

SQL ASSIGNMENT SET 3

Q101) Write an SQL query to show the second most recent activity of each user. If the user only has one activity, return that one. A user cannot perform more than one activity at the same time.

Return the result table in any order. The query result format is in the following example.

Sol) At first create an empty table named ‘UserActivity’ and then insert records in it using multi insert statement.

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, a database named 'assignment3' is selected, and a table named 'useractivity' is created with columns 'username' and 'activity'. A multi-insert statement is run, inserting four rows of data. The results grid shows the inserted data:

username	activity	start_date	end_date
Alice	Travel	2020-02-12	2020-02-20
Alice	Dancing	2020-02-21	2020-02-23
Alice	Travel	2020-02-24	2020-02-28
Bob	Travel	2020-02-11	2020-02-18

The status bar indicates 4 row(s) returned.

Select username,activity,start_date,end_date from
(select *,
rank() over(partition by username order by start_date desc)
as 'rnk',
count(activity) over(partition by username) as 'count'
from useractivity) as u
where (count > 1 and rnk = 2) or (count = 1 and rnk = 1);

The screenshot shows the execution of the final query. The results grid displays the expected output:

username	activity	start_date	end_date
Alice	Dancing	2020-02-21	2020-02-23
Bob	Travel	2020-02-11	2020-02-18

The status bar indicates 2 row(s) returned.

Q102) SAME QUESTION AS Q101

Q103) Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending order.

Sol) At first create an empty table named ‘students’ and then insert records in it using multi insert statement.

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, under 'Tables', there is a list of tables including 'actions', 'activity', 'actordirector', 'advertiser', 'calls', 'cinema', and 'company'. The 'students' table is selected. In the main editor area, the following SQL code is written:

```
16 *  create table students(id int,name varchar(15),marks int);
17 *  insert into students values(1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),
18   (3,'Belvet',84);
19 *  select * from students;
```

The results grid shows the following data:

	id	name	marks
1	Ashley	81	
2	Samantha	75	
4	Julia	76	
3	Belvet	84	

Below the results grid, the 'Action Output' section shows the history of actions:

#	Time	Action	Message	Duration / Fetch
42	19:45:52	insert into students values(1,'Ashley',81),(2,'Samantha',75),(4,'Julia',76),(3,'Belvet',84)	4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0	0.015 sec
43	19:46:07	select * from students	4 row(s) returned	0.015 sec / 0.000 sec

**select name from students
where marks>75
order by SUBSTR(name, -3), id asc;**

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, under 'Tables', there is a list of tables including 'actions', 'activity', 'actordirector', 'advertiser', 'calls', 'cinema', and 'company'. The 'students' table is selected. In the main editor area, the following SQL code is written:

```
20 *  select name from students
21 where marks>75
22 order by SUBSTR(name, -3), id asc;
```

The results grid shows the following data:

	name
1	Ashley
4	Julia
3	Belvet

Q104) SAME QUESTION AS Q82

Q105) SAME QUESTION AS Q83

Q106) SAME QUESTION AS Q84

Q107) Samantha was tasked with calculating the average monthly salaries for all employees in the EMPLOYEES table, but did not realize her keyboard's 0 key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeros removed), and the actual average salary.

Write a query calculating the amount of error (i.e.: actual - miscalculated average monthly salaries), and round it up to the next integer.

Sol) At first create an empty table named 'employees' and then insert records in it using multi insert statement.

```
select ceil(avg(salary)-avg(replace(salary,'0','')))  
From employees
```

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' tree view shows various database objects like Tables, Views, Stored Procedures, Functions, and several user-defined functions (ineuron, ineuron_1, ineuron_partition, key_prm, operation, retail). The central pane displays the following SQL code:

```
23  
24 • create table employees(id int,name varchar(15),salary int);  
25 • insert into employees values(1,'Kristten',1420),(2,'Ashley',2006),  
26 (3,'Julia',2210),(4,'Maria',3000);  
27  
28 • select ceil(avg(salary)-avg(replace(salary,'0','')))  
29 from employees;  
30
```

Below the code, the 'Result Grid' shows the result of the query: '2061'. The 'Output' tab shows the execution details:

- Action Output: # 17 10:31:01 SELECT CEIL(AVG(Salary) - AVG(REPLACE(Salary, '0', ''))) FROM EMPLOYEES
- Action Output: # 18 10:32:00 select ceil(avg(salary)-avg(replace(salary,'0',''))) from employees

Messages indicate '1 row(s) returned' for both actions. The total duration is 0.015 sec / 0.000 sec.

Q108) Write a query to find the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as 2 space-separated integers.

Sol) At first create an empty table named 'employee' and then insert records in it using multi insert statement.

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' tree view shows various database objects like Tables, Views, Stored Procedures, Functions, and several user-defined functions (ineuron, ineuron_1, ineuron_partition, key_prm, operation, retail). The central pane displays the following SQL code:

```
31 • create table employee(emp_id int,name varchar(15),months int,  
32 salary int);  
33 • insert into employee values(12228,'Rose',15,1968),(33645,'Angela',1,3443),  
34 (45692,'Frank',17,1608),(56118,'Patrick',7,1345),(59725,'Lisa',11,2330),  
35 (74197,'Kimberly',16,4372),(78454,'Bonnie',8,1771),(83565,'Michael',6,2017),  
36 (98607,'Todd',5,3396),(99989,'Joe',9,3573);  
37 • select * from employee;
```

Below the code, the 'Result Grid' shows the result of the query, listing 6 rows of employee data. The 'Output' tab shows the execution details:

- Action Output: # 12228 Rose 15 1968
- Action Output: # 33645 Angela 1 3443
- Action Output: # 45692 Frank 17 1608
- Action Output: # 56118 Patrick 7 1345
- Action Output: # 59725 Lisa 11 2330
- Action Output: # employee 6

Messages indicate '1 row(s) returned' for each action. The total duration is 0.000 sec / 0.000 sec.

The screenshot shows a MySQL Workbench interface. In the top pane, a query is being run:

```

37 • select * from employee;
38 • select months*salary as 'earnings' , count(*) from employee
39 group by earnings
40 order by earnings desc
41 limit 1;

```

The results grid shows one row:

earnings	count(*)
69952	1

The action output log shows two entries:

- # 20 10:36:03 select months*salary , count(*) from employee group by months*salary order by mont... 1 row(s) returned Duration / Fetch 0.016 sec / 0.000 sec
- # 21 10:36:44 select months*salary as 'earnings' , count(*) from employee group by earnings order ... 1 row(s) returned Duration / Fetch 0.000 sec / 0.000 sec

**select months*salary as 'earnings' , count(*) from employee
group by earnings
order by earnings desc
limit 1;**

Q109) Query the number of occurrences of each occupation in OCCUPATIONS. Sort the occurrences in ascending order, and output them in the following format:

Sol) At first create an empty table named ‘occupations’ and then insert records in it using multi insert statement.

The screenshot shows a MySQL Workbench interface. In the top pane, a query is being run:

```

43 • create table occupations(name varchar(15),occupation varchar(15));
44 • insert into occupations value('Samantha','Doctor'),('Julia','Actor'),
45 ('Maria','Actor'),('Meera','Singer'),('Ashley','Professor'),('Ketty','Proffesor'),
46 ('Christeen','Proffesor'),('Jane','Actor'),('Jenny','Doctor'),('Priya','Singer');
47 • select * from occupations;

```

The results grid shows the inserted data:

name	occupation
Samantha	Doctor
Julia	Actor
Maria	Actor
Meera	Singer
Ashley	Professor
Ketty	Proffesor
Christeen	Proffesor
Jane	Actor
Jenny	Doctor
Priya	Singer

The action output log shows two entries:

- # 23 10:41:51 insert into occupations value('Samantha','Doctor'),('Julia','Actor'),(M... 10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0 Duration / Fetch 0.015 sec
- # 24 10:42:21 select * from occupations 10 row(s) returned Duration / Fetch 0.000 sec / 0.000 sec

**select concat(name,','left(occupation, 1),')') as
sample_output
from occupations
order by name;**

Q110) Pivot the Occupation column in OCCUPATIONS so that each Name is sorted alphabetically and displayed underneath its corresponding Occupation. The output column headers should be Doctor, Professor, Singer, and Actor, respectively.

Note: Print NULL when there are no more names corresponding to an occupation.

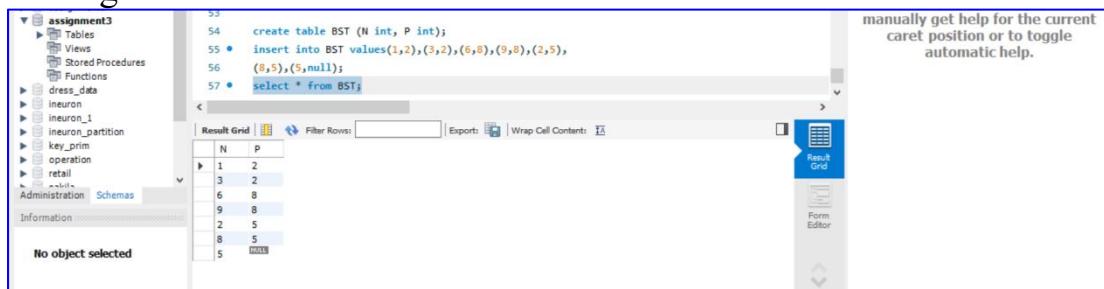
Sol) The table ‘Occupations’ used for this query is already created in previous question.

```
select Doctor,Professor,Singer,Actor
from (select NameOrder,
           max(case Occupation when 'Doctor' then Name end) as Doctor,
           max(case Occupation when 'Professor' then Name end) as Professor,
           max(case Occupation when 'Singer' then Name end) as Singer,
           max(case Occupation when 'Actor' then Name end) as Actor
      from (select Occupation,Name,
                 row_number() over(partition by Occupation order by Name
ASC) as NameOrder
            from Occupations
          ) as NameLists group by NameOrder
      ) as Names;
```

Q111) Write a query to find the node type of Binary Tree ordered by the value of the node. Output one of the following for each node:

- **Root:** If node is root node.
- **Leaf:** If node is leaf node.
- **Inner:** If node is neither root nor leaf node

Sol) At first create an empty table named ‘BST’ and then insert records in it using multi insert statement.



```

53
54 • create table BST (N int, P int);
55 • insert into BST values(1,2),(3,2),(6,8),(9,8),(2,5),
56 (8,5),(5,NULL);
57 • select * from BST;
  
```

The screenshot shows the Object Explorer on the left with the schema 'assignment3' selected. In the center, a query window displays the creation of a table 'BST' and its population with data. The table has two columns: 'N' and 'P'. The data inserted is:

N	P
1	2
3	2
6	8
9	8
2	5
8	5
5	NULL

select N,

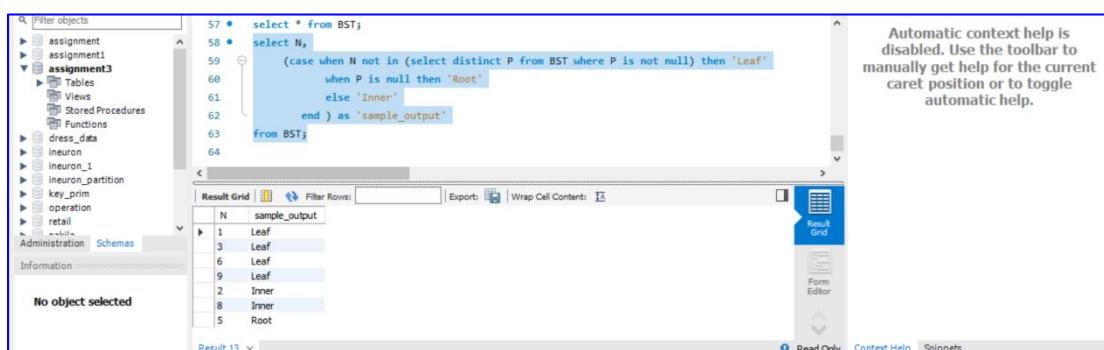
(case when N not in (select distinct P from BST where P is not null) then 'Leaf'

when P is null then 'Root'

else 'Inner'

end) as 'sample_output'

from BST;



```

57 • select * from BST;
58 • select N,
59   (case when N not in (select distinct P from BST where P is not null) then 'Leaf'
60         when P is null then 'Root'
61         else 'Inner'
62       end ) as 'sample_output'
63
64 From BST;
  
```

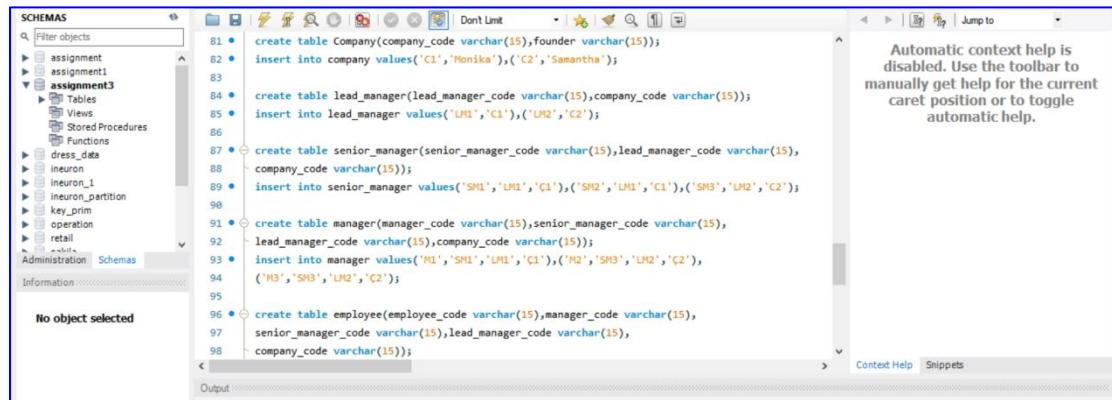
The screenshot shows the execution of the query. The results are displayed in a grid:

N	sample_output
1	Leaf
3	Leaf
6	Leaf
9	Leaf
2	Inner
8	Inner
5	Root

Q112) Given the table schemas below, write a query to print the company_code, founder name, total number of lead managers, total

number of senior managers, total number of managers, and total number of employees. Order your output by ascending company_code.

Sol) In this question, we have to first create 5 empty tables naming 'Company', 'Lead_manager', 'Senior_manager', 'Manager', 'Emp' and then insert records in it using multi insert statement.



```

SCHEMAS
Filter objects
assignment
assignment1
assignment3
Tables
Views
Stored Procedures
Functions
dress_data
ineuron
ineuron_1
ineuron_partition
key_prim
operation
retail
Administration Schemas
Information
No object selected
Output

81 • create table Company(company_code varchar(15),founder varchar(15));
82 • insert into company values('C1','Monika'),('C2','Samantha');
83
84 • create table lead_manager(lead_manager_code varchar(15),company_code varchar(15));
85 • insert into lead_manager values('LM1','C1'),('LM2','C2');
86
87 • create table senior_manager(senior_manager_code varchar(15),lead_manager_code varchar(15),
     company_code varchar(15));
88 • insert into senior_manager values('SM1','LM1','C1'),('SM2','LM1','C1'),('SM3','LM2','C2');
89
90 • create table manager(manager_code varchar(15),senior_manager_code varchar(15),
     lead_manager_code varchar(15),company_code varchar(15));
91 • insert into manager values('M1','SM1','LM1','C1'),('M2','SM3','LM2','C2'),
     ('M3','SM3','LM2','C2');
92
93 • create table employee(employee_code varchar(15),manager_code varchar(15),
     senior_manager_code varchar(15),lead_manager_code varchar(15),
     company_code varchar(15));
94
95
96 •
97
98

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Context Help Snippets

```

```

select c.company_code, c.founder,
       count(distinct l.lead_manager_code),
       count(distinct s.senior_manager_code),
       count(distinct m.manager_code),
       count(distinct e.employee_code)
from Company as c join Lead_Manager as l
on c.company_code = l.company_code
join Senior_Manager as s on l.lead_manager_code
= s.lead_manager_code
join Manager as m on m.senior_manager_code =
s.senior_manager_code
join Employee as e on e.manager_code = m.manager_code
group by c.company_code, c.founder
order by c.company_code;

```

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' tab, there is a tree view of database objects. The 'Information' schema is selected. In the main area, a SQL query is being run:

```

111 join Senior_Manager as s
112   on l.lead_manager_code = s.lead_manager_code
113 join Manager as m
114   on m.senior_manager_code = s.senior_manager_code
115 join Employee as e
116   on e.manager_code = m.manager_code
117 group by c.company_code, c.founder
118 order by c.company_code;

```

The result grid shows the following data:

company_code	founding	count(distinct lead_manager_code)	count(distinct senior_manager_code)	count(distinct manager_code)
C1	Monika	1	1	1
C2	Samantha	1	1	2

Q113) Write a query to print all prime numbers less than or equal to 1000. Print your result on a single line, and use the ampersand () character as your separator (instead of a space). For example, the output for all prime numbers ≤ 10 would be: 2&3&5&7

Q114) P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

```

*
* *
* * *
* * * *
* * * * *

```

Write a query to print the pattern P(20)

Sol) set @number = 0;
 select repeat('*', @number := @number + 1)
 from information_schema.tables
 where @number < 20

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' tab, there is a tree view of database objects. The 'Information' schema is selected. In the main area, a SQL query is being run:

```

65 •  set @number = 0;
66 •  select repeat('*', @number := @number + 1)
67  from information_schema.tables
68  where @number < 20

```

The result grid shows the following pattern:

```

*
**
***
*****
*****

```

Q115) P(R) represents a pattern drawn by Julia in R rows. The following pattern represents P(5):

```

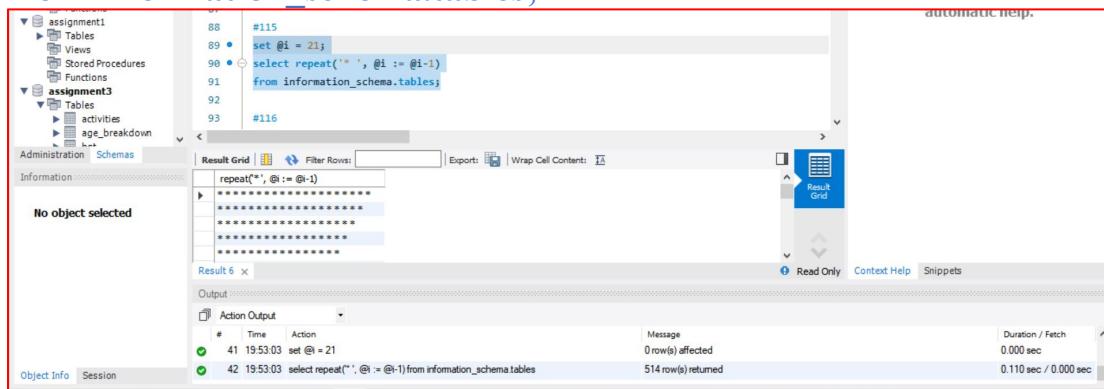
* * * * *

```

* * * *
 * * *
 * *
 *

Write a query to print the pattern P(20)

Sol) set @i = 21;
 select repeat('* ', @i := @i-1)
 from information_schema.tables;



```

88 • #115
89 •     set @i = 21;
90 •     select repeat('* ', @i := @i-1)
91     from information_schema.tables;
92
93 • #116

```

The screenshot shows the MySQL Workbench interface. The left sidebar shows the database schema with tables like assignment, assignment3, and information. The main area shows the SQL editor with the above query. The results pane displays a diamond-shaped pattern of asterisks:

```

*****
 ****
  ***
   *

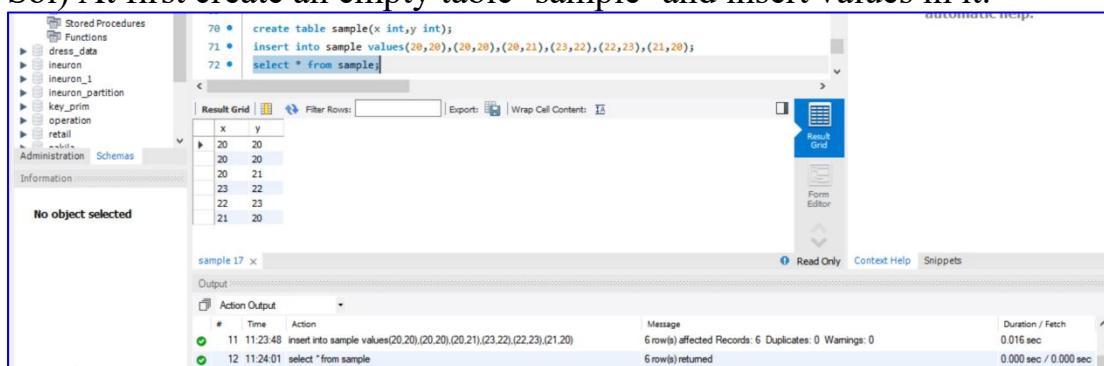
```

The output pane shows the execution details:

#	Time	Action	Message	Duration / Fetch
41	19:53:03	set @i = 21	0 row(s) affected	0.000 sec
42	19:53:03	select repeat('* ', @i := @i-1) from information_schema.tables	514 row(s) returned	0.110 sec / 0.000 sec

Q116) Two pairs (X1, Y1) and (X2, Y2) are said to be symmetric pairs if X1 = Y2 and X2 = Y1.
 Write a query to output all such symmetric pairs in ascending order by the value of X. List the rows such that X1 ≤ Y1.

Sol) At first create an empty table ‘sample’ and insert values in it.



```

70 • create table sample(x int,y int);
71 • insert into sample values(20,20),(20,20),(20,21),(23,22),(22,23),(21,20);
72 • select * from sample;

```

The screenshot shows the MySQL Workbench interface. The left sidebar shows the database schema with tables like dress_data, inuron, inuron_1, inuron_partition, key_prime, operation, ref, and sample. The main area shows the SQL editor with the above query. The results pane displays the following data:

x	y
20	20
20	20
20	21
23	22
22	23
21	20

The output pane shows the execution details:

#	Time	Action	Message	Duration / Fetch
11	11:23:48	insert into sample values(20,20),(20,20),(20,21),(23,22),(22,23),(21,20)	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0	0.016 sec
12	11:24:01	select * from sample	6 row(s) returned	0.000 sec / 0.000 sec

select f1.x,f1.y
 from sample f1, sample f2
 where f1.x=f2.y and f1.y=f2.x
 group by f1.x,f1.y
 having count(f1.x)>1 or f1.x<f1.y
 order by f1.x;

The screenshot shows a database interface with a sidebar containing objects like assignment, assignment1, and assignment3. The main area displays a SQL query:

```
73  
74 • select f1.x,f1.y  
75   from sample f1, sample f2  
76   where f1.x=f2.y and f1.y=f2.x  
77   group by f1.x,f1.y  
78   having count(f1.x)>1 or f1.x<f1.y  
79   order by f1.x;
```

The results grid shows the following data:

x	y
20	20
20	21
22	23

A tooltip on the right says: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

Q117) SAME QUESTION AS Q103

Q118) SAME QUESTION AS Q104

Q119) SAME QUESTION AS Q105

Q120) SAME QUESTION AS Q106

Q121) SAME QUESTION AS Q85

Q122) SAME QUESTION AS Q86

Q123) SAME QUESTION AS Q87

Q124) SAME QUESTION AS Q88

Q125) SAME QUESTION AS Q89

Q126) SAME QUESTION AS Q90

Q127) SAME QUESTION AS Q91

Q128) SAME QUESTION AS Q92

Q129) SAME QUESTION AS Q93

Q130) SAME QUESTION AS Q94

Q131) SAME QUESTION AS Q95

Q132) SAME QUESTION AS Q96

Q133) SAME QUESTION AS Q97

Q134) SAME QUESTION AS Q98

Q135) SAME QUESTION AS Q99

Q136) SAME QUESTION AS Q100

Q137) SAME QUESTION AS Q101

Q138) SAME QUESTION AS Q137

Q139) SAME QUESTION AS Q103

Q140) SAME QUESTION AS Q104

Q141) SAME QUESTION AS Q105

Q142) SAME QUESTION AS Q106

Q143) SAME QUESTION AS Q107

Q144) SAME QUESTION AS Q108

Q145) SAME QUESTION AS Q109

Q146) SAME QUESTION AS Q110

Q147) SAME QUESTION AS Q111

Q148) SAME QUESTION AS Q112

Q149) SAME QUESTION AS Q113

Q150) Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students get the same salary offer.

Sol) At first create an empty 3 table ‘Students’, ‘friends’ & ‘packages’ and insert values in it.

The screenshot shows the Oracle SQL Developer interface. On the left, there's a tree view of schemas: key_prim, operation, retail, and a newly created schema named 'test'. The 'Information' tab is selected, showing 'No object selected'. In the main pane, SQL code is being entered:

```
119
120 • create table students(id int, name varchar(15));
121 • create table friends(id int,friend_id int);
122 • create table packages(id int,salary float);
123 • insert into students values(1,'Ashley'),(2,'Samantha'),(3,'Julia')
124 ,,(4,'Scarlet');
125 • insert into friends values(1,2),(2,3),(3,4),(4,1);
126 • insert into packages values(1,15.20),(2,10.06),(3,11.55),(4,12.12);
```

Below the code, the 'Output' tab is open, showing the execution results:

#	Time	Action	Message	Duration / Fetch
22	16:59:44	insert into friends values(1,2),(2,3),(3,4),(4,1)	4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0	0.016 sec
23	17:00:32	insert into packages values(1,15.20),(2,10.06),(3,11.55),(4,12.12)	4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0	0.016 sec

```
select S1.Name from Students s1
inner join Packages p1 on s1.Id = p1.Id
inner join Friends f on s1.id = f.Id
inner join Students s2 on f.Friend_Id = s2.Id
inner join Packages p2 on s2.id = p2.Id
where p1.Salary < p2.Salary
order by p2.Salary ;
```

The screenshot shows a database interface with a left sidebar displaying schema objects like assignment, assignment1, and assignment3. The main area contains a code editor with the following SQL query:

```

128 •    select S1.Name from Students s1
129     inner join Packages p1 on s1.Id = p1.Id
130     inner join Friends f on s1.Id = f.Id
131     inner join Students s2 on f.Friend_Id = s2.Id
132     inner join Packages p2 on s2.Id = p2.Id
133     where p1.Salary < p2.Salary
134     order by p2.Salary ;

```

The results grid below the code shows three rows:

Name
Samantha
Julia
Scarlet

A message in the top right corner says: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

Q151) Julia just finished conducting a coding contest, and she needs your help assembling the leaderboard! Write a query to print the respective hacker_id and name of hackers who achieved full scores for more than one challenge. Order your output in descending order by the total number of challenges in which the hacker earned a full score. If more than one hacker received full scores in the same number of challenges, then sort them by ascending hacker_id.

Sol) At first create an empty 4 table 'Hackers', 'Difficulty' , 'Challenges' & 'Submissions' and insert values in it.

The screenshot shows a database interface with a left sidebar displaying schema objects like assignment, assignment1, and assignment3. The main area contains a code editor with the following SQL queries:

```

136 •    create table hackers(hacker_id int,name varchar(15));
137 •    create table difficulty(difficulty_level int,score int);
138 •    create table challenges(challenge_id int,hacker_id int,difficulty_level int);
139 •    create table submissions(submission_id int,hacker_id int,challenge_id int,score int);
140
141 •    insert into hackers values(5580,'Rose'),(8439,'Angela'),(27205,'Frank'),
142     (52249,'Patrick'),(52348,'Lisa'),(57645,'Kimberly'),(77726,'Bonnie'),
143     (83082,'Michael'),(86870,'Todd'),(90411,'Joe');
144
145 •    insert into difficulty values(1,20),(2,30),(3,40),(4,60),(5,80),(6,100),(7,120);
146
147 •    insert into challenges values(4810,77726,4),(21069,27205,1),(36566,5580,7),
148     (66730,52243,6),(71055,52243,2);
149
150 •    insert into submissions values(68628,77726,36566,30),(65300,77726,21089,10),
151     (40326,52243,36566,77),(8941,27205,4810,4),(83544,77726,66730,30),(43353,52243,66730,6),
152     (55385,52248,71055,20),(39784,27205,71055,23),(94613,86870,71055,30),(45788,52248,36566,8),
153     (93058,86870,36566,30),(7344,8439,66730,92),(2723,8439,4810,36),(523,5580,71055,4),

```

The results grid below the code shows two rows of execution log:

#	Action	Time	Message	Duration / Fetch
31	insert into challenges values(4810,77726,4)	17:15:22	(21069,27205,1),(36566,5580,7), (66730,52243,6),(71055,52243,2)	0.015 sec
32	insert into submissions values(68628,77726,36566,30)	17:27:08	(40326,52243,36566,77),(8941,27205,4810,4),(83544,77726,66730,30),(43353,52243,66730,6), (55385,52248,71055,20),(39784,27205,71055,23),(94613,86870,71055,30),(45788,52248,36566,8), (93058,86870,36566,30),(7344,8439,66730,92),(2723,8439,4810,36),(523,5580,71055,4)	0.015 sec

A message in the top right corner says: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

```

select s.hacker_id, name from submissions as s
join hackers as h on s.hacker_id = h.hacker_id
join challenges as c on s.challenge_id = c.challenge_id
join difficulty as d on c.difficulty_level = d.difficulty_level
where s.score = d.score
group by name, s.hacker_id
having count(s.challenge_id) > 1
order by count(s.challenge_id) desc, s.hacker_id;

```

```

156
157 • select s.hacker_id, name from submissions as s
158 join hackers as h on s.hacker_id = h.hacker_id
159 join challenges as c on s.challenge_id = c.challenge_id
160 join difficulty as d on c.difficulty_level = d.difficulty_level
161 where s.score = d.score
162 group by name, s.hacker_id
163 having count(s.challenge_id) > 1
164 order by count(s.challenge_id) desc, s.hacker_id

```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Q152) Write a query to output the start and end dates of projects listed by the number of days it took to complete the project in ascending order. If there is more than one project that have the same number of completion days, then order by the start date of the project.

Sol) At first create an empty table ‘projects’ and insert values in it using multi insert statement.

```

157
158
159 • create table projects(task_id int,start_date date,end_date date);
160 • insert into projects value(1,'2015-10-01','2015-10-02'),(2,'2015-10-02','2015-10-03'),
161 (3,'2015-10-03','2015-10-04'),(4,'2015-10-13','2015-10-14'),(5,'2015-10-14','2015-10-15'),
162 (6,'2015-10-28','2015-10-29'),(7,'2015-10-30','2015-10-31');
163
164 • select * from projects;

```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

```

select start_date, min(end_date)
from
    (select start_date from projects where start_date not in
    (select end_date from projects))a,
    (select end_date from projects where end_date not in
    (select start_date from projects))b
where start_date < end_date
group by start_date
order by datediff(min(end_date), start_date) asc, start_date
asc;

```

The screenshot shows a database interface with a sidebar containing a tree view of tables and schemas. The main area displays a SQL query and its results. The query is:

```

164 • select * from projects;
165 • select start_date, min(end_date)
  from
    (select start_date from projects where start_date not in (select end_date from projects))a,
    (select end_date from projects where end_date not in (select start_date from projects))b
  where start_date < end_date
  group by start_date
170   order by datediff(min(end_date), start_date) asc, start_date asc;
171

```

The result grid shows the following data:

start_date	min(end_date)
2015-10-28	2015-10-29
2015-10-30	2015-10-31
2015-10-13	2015-10-15
2015-10-01	2015-10-04

Q153) In an effort to identify high-value customers, Amazon asked for your help to obtain data about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more consecutive days. List the user IDs who have gone on at least 1 shopping spree in ascending order.

Sol) At first create an empty an table ‘trans’ and insert values in it.

The screenshot shows a database interface with a sidebar containing a tree view of tables and schemas. The main area displays a SQL query and its results. The query is:

```

180 • create table trans(user_id int,amount int,transaction_date timestamp);
181 • insert into trans values(1,99,STR_TO_DATE("2022/08/01 10:00:00", "%Y/%m/%d %H:%i:%s")),
182   (1,55,STR_TO_DATE("2022/08/17 10:00:00", "%Y/%m/%d %H:%i:%s")),
183   (2,149,5,STR_TO_DATE("2022/08/05 10:00:00", "%Y/%m/%d %H:%i:%s")),
184   (2,4,89,STR_TO_DATE("2022/08/06 10:00:00", "%Y/%m/%d %H:%i:%s")),
185   (2,34,STR_TO_DATE("2022/08/07 10:00:00", "%Y/%m/%d %H:%i:%s"));
186 • select * from trans;

```

The result grid shows the following data:

user_id	amount	transaction_date
1	10	2022-08-01 10:00:00
1	55	2022-08-17 10:00:00
2	150	2022-08-05 10:00:00
2	5	2022-08-06 10:00:00
2	34	2022-08-07 10:00:00

The Action Output pane shows the following log entries:

#	Time	Action	Message	Duration / Fetch
12	08:45:32	insert into trans values(1,99,STR_TO_DATE("2022/08/01 10:00:00", "%Y/%m/%d %H:%i:%s"))	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.015 sec
13	08:45:43	select * from trans	5 row(s) returned	0.000 sec / 0.000 sec

with cte as (

select *, lead(date_diff,1) over(partition by user_id order by transaction_date

) - lead(date_diff,0) over(partition by user_id order by transaction_date)as'diff'

from (

select *,datediff(lead(transaction_date,1) over(partition by user_id order by

transaction_date), lead(transaction_date,0) over(partition by

user_id order by

transaction_date)) as 'date_diff'

from trans) sub),

cte2 as (select user_id,date_diff,diff,

```

        count(date_diff),count(diff) from cte
group by user_id,date_diff,diff
having count(date_diff) >= 2
and date_diff=1 or diff=0)
select user_id from cte2;

```

The screenshot shows a database interface with a code editor containing the following SQL query:

```

        count(date_diff),count(diff) from cte
group by user_id,date_diff,diff
having count(date_diff) >= 2
and date_diff=1 or diff=0)
select user_id from cte2;

```

The results pane shows a single row with the value '2'.

Q154) You are given a table of PayPal payments showing the payer, the recipient, and the amount paid. A two-way unique relationship is established when two people send money back and forth. Write a query to find the number of two-way unique relationships in this data.

Assumption:

- A payer can send money to the same recipient multiple times

Sol) At first create an empty table ‘payments’ and insert values in it.

The screenshot shows a database interface with a code editor containing the following SQL code for creating a table and inserting data:

```

create table payments(player_id int,recipient_id int,amount int);
insert into payments value(101,201,30),(201,101,10),(101,301,20),(301,101,80),
(201,301,70);
select * from payments;

```

The results pane shows the following data:

player_id	recipient_id	amount
101	201	30
201	101	10
101	301	20
301	101	80
201	301	70

The output pane shows the execution details:

- Action Output: 15 09:07:43 insert into payments value(101,201,30),(201,101,10),(101,301,20),(301,101,80), (201,301,70); 5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0
- 16 09:08:02 select * from payments 5 row(s) returned

with cte as

```

(select player_id, recipient_id from payments
intersect
Select recipient_id,player_id
from payments)

```

Select count(payer_id)/2 as unique_relationships
From cte;

```

206 •    select * from payments;
207 •    with cte as
208     (select player_id,recipient_id
209      from payments
210      INTERSECT
211      Select recipient_id,player_id
212      from payments)
213
214      select count(player_id)/2 as unique_relationships
215      from cte

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Result Grid | Read Only | Context Help | Snippets

unique_relationships
2.0000

Action Output

#	Time	Action	Message	Duration / Fetch
43	19:04:55	WITH T1 AS (SELECT PLAYER_ID, RECIPIENT_ID FROM PAYMENTS ...)	Error Code: 1054. Unknown column 'PAYER_ID' in field list'	0.000 sec
44	19:05:33	with t1 as (select player_id, recipient_id from payments INTERSECT Select re...)	1 row(s) returned	0.000 sec / 0.000 sec
45	19:06:15	with cte as (select player_id,recipient_id from payments INTERSECT Select re...)	1 row(s) returned	0.000 sec / 0.000 sec

Q155) Assume you are given the table below containing information on Facebook user logins. Write a query to obtain the number of reactivated users (which are dormant users who did not log in the previous month, then logged in during the current month).
Output the current month (in numerical) and number of reactivated users. Assumption:

- The rows in the user_logins table are complete for the year of 2022 and there are no missing dates.

Sol) At first create an empty table ‘user_logins’ and insert values in it using multi insert statement.

```

195
196 •    create table user_logins(user_id int,login_date datetime);
197 •    insert into user_logins values(725,STR_TO_DATE("2022/03/03 12:00:00", "%Y/%m/%d %H:%i:%s")),
198     (245,STR_TO_DATE("2022/03/28 12:00:00", "%Y/%m/%d %H:%i:%s")),
199     (112,STR_TO_DATE("2022/03/05 12:00:00", "%Y/%m/%d %H:%i:%s")),
200     (245,STR_TO_DATE("2022/04/29 12:00:00", "%Y/%m/%d %H:%i:%s")),
201     (112,STR_TO_DATE("2022/04/05 12:00:00", "%Y/%m/%d %H:%i:%s"));
202 •    select * from user_logins

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Result Grid | Form Editor | Read Only | Context Help | Snippets

user_id	login_date
725	2022-03-03 12:00:00
245	2022-03-28 12:00:00
112	2022-03-05 12:00:00
245	2022-04-29 12:00:00
112	2022-04-05 12:00:00

Action Output

#	Time	Action	Message	Duration / Fetch
20	09:13:14	insert into user_logins values(725,STR_TO_DATE("2022/03/03 12:00:00", "%Y/%m/%d %H:%i:%s"))	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.016 sec
21	09:13:27	select * from user_logins	5 row(s) returned	0.000 sec / 0.000 sec

Q156) Assume you are given the table below on user transactions. Write a query to obtain the list of customers whose first transaction was valued at \$50 or more. Output the number of users.

Clarification:

- Use the `transaction_date` field to determine which transaction should be labeled as the first for each user.
- Use a specific function (we can't give too much away!) to account for scenarios where a user had multiple transactions on the same day, and one of those was the first.

Sol) At first create an empty table 'user_trans' and insert values in it.

The screenshot shows the Oracle SQL Developer interface. In the left sidebar, under the 'Schemas' section, there is a tree view of database objects. A table named 'user_trans' is visible under the 'Tables' node of the 'assignment3' schema. The main pane displays the following SQL code and its execution results:

```

206 • create table user_trans(trans_id int,user_id int,spend decimal,
207   trans_date timestamp);
208 • insert into user_trans values
209   (759274,111,49.50,STR_TO_DATE("2022/02/03 00:00:00", "%Y/%m/%d %H:%i:%s")),
210   (850371,111,51.00,STR_TO_DATE("2022/03/15 00:00:00", "%Y/%m/%d %H:%i:%s")),
211   (615348,145,36.30,STR_TO_DATE("2022/03/22 00:00:00", "%Y/%m/%d %H:%i:%s")),
212   (137424,156,151.00,STR_TO_DATE("2022/04/04 00:00:00", "%Y/%m/%d %H:%i:%s")),
213   (248475,156,87.00,STR_TO_DATE("2022/04/16 00:00:00", "%Y/%m/%d %H:%i:%s"));
214 • select * from user_trans;

```

The result grid shows the following data:

trans_id	user_id	spend	trans_date
759274	111	49.50	2022-02-03 00:00:00
850371	111	51.00	2022-03-15 00:00:00
615348	145	36.30	2022-03-22 00:00:00
137424	156	151.00	2022-04-04 00:00:00
248475	156	87.00	2022-04-16 00:00:00

The output pane shows the execution details:

#	Time	Action	Message	Duration / Fetch
23	09:34:23	insert into user_trans values (759274,111,49.50,STR_TO_DATE("2022/02/03 00:00:00", "%Y/%m/%d %H:%i:%s"))	5 row(s) affected. 2 warning(s). 1265 Data truncated for column 'spend' at row 1 12...	0.016 sec
24	09:34:38	select * from user_trans	5 row(s) returned	0.000 sec / 0.000 sec

with cte as (
select *, row_number() over(partition by user_id order by trans_date)
as rn
from user_trans
order by user_id, trans_date)
select count(user_id) as users from cte where rn=1 and spend>=50;

The screenshot shows the Oracle SQL Developer interface with the following SQL code in the editor:

```

264 • select * from user_trans;
265 • with cte as (
266   select *, row_number() over(partition by user_id order by trans_date) as rn
267   from user_trans
268   order by user_id, trans_date )
269   select count(user_id) as users from cte where rn=1 and spend>=50;

```

The result grid shows the count of users:

users
2

The output pane shows the execution details:

#	Time	Action	Message	Duration / Fetch
1	20:21:29	with cte as (select *, row_number() over(partition by user_id order by trans_date) as rn, count(user_id) as users from cte where rn=1 and spend>=50)	1 row(s) returned	0.000 sec / 0.000 sec

Q157) Assume you are given the table below containing measurement values obtained from a sensor over several days. Measurements are taken several times within a given day. Write a query to obtain the sum of the odd-numbered and even-numbered measurements on a particular day, in two different columns.

Note that the 1st, 3rd, 5th measurements within a day are considered odd-numbered measurements and the 2nd, 4th, 6th measurements are even-numbered measurements.

Sol) At first create an empty table ‘Measurements’ and insert values in it.

```

216 • create table measuremet(m_id int,m_value float,m_time datetime);
217 • insert into measuremet values
218   (131233,1109.51,STR_TO_DATE("2022/07/10 09:00:00", "%Y/%m/%d %H:%i:%s")),
219   (135211,1662.74,STR_TO_DATE("2022/07/10 11:00:00", "%Y/%m/%d %H:%i:%s")),
220   (523542,1246.24,STR_TO_DATE("2022/07/10 13:15:00", "%Y/%m/%d %H:%i:%s")),
221   (143562,1124.59,STR_TO_DATE("2022/07/11 15:00:00", "%Y/%m/%d %H:%i:%s")),
222   (346462,1234.14,STR_TO_DATE("2022/07/11 16:45:00", "%Y/%m/%d %H:%i:%s")));
223
224 • select * from measuremet;

```

The screenshot shows the Oracle SQL Developer interface. On the left, the object browser displays a schema with tables like assignment, assignment1, assignment3, and assignment4. The 'Tables' node under assignment3 contains tables bst, challenges, company, difficult, employee, employees, and fns. The 'Information' pane shows 'No object selected'. The main workspace shows the execution of SQL statements. A message bar at the top right says: 'Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.' Below the message bar, the 'Result Grid' shows the following data:

m_id	m_value	m_time
131233	1109.51	2022-07-10 09:00:00
135211	1662.74	2022-07-10 11:00:00
523542	1246.24	2022-07-10 13:15:00
143562	1124.59	2022-07-11 15:00:00
346462	1234.14	2022-07-11 16:45:00

The 'Output' pane at the bottom shows the execution history:

#	Time	Action	Message	Duration / Fetch
32	13:28:57	select * from measurements	5 row(s) returned	0.000 sec / 0.000 sec
33	13:29:11	select * from measuremet	5 row(s) returned	0.000 sec / 0.000 sec

with m as

```

(select cast(m_time as date) md, m_value mv,
row_number() over(partition by cast(m_time as date)
order by m_time) mn
from measuremet)
select md,sum(case when m.mn % 2 = 0 then mv else 0 end) as even_sum,sum(case when m.mn % 2 != 0 then mv else 0 end) as odd_sum from m
group by md;
```

```

254 • with m as
255   (select cast(m_time as date) md, m_value mv,
256    row_number() over(partition by cast(m_time as date)
257      order by m_time) mn
258    from measurement)
259
260   select md,
261   sum(case when m.mn % 2 = 0 then mv else 0 end) as even_sum,
262   sum(case when m.mn % 2 != 0 then mv else 0 end) as odd_sum
263
264   from m
265   group by md

```

Result Grid | Filter Rows: [] Export: [] Wrap Cell Content: []

	md	even_sum	odd_sum
▶	2022-07-10	1662.739990234375	2355.75
▶	2022-07-11	1234.1400146484375	1124.5

Action Output

#	Time	Action	Message	Duration / Fetch
53	19:38:15	WITH M AS (SELECT CAST(M_TIME AS DATE) MD, M_VALUE MV, ROW_NUM... 2 row(s) returned		0.000 sec / 0.000 sec
54	19:40:10	with m as (select cast(m_time as date) md, m_value mv, row_number() over(partiti... 2 row(s) returned		0.000 sec / 0.000 sec
55	19:40:42	with m as (select cast(m_time as date) md, m_value mv, row_number() over(partiti... 2 row(s) returned		0.000 sec / 0.000 sec

Q158) In an effort to identify high-value customers, Amazon asked for your help to obtain data about users who go on shopping sprees. A shopping spree occurs when a user makes purchases on 3 or more consecutive days. List the user IDs who have gone on at least 1 shopping spree in ascending order.

Sol) At first create an empty table ‘Transaction’ and insert values in it.

```

227 • create table transaction(user_id int,amount int,transaction_date timestamp);
228 • insert into transaction values
229   (1,9.99,STR_TO_DATE("2022/08/01 10:00:00", "%Y/%m/%d %H:%i:%s")),
230   (1,55,STR_TO_DATE("2022/08/17 10:00:00", "%Y/%m/%d %H:%i:%s")),
231   (2,149.5,STR_TO_DATE("2022/08/05 10:00:00", "%Y/%m/%d %H:%i:%s")),
232   (2,4.89,STR_TO_DATE("2022/08/06 10:00:00", "%Y/%m/%d %H:%i:%s")),
233   (2,34,STR_TO_DATE("2022/08/07 10:00:00", "%Y/%m/%d %H:%i:%s"));
234
235 • select * from transaction;

```

Result Grid | Filter Rows: [] Export: [] Wrap Cell Content: []

	user_id	amount	transaction_date
▶	1	10	2022-08-01 10:00:00
▶	1	55	2022-08-17 10:00:00
▶	2	150	2022-08-05 10:00:00
▶	2	5	2022-08-06 10:00:00
▶	2	34	2022-08-07 10:00:00

Action Output

#	Time	Action	Message	Duration / Fetch
38	13:35:20	Insert into transaction values (1,9.99,STR_TO_DATE("2022/08/01 10:00:00", "%Y/%m/%d %H:%i:%s")) 5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0		0.015 sec
39	13:35:36	select * from transaction	5 row(s) returned	0.000 sec / 0.015 sec

with cte as (

```

        select *,lead(transaction_date, 1) over(partition by user_id
          order by transaction_date) - lead(transaction_date, 0) over(
            partition by user_id order by transaction_date) as 'diff'
from( select *, datediff(lead(transaction_date, 1) over(partition by
user_id
order by transaction_date),lead(transaction_date, 0) over(partition
by user_id
order by transaction_date)) as 'date_diff'
from transaction) sub),

```

```

cte2 as (select user_id,date_diff,diff,count(date_diff),count( diff )
from cte group by user_id,date_diff,diff
having count(date_diff) >= 2 and date_diff = 1 or diff = 0)
select user_id from cte2;

```

The screenshot shows a SQL editor interface with the following details:

- Schemas:** A tree view showing various database objects like manager, measurement, occupations, packages, projects, payments, sample, senior_manager, students, submissions, user_trans, user_logins, trans, and useractivity.
- Query Editor:** The main area contains the following SQL code:


```

order by transaction_date) - lead(date_diff, 0) over(
partition by user_id order by transaction_date) as 'diff'
from( select *, datediff(lead(transaction_date, 1) over(partition by user_id
order by transaction_date),lead(transaction_date, 0) over(partition by user_id
order by transaction_date)) as 'date_diff'
from transaction) sub),
cte2 as (select user_id,date_diff,diff,count(date_diff),count( diff )
from cte group by user_id,date_diff,diff
having count(date_diff) >= 2 and date_diff = 1 or diff = 0)
select user_id from cte2;
      
```
- Result Grid:** Below the query editor, it shows the result of the query:

user_id
2
- Information:** A panel on the left showing "Administration Schemas" and "Information".
- Help:** A right-hand sidebar with the message: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

Q159) The Airbnb Booking Recommendations team is trying to understand the "substitutability" of two rentals and whether one rental is a good substitute for another. They want you to write a query to find the unique combination of two Airbnb rentals with the same exact amenities offered. Output the count of the unique combination of Airbnb rentals.

Assumptions:

- If property 1 has a kitchen and pool, and property 2 has a kitchen and pool too, it is a good substitute and represents a unique matching rental.
- If property 3 has a kitchen, pool and fireplace, and property 4 only has a pool and fireplace, then it is not a good substitute.

Sol) At first create an empty 4 table 'Hackers', 'Difficulty' , 'Challenges' & 'Submissions' and insert values in it.

Q160) Google marketing managers are analysing the performance of various advertising accounts over the last month. They need your help to gather the relevant data. Write a query to calculate the return on ad spend (ROAS) for each advertiser across all ad campaigns. Round your answer to 2 decimal places, and order your output by the advertiser_id.

Sol) At first create an empty 4 table 'Hackers' , 'Difficulty' , 'Challenges' & 'Submissions' and insert values in it.

Q161) An employee is considered to be potentially overpaid if they earn more than 2 times the average salary for people with the same title. Similarly, an employee might be underpaid if they earn less than half of the average for their title. We'll refer to employees who are both underpaid and overpaid as compensation outliers for the purposes of this problem.

Write a query that shows the following data for each compensation outlier: employee ID, salary, and whether they are potentially overpaid or potentially underpaid (refer to Example Output below).

Sol)

Q162) SAME AS Q154)

Q163) Assume you are given the table below containing information on user purchases. Write a query to obtain the number of users who purchased the same product on two or more different days. Output the number of unique users.

PS. On 26 Oct 2022, we expanded the purchases data set, thus the official output may vary from before.

Sol) At first create an empty table ‘purchases’ and insert values in it.

The screenshot shows a database interface with a code editor at the top and a result grid at the bottom. The code editor contains the following SQL:

```
287 create table purchases(user_id int,product_id int,quantity int,purchase_date datetime);
288 • insert into purchases values
289 (536,3223,6,STR_TO_DATE("2022/01/11 12:33:44", "%Y/%m/%d %H:%i:%s")),
290 (827,3585,35,STR_TO_DATE("2022/02/20 14:05:26", "%Y/%m/%d %H:%i:%s")),
291 (536,3223,5,STR_TO_DATE("2022/03/02 09:33:28", "%Y/%m/%d %H:%i:%s")),
292 (536,1435,10,STR_TO_DATE("2022/03/02 08:40:00", "%Y/%m/%d %H:%i:%s")),
293 (827,2452,45,STR_TO_DATE("2022/04/09 00:00:00", "%Y/%m/%d %H:%i:%s"));
294 • select * from purchases;
```

The result grid shows the following data:

user_id	product_id	quantity	purchase_date
536	3223	6	2022-01-11 12:33:44
827	3585	35	2022-02-20 14:05:26
536	3223	5	2022-03-02 09:33:28
536	1435	10	2022-03-02 08:40:00
827	2452	45	2022-04-09 00:00:00

```
select count(distinct user_id) as repeat_purchases
from (
    select user_id, rank() over (partition by user_id,
product_id
        order by date(purchase_date) asc
    ) as purchase_no
    from purchases
) as ranking
where purchase_no = 2;
```

The screenshot shows a database interface with a code editor at the top and a result grid at the bottom. The code editor contains the following SQL:

```
295 • select * from purchases;
296 • select count(distinct user_id) as repeat_purchases
297 from (
298     select user_id, rank() over (partition by user_id, product_id
299         order by date(purchase_date) asc
300     ) as purchase_no
301     from purchases
302 ) as ranking
303 where purchase_no = 2;
```

The result grid shows the following data:

repeat_purchases
1

Q164) Assume you are given the table below containing the information on the searches attempted and the percentage of invalid searches by country. Write a query to obtain the percentage of invalid searches. Output the country in ascending order, total searches and overall percentage of invalid searches rounded to 2 decimal places.

Sol) At first create an empty table ‘Search_category’ and insert values in it.

```

303 • create table search_category(country varchar(15),search_cat varchar(15),
304     num_search int,invalid_result_pct float);
305
306 • insert into search_category values('UK','home',null,null),
307     ('UK','tax',98000,1.00),('UK','travel',100000,3.25);
308
309 • select * from search_category;

```

country	search_cat	num_search	invalid_result_pct
UK	home	NULL	NULL
UK	tax	98000	1
UK	travel	100000	3.25

with invalid_results

```

as (select country,num_search,invalid_result_pct,
(case when invalid_result_pct is not null then
num_search
else null end) as num_search_2,
round((num_search * invalid_result_pct)/100.0,0) as
invalid_search from search_category
where num_search is not null
and invalid_result_pct is not null
)

```

```

Select country,sum(num_search_2) as total_search,
round (sum(invalid_search) /sum(num_search_2) *
100.0,2) as invalid_result_pct
from invalid_results group by country order by country;

```

```

313 num_search
314 else null end) as num_search_2,
315 round((num_search * invalid_result_pct)/100.0,0) as
316 invalid_search from search_category
317 where num_search is not null
318 and invalid_result_pct is not null
319 )
320 Select country,sum(num_search_2) as total_search,
321 round (sum(invalid_search) /sum(num_search_2) *
322 100.0,2) as invalid_result_pct
323 from invalid_results group by country order by country;

```

country	total_search	invalid_result_pct
UK	198000	2.14

Q165) Say you have access to all the transactions for a given merchant account. Write a query to print the cumulative balance of the merchant account at the end of each day, with the total balance reset back to zero at the end of the month. Output the transaction date and cumulative balance.

Hint-You should use CASE.

Sol)

Q166) Assume you are given the table below containing information on Amazon customers and their spend on products belonging to various categories. Identify the top two highest-grossing products within each category in 2022. Output the category, product, and total spend.

Sol) At first create an empty 4 table 'product_spend' and insert values in it.

The screenshot shows a SQL database interface with the following details:

- Navigator:** Shows the schema structure with tables like bst, challenges, company, difficulty, employee, employees, friends, hackers, lead_manager, manager, assignment, assignment1, and assignment3.
- SQL Editor:** Contains two queries:
 - Query 1 (Line 328): `insert into product_spend values`
 - Query 2 (Line 329): `('appliance','refrigerator',165,246.00,STR_TO_DATE('2022/12/26 12:00:00', "%Y/%m/%d %H:%i:%s"))`
 - Query 3 (Line 330): `('appliance','refrigerator',123,299.99,STR_TO_DATE('2022/03/02 12:00:00', "%Y/%m/%d %H:%i:%s"))`
 - Query 4 (Line 331): `('appliance','washing machine',123,219.00,STR_TO_DATE('2022/12/26 12:00:00', "%Y/%m/%d %H:%i:%s"))`
 - Query 5 (Line 332): `('electronics','vacuum',178,152.00,STR_TO_DATE('2022/04/05 12:00:00', "%Y/%m/%d %H:%i:%s"))`
 - Query 6 (Line 333): `('electronics','wireless headset',156,249.90,STR_TO_DATE('2022/07/08 12:00:00', "%Y/%m/%d %H:%i:%s"))`
 - Query 7 (Line 334): `('electronics','vacuum',145,189.00,STR_TO_DATE('2022/07/15 12:00:00', "%Y/%m/%d %H:%i:%s"))`
 - Query 8 (Line 335): `select * from product_spend;`
- Result Grid:** Displays the data inserted into the 'product_spend' table:

category	product	user_id	spend	transaction_date
appliance	refrigerator	165	246	2022-12-26 12:00:00
appliance	refrigerator	123	299.99	2022-03-02 12:00:00
appliance	washing machine	123	219.00	2022-12-26 12:00:00
electronics	vacuum	178	152	2022-04-05 12:00:00
electronics	wireless headset	156	249.90	2022-07-08 12:00:00
electronics	vacuum	145	189	2022-07-15 12:00:00
- SQLAdditions:** A note stating: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

```
select category, product, total_spend from (
    select *, rank() over (partition by category
        order by total_spend desc) as ranking
    from ( select category, product, sum(spend) as total_spend
        from product_spend where transaction_date >= '2022-01-01'
            and transaction_date <= '2022-12-31'
        group by category, product) as total_spend
    ) as top_spend
where ranking <= 2
order by category, ranking;
```

The screenshot shows a SQL editor interface with the following details:

- Navigator:** Shows the schema structure with tables like assignment, assignment1, and assignment3.
- SQL Editor:** Displays the following query:


```

      select *, rank() over (partition by category
      order by total_spend desc) as ranking
      from (
        select category, product, sum(spend) as total_spend
        from product_spend
        where transaction_date >= '2022-01-01'
        and transaction_date <= '2022-12-31'
        group by category, product
      ) as top_spending
      where ranking <= 2
      order by category, ranking;
      
```
- Result Grid:** Shows the output of the query:

category	product	total_spend
appliance	refrigerator	545.989990234375
appliance	washing machine	219.8000030517578
electronics	vacuum	341
electronics	wireless headset	249.8999938948438
- Help:** A tooltip message states: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

Q167) Facebook is analysing its user signup data for June 2022. Write a query to generate the churn rate by week in June 2022. Output the week number (1, 2, 3, 4, ...) and the corresponding churn rate rounded to 2 decimal places.

For example, week number 1 represents the dates from 30 May to 5 Jun, and week 2 is from 6 Jun to 12 Jun.

Hint- Use Extract.

Assumptions:

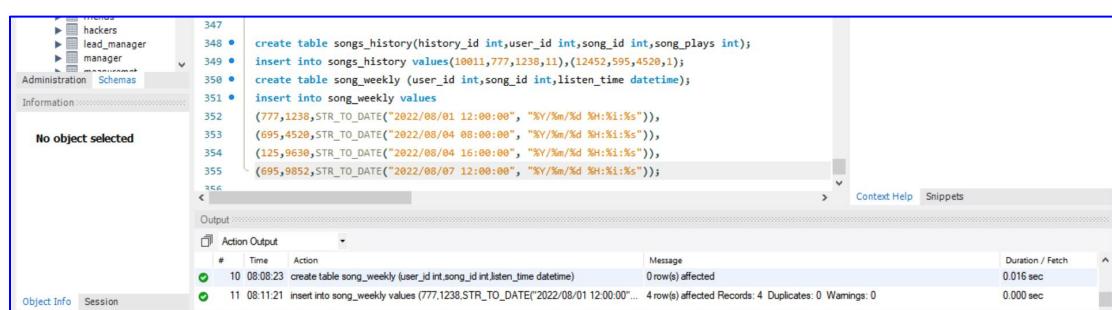
- If the last_login date is within 28 days of the signup_date, the user can be considered churned.
- If the last_login is more than 28 days after the signup date, the user didn't churn.

users Table:

Sol)

Q168) You're given two tables on Spotify users' streaming data. `songs_history` table contains the historical streaming data and `songs_weekly` table contains the current week's streaming data. Write a query to output the user id, song id, and cumulative count of song plays as of 4 August 2022 sorted in descending order.
Hint- Use group by

Sol) At first create 2 empty tables 'Songs_weekly' & 'songs_history' and insert values in it.



```

347
348 • create table songs_history(history_id int,user_id int,song_id int,song_plays int);
349 • insert into songs_history values(10011,777,1238,11),(12452,595,4520,1);
350 • create table song_weekly (user_id int,song_id int,listen_time datetime);
351 • insert into song_weekly values
352 (777,1238,STR_TO_DATE("2022/08/01 12:00:00", "%Y/%m/%d %H:%i:%s")),
353 (695,4520,STR_TO_DATE("2022/08/04 08:00:00", "%Y/%m/%d %H:%i:%s")),
354 (125,9630,STR_TO_DATE("2022/08/04 16:00:00", "%Y/%m/%d %H:%i:%s")),
355 (695,9852,STR_TO_DATE("2022/08/07 12:00:00", "%Y/%m/%d %H:%i:%s"));

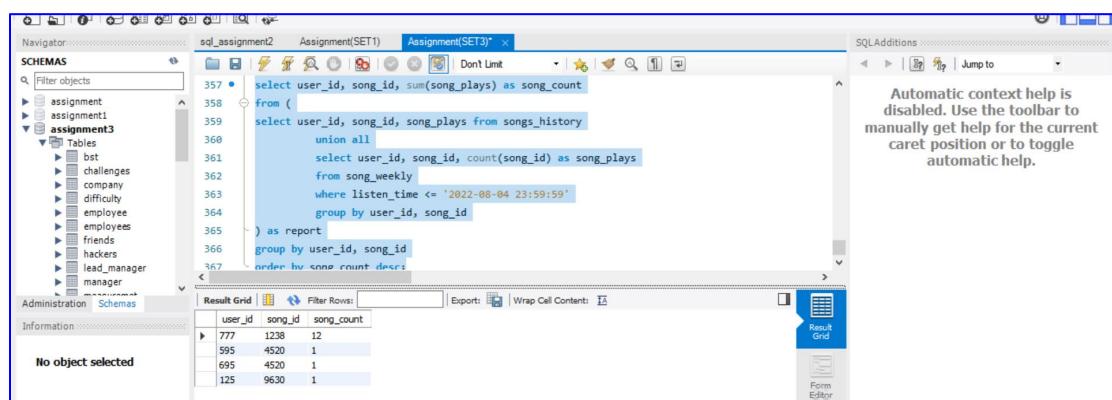
Action Output
# Time Action Message Duration / Fetch
10 08:08:23 create table song_weekly (user_id int,song_id int,listen_time datetime) 0 rows affected 0.016 sec
11 08:11:21 insert into song_weekly values (777,1238,STR_TO_DATE("2022/08/01 12:00:00", "%Y/%m/%d %H:%i:%s")) 4 rows affected Records: 4 Duplicates: 0 Warnings: 0 0.000 sec

```

```

select user_id, song_id, sum(song_plays) as song_count
from (
select user_id, song_id, song_plays from songs_history
union all
select user_id, song_id, count(song_id) as song_plays
from song_weekly
where listen_time <= '2022-08-04 23:59:59'
group by user_id, song_id
) as report
group by user_id, song_id
order by song_count desc;

```



user_id	song_id	song_count
777	1238	12
595	4520	1
695	4520	1
125	9630	1

Q169) New TikTok users sign up with their emails, so each signup requires a text confirmation to activate the new user's account. Write a query to find the confirmation rate of users who confirmed their signups with text messages. Round the result to 2 decimal places.

Hint- Use Joins

Sol) At first create 2 empty tables ‘Emails’ & ‘Texts’ and insert values in it.

```

368
369 • create table emails(email_id int,user_id int,signup_date datetime);
370 • create table texts(text_id int,email_id int,signup_action varchar(15));
371 • insert into emails values(125,7771,STR_TO_DATE("2022/06/14 00:00:00", "%Y/%m/%d %H:%i:%s"));
372 (236,6950,STR_TO_DATE("2022/07/01 00:00:00", "%Y/%m/%d %H:%i:%s"));
373 (433,1052,STR_TO_DATE("2022/07/09 00:00:00", "%Y/%m/%d %H:%i:%s"));
374 • insert into texts values(6878,125,'confirmed'),
375 (6920,236,'Not confirmed'),(6994,236,'Confirmed');

```

Action	Time	Message	Duration / Fetch
insert into emails values(125,7771,STR_TO_DATE("2022/06/14 00:00:00", "%Y/%m/%d %H:%i:%s"))	20 08:22:52	3 rows affected Records: 3 Duplicates: 0 Warnings: 0	0.016 sec
insert into texts values(6878,125,'confirmed'),(6920,236,'Not confirmed'),(6994,236,'Confirmed')	21 08:23:59	3 rows affected Records: 3 Duplicates: 0 Warnings: 0	0.015 sec

Select round(sum(signup) / count(user_id), 2) as confirmation_rate from (select user_id, (case when texts.email_id is not null then 1 else 0 End) as signup from emails left join texts on emails.email_id = texts.email_id and signup_action = 'confirmed') as rate;

The screenshot shows the MySQL Workbench interface with the query editor containing the code provided above. The results pane displays a single row with the value '0.67' under the column 'confirmation_rate'. A tooltip in the results pane states: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

Q170) The table below contains information about tweets over a given period of time. Calculate the 3-day rolling average of tweets published by each user for each date that a tweet was posted. Output the user id, tweet date, and rolling averages rounded to 2 decimal places.

Hint- Use Count and group by

Important Assumptions:

- Rows in this table are *consecutive* and ordered by date.
 - Each row represents a different day
 - A day that does not correspond to a row in this table is not counted.
- The most recent day is the next row above the current row

Sol) At first create an empty table ‘Tweets’ and insert values in it.

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the schema 'assignment3', there is a 'Tables' section containing a single table named 'tweets'. The table has three columns: 'tweet_id' (int), 'user_id' (int), and 'tweet_date' (timestamp). Below the table definition, several rows of data are listed, all inserted on June 1st, 2022, at 12:00:00. The data includes user IDs 111, 254, and 111 again. The 'Result Grid' tab is selected, showing the inserted data. The 'Action Output' tab shows the SQL commands used: 'insert into tweets values' followed by the five inserted rows, and 'select * from tweets'.

```

388 • create table tweets(tweet_id int,user_id int,tweet_date timestamp);
389 • insert into tweets values
390 (214252,111,STR_TO_DATE("2022/06/01 12:00:00", "%Y/%m/%d %H:%i:%s")),
391 (739252,111,STR_TO_DATE("2022/06/01 12:00:00", "%Y/%m/%d %H:%i:%s")),
392 (846402,111,STR_TO_DATE("2022/06/02 12:00:00", "%Y/%m/%d %H:%i:%s")),
393 (241425,254,STR_TO_DATE("2022/06/02 12:00:00", "%Y/%m/%d %H:%i:%s")),
394 (137374,111,STR_TO_DATE("2022/06/04 12:00:00", "%Y/%m/%d %H:%i:%s"));
395 • select * from tweets;
396

```

```

select user_id, tweet_date,
       round( avg(tweet_num) over (partition by user_id
                                     order by user_id,
                                             tweet_date rows between 2 preceding and current row), 2)
as rolling_avg_3d
from ( select user_id, tweet_date,
              count(distinct tweet_id) as tweet_num
        from tweets
      group by user_id, tweet_date) as tweet_count;

```

The screenshot shows the MySQL Workbench interface. The same schema and table structure as the previous screenshot are visible. The 'Result Grid' tab shows the output of the query, which includes columns 'user_id', 'tweet_date', and 'rolling_avg_3d'. The data shows user ID 111 with three distinct tweet dates (June 1st, 2nd, 4th) and their corresponding rolling averages (2.00, 1.60, 1.33). The 'Action Output' tab shows the executed SQL query and its execution details.

```

397 •
398 • select user_id, tweet_date,
399       round( avg(tweet_num) over (partition by user_id
400             order by user_id,
401                 tweet_date rows between 2 preceding and current row), 2) as rolling_avg_3d
402   from ( select user_id, tweet_date,
403               count(distinct tweet_id) as tweet_num
404         from tweets
405       group by user_id, tweet_date) as tweet_count;
406

```

Q171) Assume you are given the tables below containing information on Snapchat users, their ages, and their time spent sending and opening snaps. Write a query to obtain a breakdown of the time spent sending vs. opening snaps (as a percentage of total time spent on these activities) for each age group.

Hint- Use join and case Output the age bucket and percentage of sending and opening snaps. Round the percentage to 2 decimal places.

Sol) At first create 2 empty tables 'Activities' & 'Age_breakdown' and insert values in it.

```

406 • create table activities(activity_id int,user_id int,
407     activity_type varchar(15),time_spent float,activity_date datetime);
408
409 • insert into activities values
410     (7274,123,'open',4.50,STR_TO_DATE("2022/06/22 12:00:00", "%Y/%m/%d %H:%i:%s")),
411     (2425,123,'send',3.50,STR_TO_DATE("2022/06/22 12:00:00", "%Y/%m/%d %H:%i:%s")),
412     (1413,456,'send',5.67,STR_TO_DATE("2022/06/23 12:00:00", "%Y/%m/%d %H:%i:%s")),
413     (1414,789,'chat',11.00,STR_TO_DATE("2022/06/25 12:00:00", "%Y/%m/%d %H:%i:%s"));
414
415 • create table age_breakdown(user_id int,age_bucket varchar(15));
416 • insert into age_breakdown values(123,31-35),(456,26-30),(789,21-25);

Action Output
# Time Action
30 08:47:45 create table age_breakdown(user_id int,age_bucket varchar(15))
31 08:48:27 insert into age_breakdown values(123,31-35),(456,26-30),(789,21-25)

Message
0 row(s) affected
3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0

Duration / Fetch
0.031 sec
0.016 sec

```

with **snaps_statistics** as (select age.age_bucket,
sum(case when activities.activity_type = 'send'
then activities.time_spent else 0 end) as **send_timespent**,
sum(case when activities.activity_type = 'open'
then activities.time_spent else 0 end) as **open_timespent**,
sum(activities.time_spent) as **total_timespent**
from activities
inner join age_breakdown as age
on activities.user_id = age.user_id
where activities.activity_type in ('send', 'open')
group by age.age_bucket)

select age_bucket,
round(100.0*send_timespent/total_timespent,2)as
send_perc,
round(100.0 * open_timespent / total_timespent, 2) as
open_perc
from snaps_statistics;

```

424 from activities
425 inner join age_breakdown as age
426   on activities.user_id = age.user_id
427 where activities.activity_type in ('send', 'open')
428 group by age.age_bucket
429
430 select age_bucket,
431   round(100.0 * send_timespent / total_timespent, 2) as send_perc,
432   round(100.0 * open_timespent / total_timespent, 2) as open_perc
433
434 from snaps_statistics;

```

age_bucket	send_perc	open_perc
31-35	43.75	56.25
26-30	100	0

Action Output

#	Time	Action	Message	Duration / Fetch
37	08:58:01	insert into age_breakdown values(123,31-35),(456,26-30),(789,21-25)	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.000 sec
38	08:58:14	insert into snaps_statistics as (select age.age_bucket, sum(case when activities.activity_type in ('send', 'open') then 1 else 0 end) as count, sum(case when activities.activity_type = 'send' then 1 else 0 end) as send_count, sum(case when activities.activity_type = 'open' then 1 else 0 end) as open_count, round(100.0 * send_timespent / total_timespent, 2) as send_perc, round(100.0 * open_timespent / total_timespent, 2) as open_perc from activities inner join age_breakdown as age on activities.user_id = age.user_id where activities.activity_type in ('send', 'open') group by age.age_bucket)	2 row(s) returned	0.000 sec / 0.000 sec

Q172) The LinkedIn Creator team is looking for power creators who use their personal profile as a company or influencer page. This means that if someone's LinkedIn page has more followers than all the companies they work for, we can safely assume that person is a Power Creator. Keep in mind that if a person works at multiple companies, we should take into account the company with the most followers.

Write a query to return the IDs of these LinkedIn power creators in ascending order

Sol) At first create 3 empty tables ‘personal_profile’, ‘employee_company’ & ‘company_pages’ and insert values in it using multi insert command.

```

435 • create table personal_profiles(profile_id int, name varchar(15), followers int);
436 • insert into personal_profiles values(1, 'Nick Singh', 92000), (2, 'Zach Wilson', 199000),
437   (3, 'Dollana Liu', 171000), (4, 'Ravit Jain', 107000), (5, 'Vin Vashista', 139000),
438   (6, 'Susan Wojcicki', 39000);
439
440 • create table employee_company(personal_profile_id int, company_id int);
441 • insert into employee_company values(1,4), (1,9), (2,2), (3,1), (4,3);
442
443 • create table company_pages(company_id int, name varchar(35), followers int);
444 • insert into company_pages values(1, 'The data science podcast', 8000),
445   (2, 'Airbnb', 700000), (3, 'The Ravit Show', 6000), (4, 'Datalemur', 200),
446   (5, 'Youtube', 16000000), (6, 'DataScience Vin', 4500), (7, 'Ace the Data Science Interview', 4479);

```

Action Output

#	Time	Action	Message	Duration / Fetch
46	09:32:01	create table company_pages(company_id int, name varchar(35), followers int)	0 row(s) affected	0.047 sec
47	09:34:27	insert into company_pages values(1, 'The data science podcast', 8000), (2, 'Airbnb', 700000), (3, 'The Ravit Show', 6000), (4, 'Datalemur', 200), (5, 'Youtube', 16000000), (6, 'DataScience Vin', 4500), (7, 'Ace the Data Science Interview', 4479)	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0	0.016 sec

```

select distinct p.profile_id from personal_profiles p inner
join employee_company e
on p.profile_id=e.personal_profile_id inner join
company_pages c
on e.company_id=c.company_id
where p.followers>c.followers and p.profile_id not in

```

```

    ( select p.profile_id from personal_profiles p inner join
employee_company e
        on p.profile_id=e.personal_profile_id
        inner join company_pages c on
e.company_id=c.company_id
        where p.followers<c.followers)
    order by p.profile_id;

```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

```

447
448 • select distinct p.profile_id from personal_profiles p inner join employee_company e
449 on p.profile_id=e.personal_profile_id inner join company_pages c
450 on e.company_id=c.company_id
451 where p.followers<c.followers and p.profile_id not in
452 ( select p.profile_id from personal_profiles p inner join employee_company e
453 on p.profile_id=e.personal_profile_id
454 inner join company_pages c on e.company_id=c.company_id
455 where p.followers<=c.followers)
456 order by p.profile_id;

```

profile_id
1
3
4
5

Result 18 x

Action Output	#	Time	Action	Message	Duration / Fetch
58 09:39:07	select distinct p.profile_id from personal_profiles p inner join employee_company e o... 4 row(s) returned	0.016 sec / 0.000 sec			
59 09:39:58	select distinct p.profile_id from personal_profiles p inner join employee_company e o... 4 row(s) returned	0.000 sec / 0.000 sec			