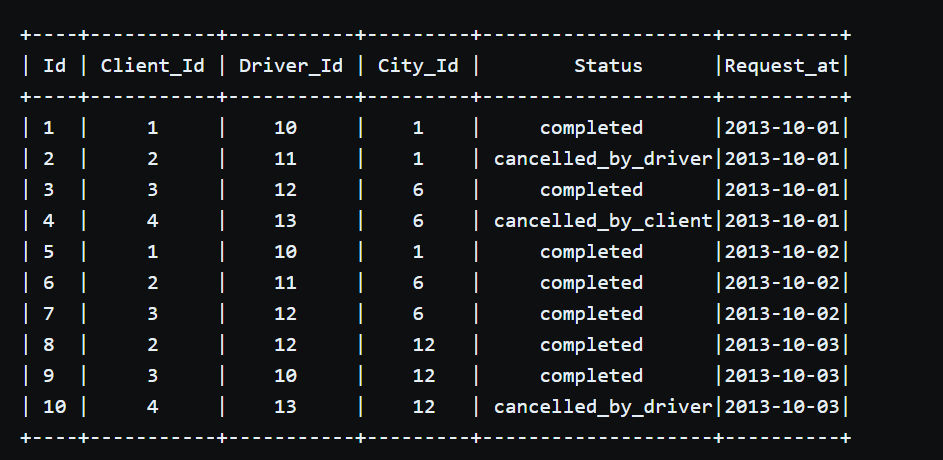
**27. How do you find all employees who are also managers?**

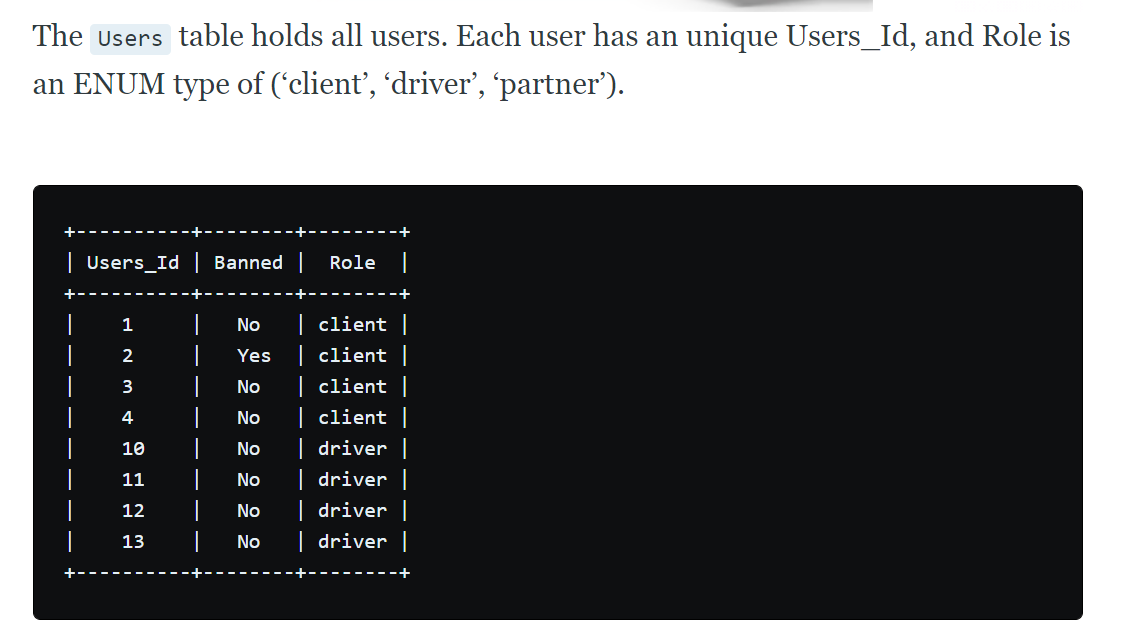
**select e.FIRST\_NAME as Employee\_Name, m.FIRST\_NAME as Manager\_name from employees e, employees m where e.MANAGER\_ID = m.EMPLOYEE\_ID**

One follow-up is to modify this query to include employees which don't have a manager

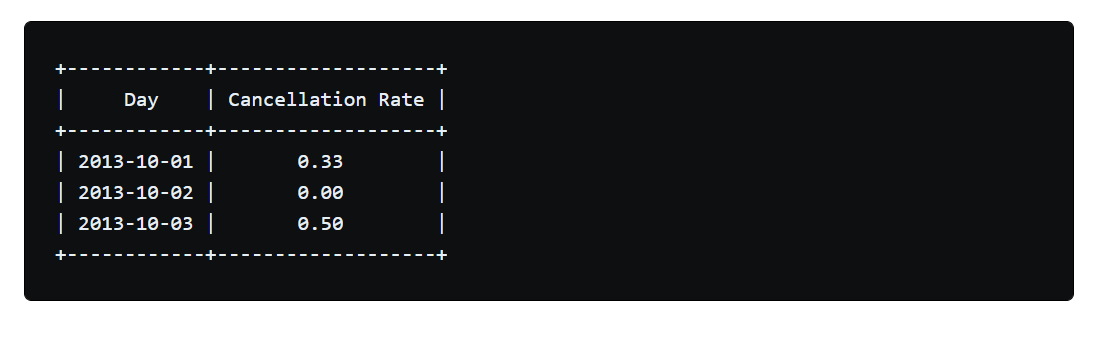
**select e.FIRST\_NAME,e.LAST\_NAME from employees e where e.MANAGER\_ID is null.**

**28.** The Trips table holds all taxi trips. Each trip has a unique Id, while Client\_Id and Driver\_Id are both foreign keys to the Users\_Id at the Users table. Status is an ENUM type of (‘completed’, ‘cancelled\_by\_driver’, ‘cancelled\_by\_client’).





Write a SQL query to find the cancellation rate of requests made by unbanned users between Oct 1, 2013 and Oct 3, 2013. For the above tables, your SQL query should return the following rows with the cancellation rate being rounded to two decimal places.



select

result.request\_at as "Day",

round(sum(case when result.status = 'completed' then 0 else 1 end)/count(\*),2 as "Cancellation Rate"

(select driver\_id,status,request\_at from trips left join users on trips.client\_id = users.users\_id where users.banned = 'NO')

result

left join users on result.users\_id = result.driver\_id ## two foreign keys are present so we need to join the both with users\_id

where

users.Banned = 'NO' and result.request\_at between '2013-10-01' and '2013-10-03'

group by result.request\_at

**29. Write a SQL query to find all duplicate emails in a table named Person.**



**For example, your query should return the following for the above table:**



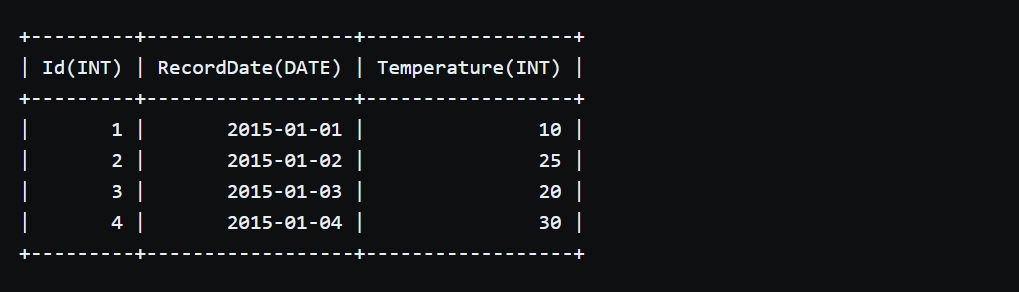
**Solution:**

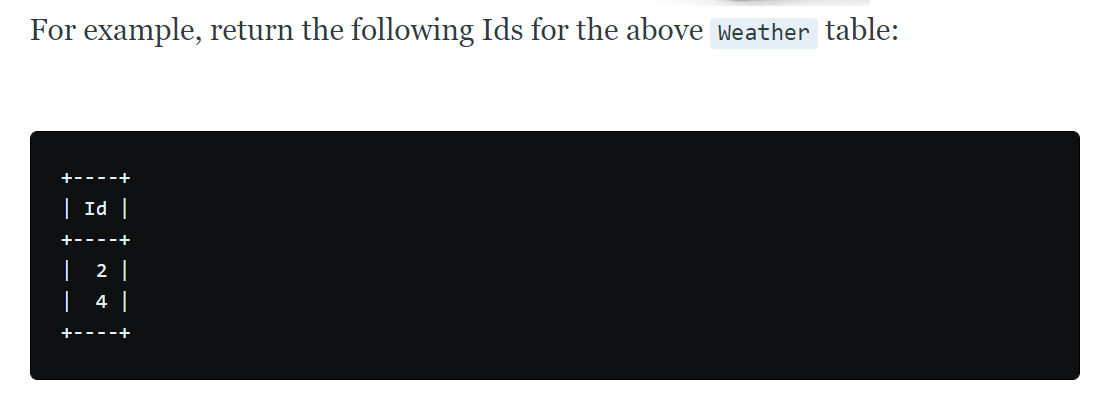
Since all email are in lowercase we can simply groupby email and print those that have a count >1.

If not in lowercase use the lower() string function.

Select lower(Email) as email from customers group by Email having count(Email)>1.

**30**. Given a Weather table, write a SQL query to find all dates' Ids with higher temperature compared to its previous (yesterday's) dates.

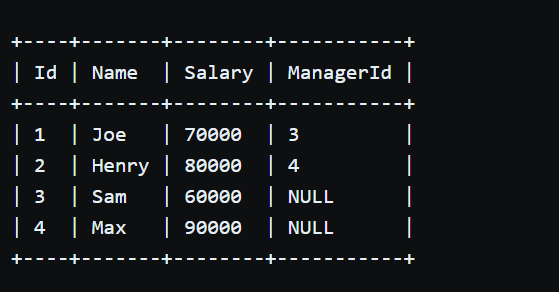




select w1.id from weather w1 inner join weather w2 on datediff(w1.recorddate,w w2.recorddate) =1

where w1.temperature > w2.temperature

**31.** The Employee table holds all employees including their managers. Every employee has an Id, and there is also a column for the manager Id.



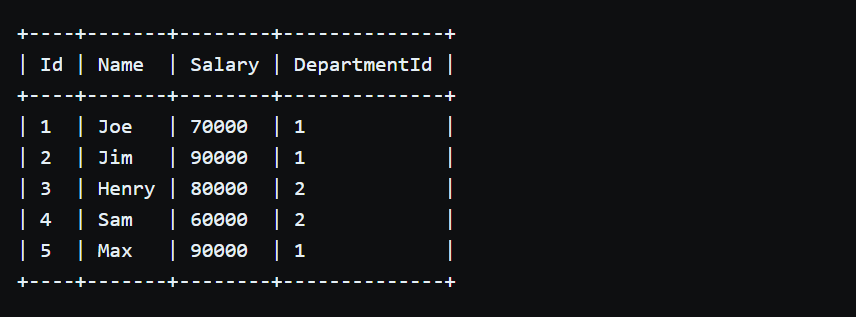
Given the Employee table, write a SQL query that finds out employees who earn more than their managers. For the above table, Joe is the only employee who earns more than his manager.

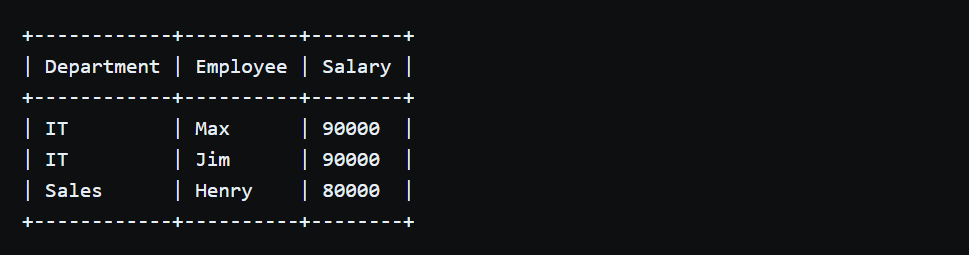


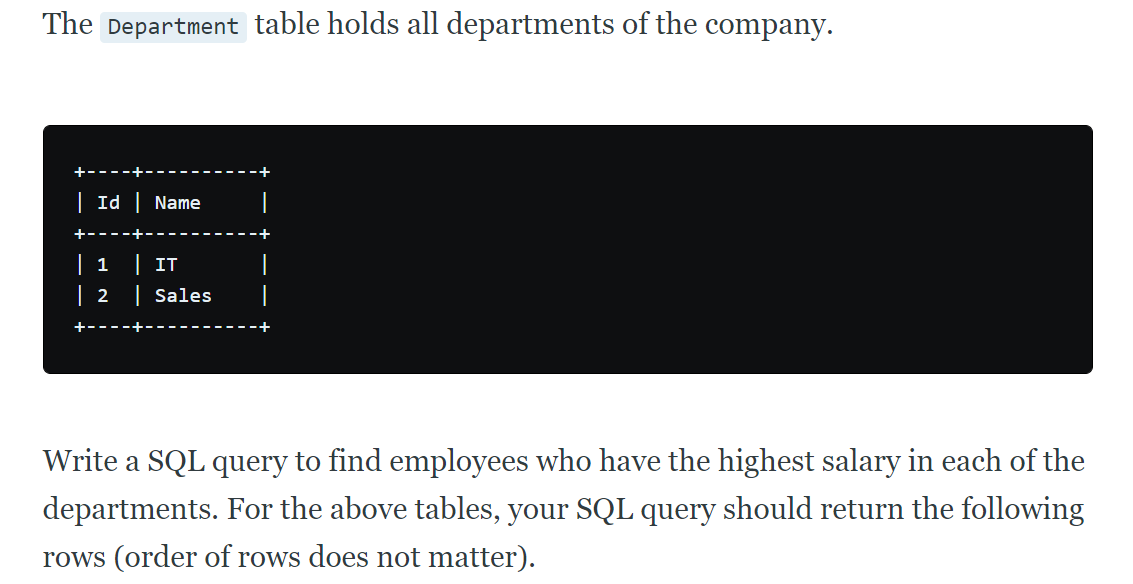
**select e1.FIRST\_NAME from employees e1 inner join employees e2**

**on e1.MANAGER\_ID = e2.EMPLOYEE\_ID where e1.salary > e2.salary**

**32.** The Employee table holds all employees. Every employee has an Id, a salary, and there is also a column for the department Id.







**Select Department.Id as ‘ID’, employee.name as ‘Name’**

**From**

**Employee inner join department on employee. DepartmentId = Department.Id**

**Where**

**(employee .salary, employee.DepartmentId)**

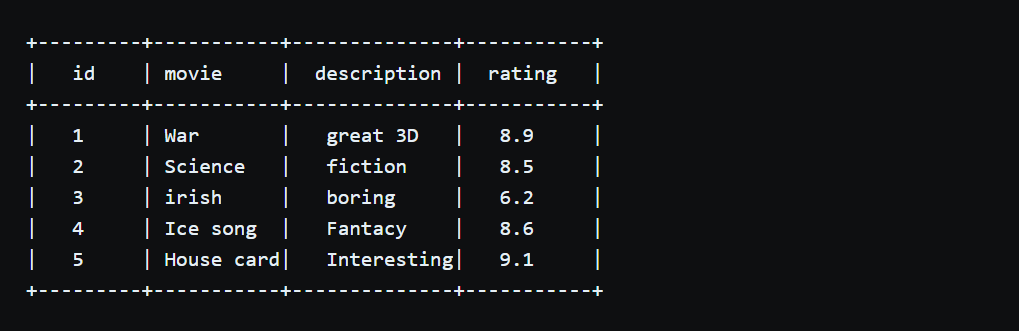
**in**

**(Select max(salary), DepartmentId from employee group by DepartmentId)**

**33.** X city opened a new cinema, many people would like to go to this cinema. The cinema also gives out a poster indicating the movies’ ratings and descriptions.

Please write a SQL query to output movies with an odd numbered ID and a description that is not 'boring'. Order the result by rating.

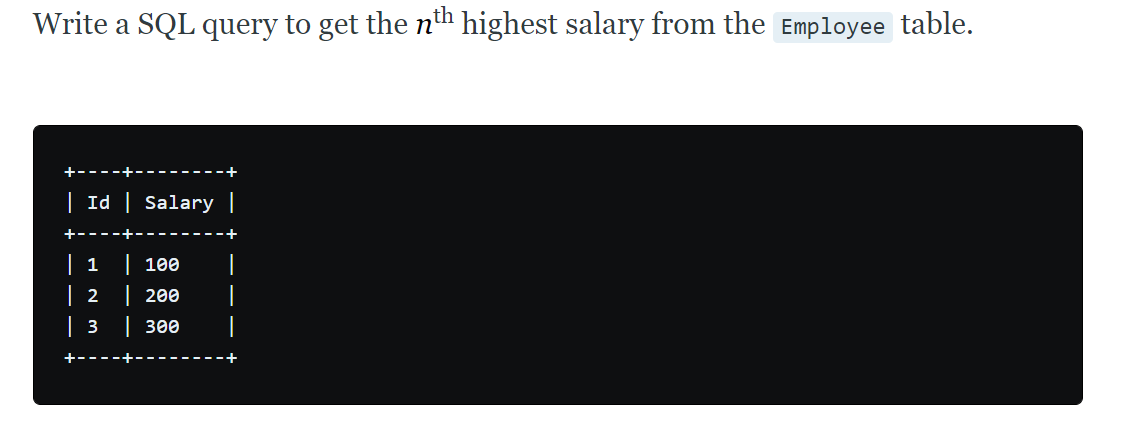
For example, table cinema:



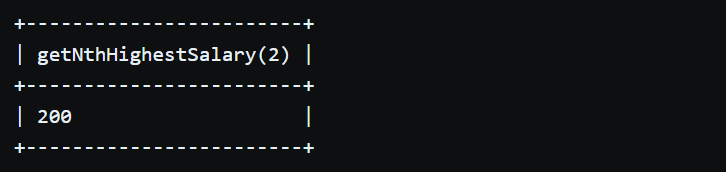
Select \* from cinema

Where id%2=1 and description != ‘boring’

Order by rating desc

**34.** 

For example, given the above Employee table, the nth highest salary where n = 2 is 200. If there is no nth highest salary, then the query should return null.



Create function getNthighestsalary (N int) returns int

BEGIN

return(

select e1.salary from employee e1 left join employee e2 on e1.salary <= e2.salary

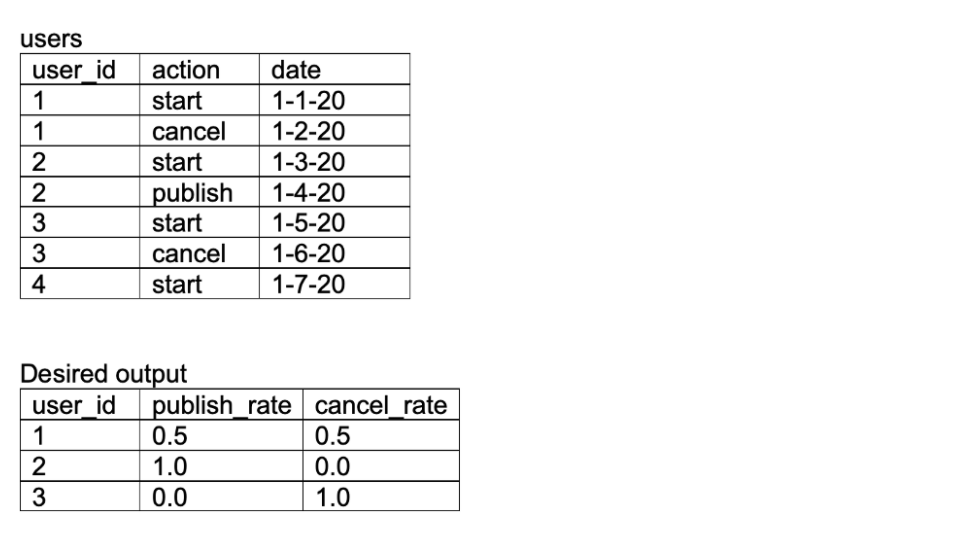
group by e1.salary

having count(distinct e2.salary) = N

);

END

**35.** From the following table of user IDs, actions, and dates, write a query to return the publication and cancellation rate for each user.



with users (user\_id,action,date)

as(values

(1,’start’,cast(‘1-1-20’ as date)),

(1,’cancel’,cast(‘1-2-20’ as date)),

(2,’start’,cast(‘1-3-20’ as date)),

(2,’publish’,cast(‘1-4-20’ as date)),

(3,’start’,cast(‘1-5-20’ as date)),

(3,’cancel’,cast(‘1-6-20’ as date)),

(4,’start’,cast(‘1-7-20’ as date))),

## now we will retrieve the sum of the start,cancel and publish

T1 as (

Select user\_id,

Sum(case when action = ‘start’ then 1 else 0 end) as starts,

Sum(case when action = ‘cancel’ then 1 else 0 end) as cancels,

Sum(case when action = ‘publish’ then 1 else 0 end) as publishes

From users

Group by 1

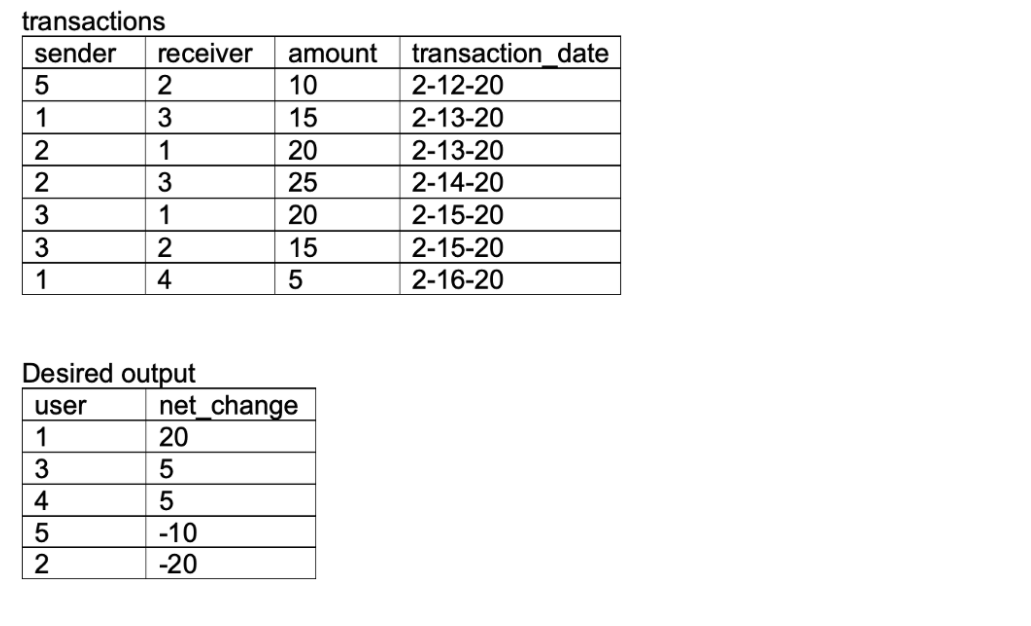
Order by 1)

## calculate the publication, cancelation rate of the each user by dividing the number by starts and casting as float by multiplying by 1

Select user\_id, 1.0\*publishes/starts as publish\_rate,

1.0\*cancels/starts as cancel\_rate from t1

**36.** From the following table of transactions between two users, write a query to return the change in net worth for each user, ordered by decreasing net change.



With transactions (sender,receiver,amount,transaction\_date)

As (values

(5,2,10,cast(‘2-12-20’ as date)),

(1,3,15,cast(‘2-13-20’ as date)),

(2,1,20,cast(‘2-14-20’ as date)),

(2,3,25,cast(‘2-15-20’ as date)),

(3,1,20,cast(‘2-16-20’ as date)),

(3,2,15,cast(‘2-17-20’ as date)),

(1,4,5,cast(‘2-18-20’ as date)),

## Total sum of debited debited amount from sender and credited amount for receiver

Debit as(

Select sender, sum(amount) as debited from transactions group by sender)

Credit as(

Select receiver, sum(amount) as credited from transactions group by receiver)

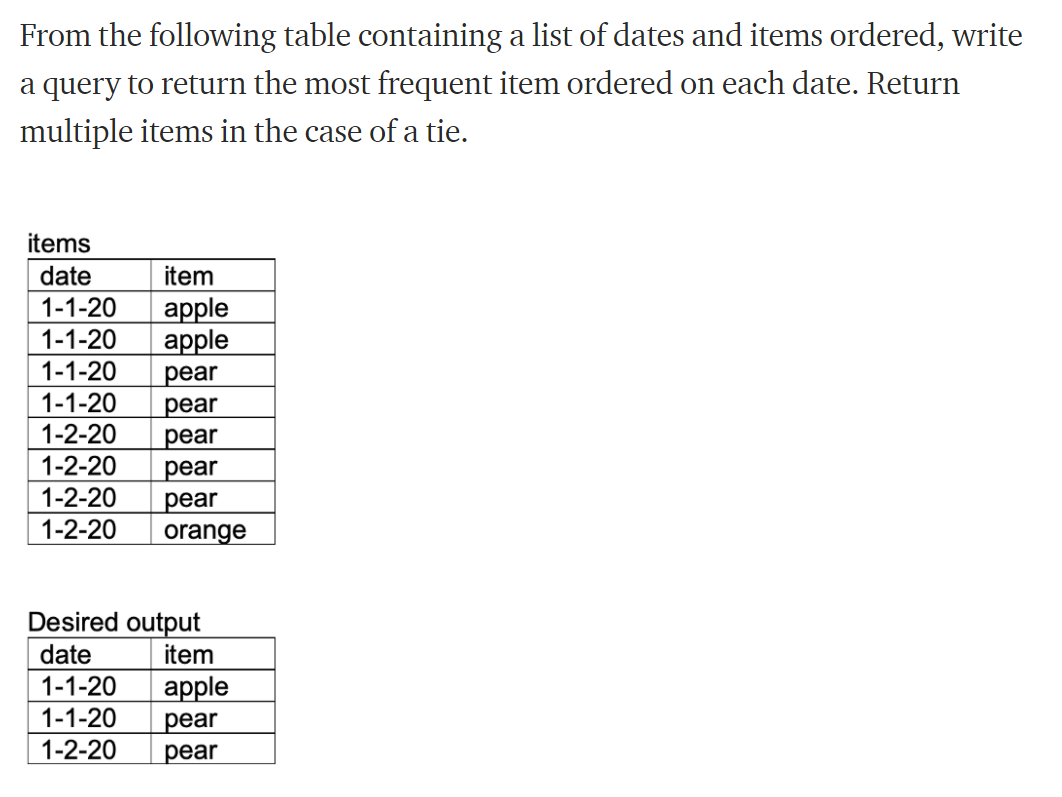
## we will need to full outer join on debit and credit on user id taking the net change as difference between credit and debit, coercing the nulls to 0 with

Coalesce() function

Select coalesce(sender,receiver) as user,

Coalesce(credited,0) – coalesce(debited,0) as net\_change from debits d full join credits c on d.sender = c.receiver order by 2 desc

**37.**



With items(date,item) as (values

(cast(‘01-01-20’ as date),’apple’),

(cast(‘01-01-20’ as date),’apple’),

(cast(‘01-01-20’ as date),’pear’),

(cast(‘01-01-20’ as date),’pear’),

(cast(‘01-02-20’ as date),’pear’),

(cast(‘01-02-20’ as date),’pear’),

(cast(‘01-02-20’ as date),’pear’),

(cast(‘01-02-20’ as date),’orange’))

## add an item count column to existing table, grouping by date and item columns

T1 as(

Select date,item,count(items) as item\_counts from items group by 1,2 order by 1)

## add a rank column in descending order, partitioning by date

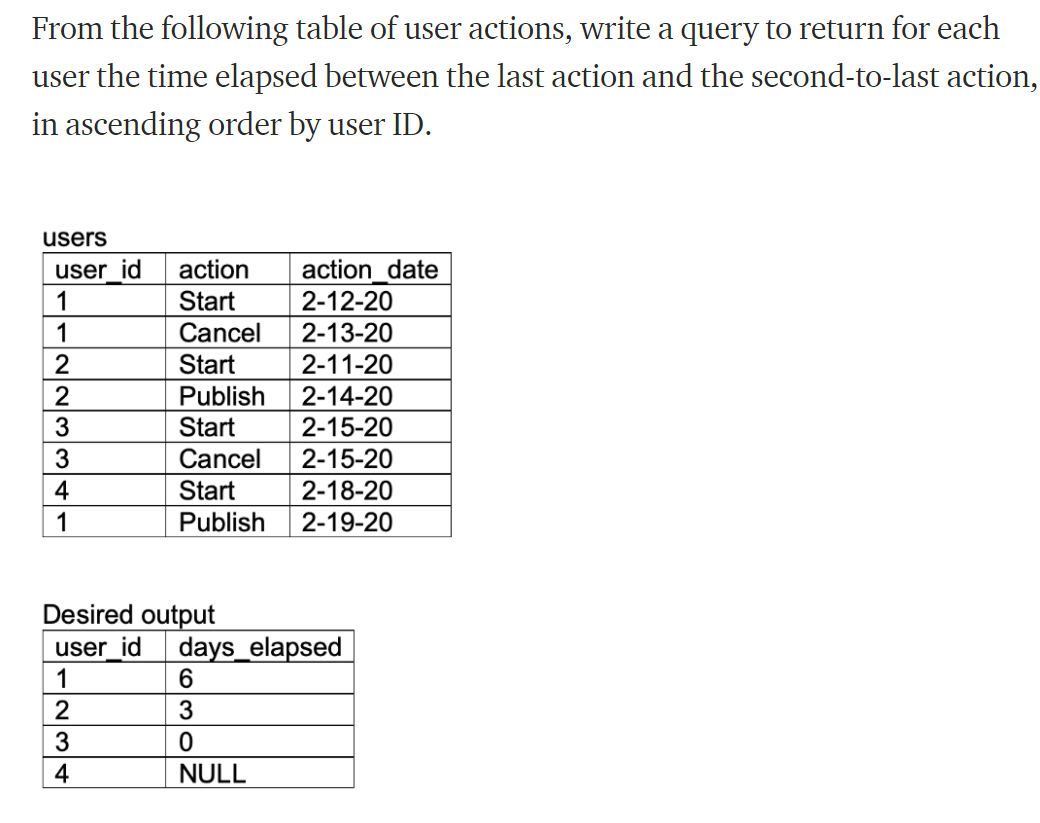
T2 as(

Select \*,rank() over (partition by date order by item\_counts desc) as date\_rank from T1

## Return all date,items where rank =1

Select date,item from T2 where date\_rank = 1

**38.**



With users (user\_id,action,action\_date) as (values

(1,’start’,caste(’02-12-20’ as date)),

(1,’cancel’,caste(’02-13-20’ as date)),

(2,’start’,caste(’02-11-20’ as date)),

(2,’publish’,caste(’02-14-20’ as date)),

(3,’start’,caste(’02-15-20’ as date)),

(3,’cancel’,caste(’02-15-20’ as date)),

(4,’start’,caste(’02-18-20’ as date)),

(1,’publish’,caste(’02-19-20’ as date))

## now we will create a date\_rank column and partition by user\_id using the row\_number() window function

T1 as (

Select \*, row\_number() over () (partition by user\_id order by action\_date desc) as date\_rank from users ),

## filter on date rank column to pull latest and next latest actions from this table.

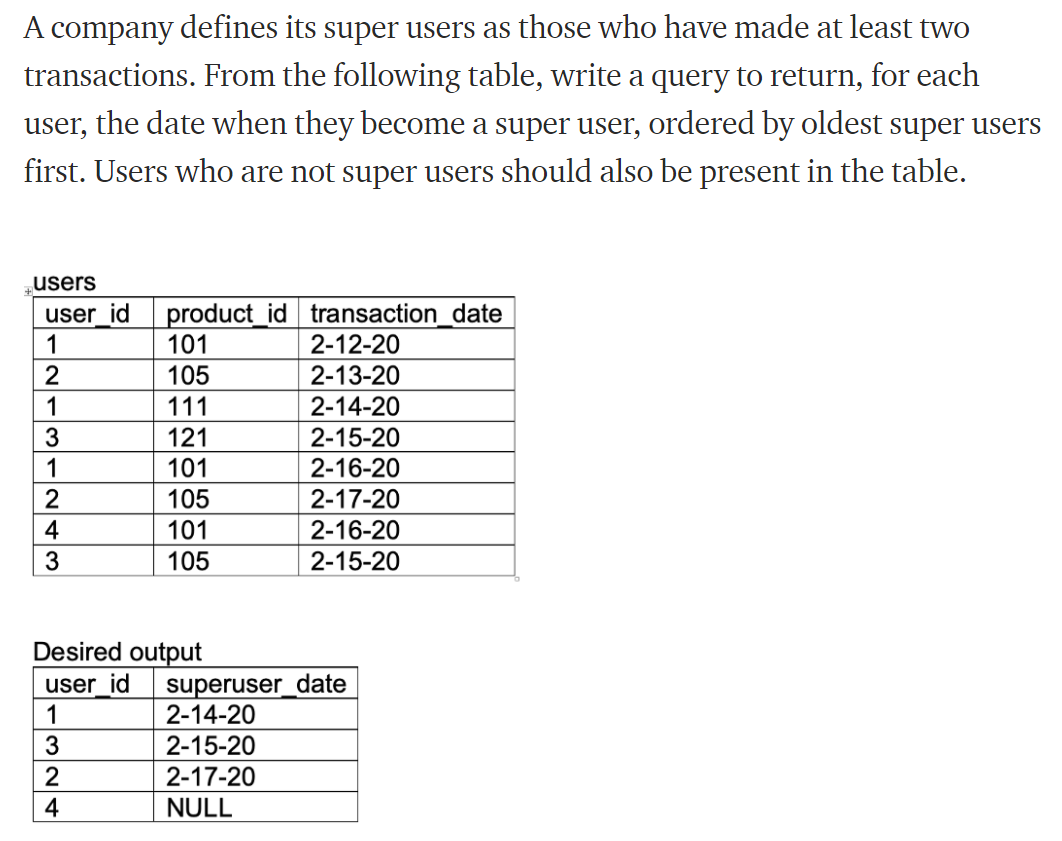
Latest as ( select \* from T1 where date\_rank = 1),

Second\_latest as ( select \* from T2 where date\_rank =2),

## left join these tables (everyone will have a latest action, not everyone will have a second latest action) so subtracting the latest with the second\_latest to get time elapsed

Select l1.user\_id, l1.action\_date – l2.action\_date as days\_elapsed from latest l1 left join second\_latest l2 on l1.user\_id = l2.user\_id order by 1

**39.**



With users( user\_id,product\_id,transaction\_date) as (values

(1,101, cast(‘2-12-20’ as date)),

(2,105, cast(‘2-13-20’ as date)),

(1,111, cast(‘2-14-20’ as date)),

(3,121, cast(‘2-15-20’ as date)),

(1,101, cast(‘2-16-20’ as date)),

(2,105, cast(‘2-17-20’ as date)),

(4,101, cast(‘2-16-20’ as date)),

(3,105, cast(‘2-15-20’ as date))

## create a transaction number column using row number () function, partitioning by user id

T1 as (

Select \*, row number() over() (partition by user\_id order by transaction\_date asc) as transaction\_number) from users),

## filter resulting table on transaction\_number =2

T2 as( select user\_id,transaction\_date from transaction\_number where transaction\_number =2),

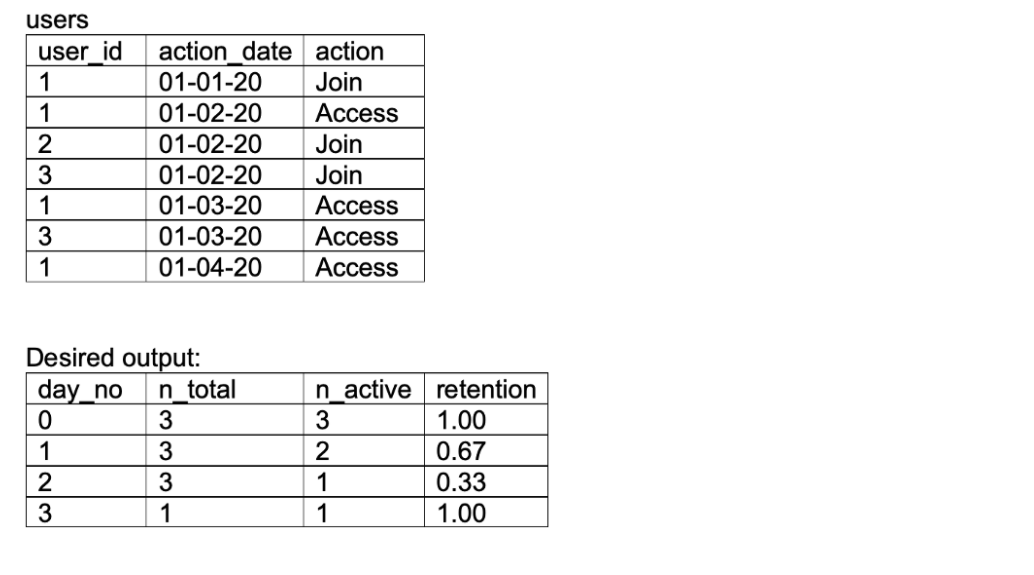
## select distinct user\_id from users

T3 as( select distinct(user\_id) from users)

## left join super\_users onto full user table order by date

Select t3.user\_id, t2.transaction\_date as superuser\_date from t3 left join t2 on t3.user\_id = t2.user\_id order by 2

**40.** From the following user activity table, write a query to return the fraction of users who are retained (show some activity) a given number of days after joining. By convention, users are considered active on their join day (day 0).



With users (user\_id,action\_date,action) as (values

(1,cast(’01-01-20’ as date),’Join’),

(1,cast(’01-02-20’ as date),’Access’),

(2,cast(’01-02-20’ as date),’Join’),

(3,cast(’01-02-20’ as date),’Join’),

(1,cast(’01-03-20’ as date), ’Access’),

(3,cast(’01-03-20’ as date), ’Access’),

(1,cast(’01-04-20’ as date), ’Access’)),

## we will get the join dates for each user

Join\_dates as (

Select user\_id, action\_date as join\_date from users where action = “Join”),

## create vector containing all dates in the date range

date\_vector as(

Select cast ( generate\_series (min (action\_date)), (max (action\_date)), ‘1 day’ :: interval) as date) as dates from users),

## cross join to get all possible user date combinations

all\_user\_dates as (

select distinct user\_id, d.dates from users cross join date\_vector d),

## left join users table on all\_users\_date combinations on matching the user\_id and date (null on days where users didn’t engage) , join onto this users sign up date, exclude user-date combination falling before user-signup

T1(

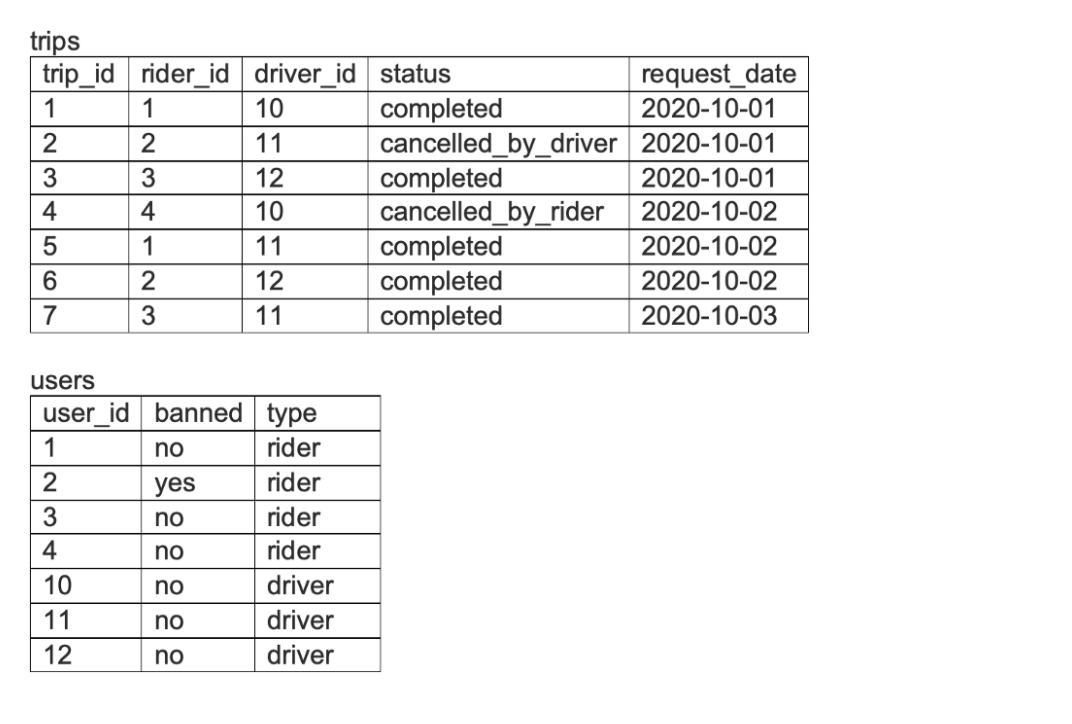
Select a.dates – c.join\_date as day\_no, b.user\_id

From all\_users\_date a left join users b on a.user\_id = b.user\_id and a.dates = b.action\_date join Join\_dates c on a.user\_id = c.user\_id where a.dates – c.join\_date>=0)

## grouping by days since sign\_up ,count (not-null) user\_ids as active users, total\_users, and the quotient as retention\_rate

Select day\_no, count(\*) as n\_total, count(distinct user\_id) as n\_active, round(1.0\*count(distinct user\_id)/count(\*),2) as retention from t1 group by 1

**42.** From the given trips and users tables for a taxi service, write a query to return the cancellation rate in the first two days in October, rounded to two decimal places, for trips not involving banned riders or drivers.



With trips (trip\_id,rider\_id,driver\_id,status,request\_date) as (values

(1,1,10,’completed’,cast(‘2020-10-01’ as date)),

(2,2,11,’cancelled\_by\_driver’,cast(‘2020-10-01’ as date)),

(3,3,12,’completed’,cast(‘2020-10-01’ as date)),

(4,4,10, ’cancelled\_by\_driver’,cast(‘2020-10-01’ as date)),

(5,1,11,’completed’,cast(‘2020-10-01’ as date)),

(6,2,12,’completed’,cast(‘2020-10-01’ as date)),

(7,3,11,’completed’,cast(‘2020-10-01’ as date))),

With users (user\_id,banned,type) as (values

(1,’no’,’rider’),

(1,’no’,’rider’),

(1,’no’,’rider’),

(1,’no’,’rider’),

(1,’no’,’rider’),

(1,’no’,’rider’),

(1,’no’,’rider’)),

--filter the trips table that will exclude the users and drivers as banned then calculate the cancelation rate from the 1- fraction(completed rides), also rounding as said and for first two days in october.

Select request\_date, round( 1 – 1.0\*sum( case when status = “completed” then 1 else 0 end/count(\*),2) as cancel\_rates from trips

Where driver\_id not in (select user\_id from users where banned = “yes” )

Where rider\_id not in (select user\_id from users where banned = “yes”)

Group\_by request\_date

Having extract (day from request\_date) <= 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Theory\_questions\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**43.** What is the use of the Intersect operator?

The Intersect operator helps combine two select statements and returns only those records that are common to both the select statements. So, after we get Table A and Table B over here and if we apply the Intersect operator on these two tables, then we will get only those records that are common to the result of the select statements of these two.

**44.** What is Cursor in SQL ?

**Cursor** is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on Table by User. Cursors are used to store Database Tables. There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors. These are explained as following below.

1. **Implicit Cursors:**  
   Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
2. **Explicit Cursors :**  
   Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

**How to create Explicit Cursor:**

1. **Declare Cursor Object.**  
   **Syntax :** DECLARE cursor\_name CURSOR FOR SELECT \* FROM table\_name

DECLARE s1 CURSOR FOR SELECT \* FROM studDetails

**45.** What is the difference between Union and Union All operators?

The UNION operator is used to combine the result-set of two or more SELECT statements.

Unlike Union Operator, Union All won't eliminate the duplicates

**46.** Explain the difference between OLTP and OLAP.

**Online Analytical Processing (OLAP):** Online Analytical Processing consists of a type of software tools that are used for data analysis for business decisions. [OLAP](https://www.geeksforgeeks.org/olap-servers/) provides an environment to get insights from the database retrieved from multiple database systems at one time.

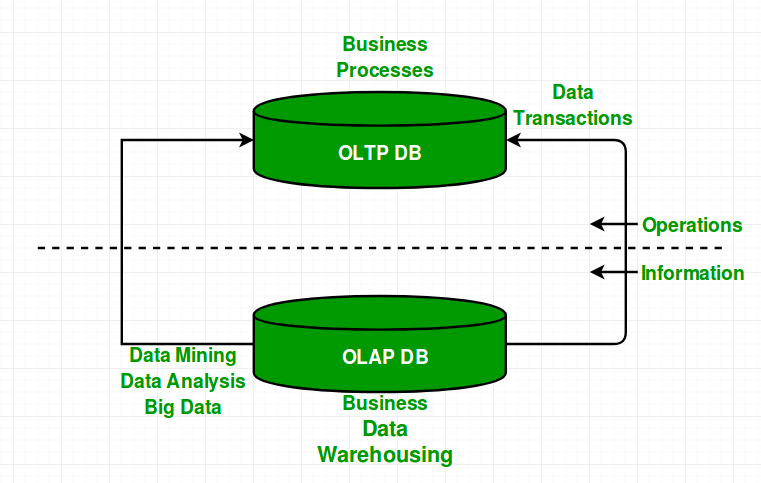
**Examples –** Any type of Data warehouse system is an OLAP system. The [uses of OLAP](https://www.geeksforgeeks.org/olap-applications/) are as follows:

* Spotify analyzed songs by users to come up with a personalized homepage of their songs and playlist.
* Netflix movie recommendation system.

**Online transaction processing (OLTP):** [Online transaction processing](https://www.geeksforgeeks.org/on-line-transaction-processing-oltp-system-in-dbms/) provides transaction-oriented applications in a [3-tier architecture](https://www.geeksforgeeks.org/introduction-of-3-tier-architecture-in-dbms-set-2/). OLTP administers the day-to-day transactions of an organization.

**Examples:**Uses of OLTP are as follows:

* ATM center is an OLTP application.
* OLTP handles the ACID properties during data transactions via the application.
* It’s also used for Online banking, Online airline ticket booking, sending a text message, add a book to the shopping cart.



**47.** What is a stored procedure? Give an example.

A stored procedure is a prepared SQL code that can be saved and reused. In other words, we can consider a stored procedure to be a function consisting of many SQL statements to access the database system. We can consolidate several SQL statements into a stored procedure and execute them whenever and wherever required.

A stored procedure can be used as a means of modular programming, i.e., we can create a stored procedure once, store it, and call it multiple times as required. This also supports faster execution when compared to executing multiple queries.

Syntax:-CREATE PROCEDURE *procedure\_name*  
AS  
*sql\_statement*  
GO;

Create procedure stored

As

Select \* from employees

Go;