

1077. Project Employees III

Description

Table: Project

Column Name	Type
project_id	int
employee_id	int

(project_id, employee_id) is the primary key of this table.
employee_id is a foreign key to Employee table.

Table: Employee

Column Name	Type
employee_id	int
name	varchar
experience_years	int

employee_id is the primary key of this table.

Write an SQL query that reports the most experienced employees in each project. In case of a tie, report all employees with the maximum number of experience years.

The query result format is in the following example:

Project table:

project_id	employee_id
1	1
1	2
1	3
2	1
2	4

Employee table:

employee_id	name	experience_years
1	Khaled	3

2	Ali	2	
3	John	3	
4	Doe	2	
+-----+-----+-----+			

Result table:

+-----+-----+			
project_id	employee_id		
+-----+-----+			
1	1		
1	3		
2	1		
+-----+-----+			

Both employees with id 1 and 3 have the most experience among the employees of the first project. For the second project, the employee with id 1 has the most experience.

Solution

01/19/2020 (MySQL):

```
# Write your MySQL query statement below
select p.project_id, e.employee_id
from
(
    select project_id, max(experience_years) as max_years
    from
        Project as p
        join
        Employee as e
        on p.employee_id = e.employee_id
    group by project_id
) as q,
Project as p,
Employee as e
where p.project_id = q.project_id and p.employee_id = e.employee_id and
e.experience_years >= max_years;
```

01/19/2020 (MySQL):

```
# Write your MySQL query statement below
select p.project_id, e.employee_id
from Project as p, Employee as e
where p.employee_id = e.employee_id and (p.project_id, e.experience_years) in (
    select project_id, max(experience_years) as experience_years
    from Project as p join Employee as e on p.employee_id = e.employee_id
    group by project_id
)
```

1082. Sales Analysis I

Description

Table: Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product_id is the primary key of this table.

Table: Sales

Column Name	Type
seller_id	int
product_id	int
buyer_id	int
sale_date	date
quantity	int
price	int

This table has no primary key, it can have repeated rows.

product_id is a foreign key to Product table.

Write an SQL query that reports the best seller by total sales price, If there is a tie, report them all.

The query result format is in the following example:

Product table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales table:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

Result table:

seller_id
1
3

Both sellers with id 1 and 3 sold products with the most total price of 2800.

Solution

01/13/2020:

```
# Write your MySQL query statement below
select seller_id
from Sales
group by seller_id
having sum(price) >= (
    select sum(price) as total_price
    from Sales
    group by seller_id
    order by total_price desc
    limit 1);

# select seller_id
# from Sales
# group by seller_id
# having sum(price) >= all(
#     select sum(price)
#     from Sales
```

```
# group by seller_id
# );
```

1083. Sales Analysis II

Description

Table: Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product_id is the primary key of this table.

Table: Sales

Column Name	Type
seller_id	int
product_id	int
buyer_id	int
sale_date	date
quantity	int
price	int

This table has no primary key, it can have repeated rows.

product_id is a foreign key to Product table.

Write an SQL query that reports the buyers who have bought S8 but not iPhone.
Note that S8 and iPhone are products present in the Product table.

The query result format is in the following example:

Product table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales table:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	1	3	2019-06-02	1	800
3	3	3	2019-05-13	2	2800

Result table:

buyer_id
1

The buyer with id 1 bought an S8 but didn't buy an iPhone. The buyer with id 3 bought both.

Solution

01/18/2020:

```
# Write your MySQL query statement below
select distinct s.buyer_id
from Sales as s join Product as p on s.product_id = p.product_id
where product_name = 'S8' and s.buyer_id not in (
    select buyer_id
    from Sales as s join Product as p on s.product_id = p.product_id
    where product_name = 'iPhone'
);
```

01/18/2020:

```
# Write your MySQL query statement below
select buyer_id
from Sales join Product using(product_id)
group by buyer_id
having sum(product_name = 'S8') > 0 and sum(product_name = 'iPhone') = 0;
```

1084. Sales Analysis III

Description

Table: Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product_id is the primary key of this table.

Table: Sales

Column Name	Type
seller_id	int
product_id	int
buyer_id	int
sale_date	date
quantity	int
price	int

This table has no primary key, it can have repeated rows.

product_id is a foreign key to Product table.

Write an SQL query that reports the products that were only sold in spring 2019. That is, between 2019-01-01 and 2019-03-31 inclusive.

The query result format is in the following example:

Product table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales table:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

+-----+-----+-----+-----+-----+

Result table:

+-----+-----+	
product_id	product_name
+-----+-----+	
1	S8
+-----+-----+	

The product with id 1 was only sold in spring 2019 while the other two were sold after.

Solution

01/18/2020:

```
select product_id, product_name
from Sales inner join product using(product_id)
group by product_id
having sum(if(sale_date between '2019-01-01' and '2019-03-31', 1, 0)) =
sum(if(sale_date, 1, 0));
```

1112. Highest Grade For Each Student

Description

Table: Enrollments

+-----+-----+		
Column Name	Type	
+-----+-----+		
student_id	int	
course_id	int	
grade	int	
+-----+-----+		

(student_id, course_id) is the primary key of this table.

Write a SQL query to find the highest grade with its corresponding course for each student. In case of a tie, you should find the course with the smallest course_id. The output must be sorted by increasing student_id.

The query result format is in the following example:

Enrollments table:

+-----+-----+		
student_id	course_id	grade

2	2	95
2	3	95
1	1	90
1	2	99
3	1	80
3	2	75
3	3	82

Result table:

student_id	course_id	grade
1	2	99
2	2	95
3	3	82

Solution

01/21/2020 (MySQL):

```
# Write your MySQL query statement below
select student_id, min(course_id) as course_id, grade
from Enrollments
where (student_id, grade) in (
  select student_id, max(grade)
  from Enrollments
  group by student_id
)
group by student_id
order by student_id asc;
```

1113. Reported Posts

Description

Table: Actions

Column Name	Type	
user_id	int	
post_id	int	

action_date	date	
action	enum	
extra	varchar	
+-----+-----+		

There is no primary key for this table, it may have duplicate rows.

The action column is an ENUM type of ('view', 'like', 'reaction', 'comment', 'report', 'share').

The extra column has optional information about the action such as a reason for report or a type of reaction.

Write an SQL query that reports the number of posts reported yesterday for each report reason. Assume today is 2019-07-05.

The query result format is in the following example:

Actions table:

+-----+-----+-----+-----+-----+					
user_id	post_id	action_date	action	extra	
+-----+-----+-----+-----+-----+					
1	1	2019-07-01	view	null	
1	1	2019-07-01	like	null	
1	1	2019-07-01	share	null	
2	4	2019-07-04	view	null	
2	4	2019-07-04	report	spam	
3	4	2019-07-04	view	null	
3	4	2019-07-04	report	spam	
4	3	2019-07-02	view	null	
4	3	2019-07-02	report	spam	
5	2	2019-07-04	view	null	
5	2	2019-07-04	report	racism	
5	5	2019-07-04	view	null	
5	5	2019-07-04	report	racism	
+-----+-----+-----+-----+-----+					

Result table:

+-----+-----+		
report_reason	report_count	
+-----+-----+		
spam	1	
racism	2	
+-----+-----+		

Note that we only care about report reasons with non zero number of reports.

Solution

01/14/2020:

```
# Write your MySQL query statement below
select extra as report_reason, count(*) as report_count
from (
  select post_id, extra
  from Actions
  where action_date = '2019-07-04' and action = 'report'
  group by post_id, extra) as t
group by t.extra;
```

01/14/2020:

```
# Write your MySQL query statement below
select extra as report_reason, count(distinct post_id) as report_count
from Actions
where action_date = '2019-07-04' and action = 'report'
group by extra;
```

1126. Active Businesses

Description

Table: Events

Column Name	Type
business_id	int
event_type	varchar
occurences	int

(business_id, event_type) is the primary key of this table.

Each row in the table logs the info that an event of some type occurred at some business for a number of times.

Write an SQL query to find all active businesses.

An active business is a business that has more than one event type with occurrences greater than the average occurrences of that event type among all businesses.

The query result format is in the following example:

Events table:

Column Name	Type
-------------	------

business_id	event_type	occurrences
1	reviews	7
3	reviews	3
1	ads	11
2	ads	7
3	ads	6
1	page views	3
2	page views	12

Result table:

business_id
1

Average for 'reviews', 'ads' and 'page views' are $(7+3)/2=5$, $(11+7+6)/3=8$, $(3+12)/2=7.5$ respectively.

Business with id 1 has 7 'reviews' events (more than 5) and 11 'ads' events (more than 8) so it is an active business.

Solution

01/21/2020:

```
# Write your MySQL query statement below
select business_id
from Events e,
(
    select event_type, avg(occurrences) as avg_occurrences
    from Events
    group by event_type
) as a
where e.event_type = a.event_type and e.occurrences > a.avg_occurrences
group by e.business_id
having count(*) > 1;
```

1141. User Activity for the Past 30 Days I

Description

Table: Activity

|--|

Column Name	Type
user_id	int
session_id	int
activity_date	date
activity_type	enum

There is no primary key for this table, it may have duplicate rows.

The activity_type column is an ENUM of type ('open_session', 'end_session', 'scroll_down', 'send_message').

The table shows the user activities for a social media website.

Note that each session belongs to exactly one user.

Write an SQL query to find the daily active user count for a period of 30 days ending 2019-07-27 inclusively. A user was active on some day if he/she made at least one activity on that day.

The query result format is in the following example:

Activity table:

user_id	session_id	activity_date	activity_type
1	1	2019-07-20	open_session
1	1	2019-07-20	scroll_down
1	1	2019-07-20	end_session
2	4	2019-07-20	open_session
2	4	2019-07-21	send_message
2	4	2019-07-21	end_session
3	2	2019-07-21	open_session
3	2	2019-07-21	send_message
3	2	2019-07-21	end_session
4	3	2019-06-25	open_session
4	3	2019-06-25	end_session

Result table:

day	active_users
2019-07-20	2
2019-07-21	2

Note that we do not care about days with zero active users.

Solution

01/18/2020:

```
# Write your MySQL query statement below
select activity_date as day, count(distinct user_id) as active_users
from Activity
where activity_date between '2019-06-28' and '2019-07-27'
group by day;
```

1142. User Activity for the Past 30 Days II

Description

Table: Activity

Column Name	Type
user_id	int
session_id	int
activity_date	date
activity_type	enum

There is no primary key for this table, it may have duplicate rows.

The activity_type column is an ENUM of type ('open_session', 'end_session', 'scroll_down', 'send_message').

The table shows the user activities for a social media website.

Note that each session belongs to exactly one user.

Write an SQL query to find the average number of sessions per user for a period of 30 days ending 2019-07-27 inclusively, rounded to 2 decimal places. The sessions we want to count for a user are those with at least one activity in that time period.

The query result format is in the following example:

Activity table:

user_id	session_id	activity_date	activity_type
1	1	2019-07-20	open_session
1	1	2019-07-20	scroll_down
1	1	2019-07-20	end_session
2	4	2019-07-20	open_session
2	4	2019-07-21	send_message

2	4	2019-07-21	end_session	
3	2	2019-07-21	open_session	
3	2	2019-07-21	send_message	
3	2	2019-07-21	end_session	
3	5	2019-07-21	open_session	
3	5	2019-07-21	scroll_down	
3	5	2019-07-21	end_session	
4	3	2019-06-25	open_session	
4	3	2019-06-25	end_session	
+-----+-----+-----+-----+				

Result table:

+-----+	
average_sessions_per_user	
+-----+	
1.33	
+-----+	

User 1 and 2 each had 1 session in the past 30 days while user 3 had 2 sessions so the average is $(1 + 1 + 2) / 3 = 1.33$.

Solution

01/18/2020:

```
# Write your MySQL query statement below
select round(ifnull(sum(sessions) / count(user_id), 0), 2) as
average_sessions_per_user
from (
  select distinct user_id, count(distinct session_id) as sessions
  from Activity
  where activity_date between '2019-06-28' and '2019-07-27'
  group by user_id
  having count(*) >= 1
) as u;
```

1148. Article Views I

Description

Table: Views

+-----+-----+		
Column Name	Type	
+-----+-----+		
article_id	int	

```
| author_id | int |
| viewer_id | int |
| view_date | date |
+-----+
```

There is no primary key for this table, it may have duplicate rows.

Each row of this table indicates that some viewer viewed an article (written by some author) on some date.

Note that equal author_id and viewer_id indicate the same person.

Write an SQL query to find all the authors that viewed at least one of their own articles, sorted in ascending order by their id.

The query result format is in the following example:

Views table:

```
+-----+-----+-----+-----+
| article_id | author_id | viewer_id | view_date |
+-----+-----+-----+-----+
| 1          | 3         | 5         | 2019-08-01 |
| 1          | 3         | 6         | 2019-08-02 |
| 2          | 7         | 7         | 2019-08-01 |
| 2          | 7         | 6         | 2019-08-02 |
| 4          | 7         | 1         | 2019-07-22 |
| 3          | 4         | 4         | 2019-07-21 |
| 3          | 4         | 4         | 2019-07-21 |
+-----+-----+-----+-----+
```

Result table:

```
+-----+
| id    |
+-----+
| 4     |
| 7     |
+-----+
```

Solution

01/13/2020:

```
# Write your MySQL query statement below
select distinct author_id as id
from Views
where author_id = viewer_id
order by author_id;
```


1164. Product Price at a Given Date

Description

Table: Products

Column Name	Type
product_id	int
new_price	int
change_date	date

(product_id, change_date) is the primary key of this table.

Each row of this table indicates that the price of some product was changed to a new price at some date.

Write an SQL query to find the prices of all products on 2019-08-16. Assume the price of all products before any change is 10.

The query result format is in the following example:

Products table:

product_id	new_price	change_date
1	20	2019-08-14
2	50	2019-08-14
1	30	2019-08-15
1	35	2019-08-16
2	65	2019-08-17
3	20	2019-08-18

Result table:

product_id	price
2	50
1	35
3	10

Solution

01/22/2020:

Write your MySQL query statement below

```
select
  i.product_id,
  max(if(i.product_id not in (select product_id from Products where change_date
    <= date '2019-08-16' group by product_id), 10, (select new_price from Products
    where product_id = i.product_id and product_id = q.product_id and change_date =
    q.max_change_date))) as price
from
  (select distinct product_id from Products) as i,
  (
    select product_id, max(change_date) as max_change_date
    from Products
    where change_date <= date '2019-08-16'
    group by product_id
  ) as q
group by i.product_id;
```

1173. Immediate Food Delivery I

Description

Table: Delivery

Column Name	Type
delivery_id	int
customer_id	int
order_date	date
customer_pref_delivery_date	date

delivery_id is the primary key of this table.

The table holds information about food delivery to customers that make orders at some date and specify a preferred delivery date (on the same order date or after it).

If the preferred delivery date of the customer is the same as the order date then the order is called immediate otherwise it's called scheduled.

Write an SQL query to find the percentage of immediate orders in the table, rounded to 2 decimal places.

The query result format is in the following example:

Delivery table:

delivery_id	customer_id	order_date	customer_pref_delivery_date
1	1	2019-08-01	2019-08-02
2	5	2019-08-02	2019-08-02
3	1	2019-08-11	2019-08-11
4	3	2019-08-24	2019-08-26
5	4	2019-08-21	2019-08-22
6	2	2019-08-11	2019-08-13

Result table:

immediate_percentage
33.33

The orders with delivery id 2 and 3 are immediate while the others are scheduled.

Solution

01/13/2020:

```
# Write your MySQL query statement below
# select round(
#   (select count(*) from Delivery where order_date =
customer_pref_delivery_date) /
#   (select count(*) from Delivery) * 100,
#   2) as immediate_percentage;

select round(
  sum(case when order_date = customer_pref_delivery_date then 1 else 0 end) /
count(delivery_id) * 100
, 2) as immediate_percentage
from Delivery;
```

1174. Immediate Food Delivery II

Description

Table: Delivery

Column Name	Type
delivery_id	int
customer_id	int
order_date	date
customer_pref_delivery_date	date

delivery_id is the primary key of this table.

The table holds information about food delivery to customers that make orders at some date and specify a preferred delivery date (on the same order date or after it).

If the preferred delivery date of the customer is the same as the order date then the order is called immediate otherwise it's called scheduled.

The first order of a customer is the order with the earliest order date that customer made. It is guaranteed that a customer has exactly one first order.

Write an SQL query to find the percentage of immediate orders in the first orders of all customers, rounded to 2 decimal places.

The query result format is in the following example:

Delivery table:

delivery_id	customer_id	order_date	customer_pref_delivery_date
1	1	2019-08-01	2019-08-02
2	2	2019-08-02	2019-08-02
3	1	2019-08-11	2019-08-12
4	3	2019-08-24	2019-08-24
5	3	2019-08-21	2019-08-22
6	2	2019-08-11	2019-08-13
7	4	2019-08-09	2019-08-09

Result table:

immediate_percentage
50.00

The customer id 1 has a first order with delivery id 1 and it is scheduled. The customer id 2 has a first order with delivery id 2 and it is immediate. The customer id 3 has a first order with delivery id 5 and it is scheduled. The customer id 4 has a first order with delivery id 7 and it is immediate. Hence, half the customers have immediate first orders.

Solution

01/22/2020:

```
# Write your MySQL query statement below
select round(sum(if(order_date = customer_pref_delivery_date, 1, 0)) / count(*)
* 100, 2) as immediate_percentage
from Delivery
where (customer_id, order_date) in (
    select customer_id, min(order_date)
    from Delivery
    group by customer_id
)
```

1179. Reformat Department Table

Description

Table: Department

Column Name	Type
id	int
revenue	int
month	varchar

(id, month) is the primary key of this table.

The table has information about the revenue of each department per month.

The month has values in

["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"].

Write an SQL query to reformat the table such that there is a department id column and a revenue column for each month.

The query result format is in the following example:

Department table:

id	revenue	month
1	8000	Jan
2	9000	Jan

3	10000	Feb
1	7000	Feb
1	6000	Mar

Result table:

id	Jan_Revenue	Feb_Revenue	Mar_Revenue	...	Dec_Revenue
1	8000	7000	6000	...	null
2	9000	null	null	...	null
3	null	10000	null	...	null

Note that the result table has 13 columns (1 for the department id + 12 for the months).

Solution

01/13/2020:

```
# Write your MySQL query statement below
# select
# id,
# sum(case when month = 'Jan' then revenue else null end) as Jan_Revenue,
# sum(case when month = 'Feb' then revenue else null end) as Feb_Revenue,
# sum(case when month = 'Mar' then revenue else null end) as Mar_Revenue,
# sum(case when month = 'Apr' then revenue else null end) as Apr_Revenue,
# sum(case when month = 'May' then revenue else null end) as May_Revenue,
# sum(case when month = 'Jun' then revenue else null end) as Jun_Revenue,
# sum(case when month = 'Jul' then revenue else null end) as Jul_Revenue,
# sum(case when month = 'Aug' then revenue else null end) as Aug_Revenue,
# sum(case when month = 'Sep' then revenue else null end) as Sep_Revenue,
# sum(case when month = 'Oct' then revenue else null end) as Oct_Revenue,
# sum(case when month = 'Nov' then revenue else null end) as Nov_Revenue,
# sum(case when month = 'Dec' then revenue else null end) as Dec_Revenue
# from Department
# group by id;

select
id,
sum(if(month = 'Jan', revenue, null)) as Jan_Revenue,
sum(if(month = 'Feb', revenue, null)) as Feb_Revenue,
sum(if(month = 'Mar', revenue, null)) as Mar_Revenue,
sum(if(month = 'Apr', revenue, null)) as Apr_Revenue,
sum(if(month = 'May', revenue, null)) as May_Revenue,
sum(if(month = 'Jun', revenue, null)) as Jun_Revenue,
sum(if(month = 'Jul', revenue, null)) as Jul_Revenue,
```

```

sum(if(month = 'Aug', revenue, null)) as Aug_Revenue,
sum(if(month = 'Sep', revenue, null)) as Sep_Revenue,
sum(if(month = 'Oct', revenue, null)) as Oct_Revenue,
sum(if(month = 'Nov', revenue, null)) as Nov_Revenue,
sum(if(month = 'Dec', revenue, null)) as Dec_Revenue
from Department
group by id;

```

1193. Monthly Transactions I

Description

Table: Transactions

Column Name	Type
id	int
country	varchar
state	enum
amount	int
trans_date	date

id is the primary key of this table.

The table has information about incoming transactions.

The state column is an enum of type ["approved", "declined"].

Write an SQL query to find for each month and country, the number of transactions and their total amount, the number of approved transactions and their total amount.

The query result format is in the following example:

Transactions table:

id	country	state	amount	trans_date
121	US	approved	1000	2018-12-18
122	US	declined	2000	2018-12-19
123	US	approved	2000	2019-01-01
124	DE	approved	2000	2019-01-07

Result table:

month	country	trans_count	approved_count	trans_total_amount	approved_total_amount
2018-12	US	2	1	3000	1000
2019-01	US	1	1	2000	2000
2019-01	DE	1	1	2000	2000

Solution

01/21/2020 (MySQL):

```
# Write your MySQL query statement below
select
  date_format(trans_date, '%Y-%m') as month, country,
  count(*) as trans_count,
  sum(if(state='approved', 1, 0)) as approved_count,
  sum(amount) as trans_total_amount,
  sum(if(state='approved', amount, 0)) as approved_total_amount
from Transactions
group by date_format(trans_date, '%Y-%m'), country;
```

1204. Last Person to Fit in the Elevator

Description

Table: Queue

Column Name	Type
person_id	int
person_name	varchar
weight	int
turn	int

person_id is the primary key column for this table.
This table has the information about all people waiting for an elevator.

The person_id and turn columns will contain all numbers from 1 to n, where n is the number of rows in the table.

The maximum weight the elevator can hold is 1000.

Write an SQL query to find the person_name of the last person who will fit in the elevator without exceeding the weight limit. It is guaranteed that the person who is first in the queue can fit in the elevator.

The query result format is in the following example:

Queue table

person_id	person_name	weight	turn
5	George Washington	250	1
3	John Adams	350	2
6	Thomas Jefferson	400	3
2	Will Johnlams	200	4
4	Thomas Jefferson	175	5
1	James Elephant	500	6

Result table

person_name
Thomas Jefferson

Queue table is ordered by turn in the example for simplicity.

In the example George Washington(id 5), John Adams(id 3) and Thomas Jefferson(id 6) will enter the elevator as their weight sum is $250 + 350 + 400 = 1000$.

Thomas Jefferson(id 6) is the last person to fit in the elevator because he has the last turn in these three people.

Solution

01/21/2020 (MySQL):

```
# Write your MySQL query statement below
select person_name
from
(
  select
    person_name, @total_weight := @total_weight + weight as total_weight
  from
```

```

Queue,
(select @total_weight := 0) as tmp
order by turn
) as t
where total_weight <= 1000
order by total_weight desc
limit 1;

```

```

# Write your MySQL query statement below
select q1.person_name
from Queue as q1 join Queue as q2 on q1.turn >= q2.turn
group by q1.turn
having sum(q2.weight) <= 1000
order by sum(q2.weight) desc
limit 1;

```

1211. Queries Quality and Percentage

Description

Table: Queries

Column Name	Type
query_name	varchar
result	varchar
position	int
rating	int

There is no primary key for this table, it may have duplicate rows.
This table contains information collected from some queries on a database.
The position column has a value from 1 to 500.
The rating column has a value from 1 to 5. Query with rating less than 3 is a poor query.

We define query quality as:

The average of the ratio between query rating and its position.

We also define poor query percentage as:

The percentage of all queries with rating less than 3.

Write an SQL query to find each query_name, the quality and poor_query_percentage.

Both quality and poor_query_percentage should be rounded to 2 decimal places.

The query result format is in the following example:

Queries table:

query_name	result	position	rating
Dog	Golden Retriever	1	5
Dog	German Shepherd	2	5
Dog	Mule	200	1
Cat	Shirazi	5	2
Cat	Siamese	3	3
Cat	Sphynx	7	4

Result table:

query_name	quality	poor_query_percentage
Dog	2.50	33.33
Cat	0.66	33.33

Dog queries quality is $((5 / 1) + (5 / 2) + (1 / 200)) / 3 = 2.50$

Dog queries poor_query_percentage is $(1 / 3) * 100 = 33.33$

Cat queries quality equals $((2 / 5) + (3 / 3) + (4 / 7)) / 3 = 0.66$

Cat queries poor_query_percentage is $(1 / 3) * 100 = 33.33$

Solution

01/14/2020:

```
# Write your MySQL query statement below
select q.query_name, round(ifnull(avg(rating / position), 0), 2) as quality,
round(ifnull(cnt / count(q.rating) * 100, 0), 2) as poor_query_percentage
from
  Queries as q
left join
  ( select query_name, count(*) as cnt
    from Queries
    where rating < 3
    group by query_name ) as p
on q.query_name = p.query_name
group by q.query_name;
```

01/14/2020:

```
# Write your MySQL query statement below
select query_name, round(avg(rating / position), 2) as quality,
round(avg(if(rating < 3, 1, 0)) * 100, 2) as poor_query_percentage
from Queries
group by query_name;
```

1212. Team Scores in Football Tournament

Description

Table: Teams

Column Name	Type
team_id	int
team_name	varchar

team_id is the primary key of this table.

Each row of this table represents a single football team.

Table: Matches

Column Name	Type
match_id	int
host_team	int
guest_team	int
host_goals	int
guest_goals	int

+-----+-----+

match_id is the primary key of this table.

Each row is a record of a finished match between two different teams.

Teams host_team and guest_team are represented by their IDs in the teams table (team_id) and they scored host_goals and guest_goals goals respectively.

You would like to compute the scores of all teams after all matches. Points are awarded as follows:

A team receives three points if they win a match (Score strictly more goals than the opponent team).

A team receives one point if they draw a match (Same number of goals as the opponent team).

A team receives no points if they lose a match (Score less goals than the opponent team).

Write an SQL query that selects the team_id, team_name and num_points of each team in the tournament after all described matches. Result table should be ordered by num_points (decreasing order). In case of a tie, order the records by team_id (increasing order).

The query result format is in the following example:

Teams table:

+-----+-----+		
team_id	team_name	
+-----+-----+		
10	Leetcode FC	
20	NewYork FC	
30	Atlanta FC	
40	Chicago FC	
50	Toronto FC	
+-----+-----+		

Matches table:

+-----+-----+-----+-----+-----+					
match_id	host_team	guest_team	host_goals	guest_goals	
+-----+-----+-----+-----+-----+					
1	10	20	3	0	
2	30	10	2	2	
3	10	50	5	1	
4	20	30	1	0	
5	50	30	1	0	
+-----+-----+-----+-----+-----+					

Result table:

+-----+-----+-----+			
team_id	team_name	num_points	
+-----+-----+-----+			
10	Leetcode FC	7	

20	NewYork FC	3	
50	Toronto FC	3	
30	Atlanta FC	1	
40	Chicago FC	0	
+-----+-----+-----+			

Solution

01/19/2020 (MySQL):

```
# Write your MySQL query statement below
select t.team_id, team_name, ifnull(num_points, 0) as num_points
from
  Teams as t
left join
  (
    select team_id, sum(num_points) as num_points
    from
      (
        select
          host_team as team_id,
          sum(case
            when host_goals > guest_goals then 3
            when host_goals = guest_goals then 1
            else 0 end) as num_points
        from Matches
        group by host_team
        union all
        select
          guest_team as team_id,
          sum(case
            when host_goals < guest_goals then 3
            when host_goals = guest_goals then 1
            else 0 end) as num_points
        from Matches
        group by guest_team
      ) as u
    group by team_id
  ) as r
on t.team_id = r.team_id
order by num_points desc, team_id asc;
```

01/19/2020 (MySQL):

```
# Write your MySQL query statement below
select team_id, team_name,
       sum(if(team_id = host_team,
```

```

        case
        when host_goals > guest_goals then 3
        when host_goals = guest_goals then 1
        else 0 end,
        0))
+ sum(if(team_id = guest_team,
        case
        when host_goals < guest_goals then 3
        when host_goals = guest_goals then 1
        else 0 end,
        0)) as num_points
from Teams as t, Matches as m
group by team_id
order by num_points desc, team_id asc;

```

1225. Report Contiguous Dates

Description

Table: Failed

Column Name	Type
fail_date	date

Primary key for this table is fail_date.

Failed table contains the days of failed tasks.

Table: Succeeded

Column Name	Type
success_date	date

Primary key for this table is success_date.

Succeeded table contains the days of succeeded tasks.

A system is running one task every day. Every task is independent of the previous tasks. The tasks can fail or succeed.

Write an SQL query to generate a report of period_state for each continuous interval of days in the period from 2019-01-01 to 2019-12-31.

period_state is 'failed' if tasks in this interval failed or 'succeeded' if tasks in this interval succeeded. Interval of days are retrieved as start_date and end_date.

Order result by start_date.

The query result format is in the following example:

Failed table:

+-----+	
fail_date	
+-----+	
2018-12-28	
2018-12-29	
2019-01-04	
2019-01-05	
+-----+	

Succeeded table:

+-----+	
success_date	
+-----+	
2018-12-30	
2018-12-31	
2019-01-01	
2019-01-02	
2019-01-03	
2019-01-06	
+-----+	

Result table:

+-----+			
period_state	start_date	end_date	
+-----+			
succeeded	2019-01-01	2019-01-03	
failed	2019-01-04	2019-01-05	
succeeded	2019-01-06	2019-01-06	
+-----+			

The report ignored the system state in 2018 as we care about the system in the period 2019-01-01 to 2019-12-31.

From 2019-01-01 to 2019-01-03 all tasks succeeded and the system state was "succeeded".

From 2019-01-04 to 2019-01-05 all tasks failed and system state was "failed".

From 2019-01-06 to 2019-01-06 all tasks succeeded and system state was "succeeded".

Solution

01/19/2020 (MySQL):

```
# Write your MySQL query statement below
select period_state, start_date, end_date
from
(
    select 'failed' as period_state, f1.fail_date as start_date, f2.fail_date as
end_date
    from
        (
            select fail_date
            from Failed
            where fail_date between '2019-01-01' and '2019-12-31' and
date_sub(fail_date, interval 1 day) not in (select * from Failed where fail_date
between '2019-01-01' and '2019-12-31')
        ) as f1,
        (
            select fail_date
            from Failed
            where fail_date between '2019-01-01' and '2019-12-31' and
date_add(fail_date, interval 1 day) not in (select * from Failed where fail_date
between '2019-01-01' and '2019-12-31')
        ) as f2
    where f1.fail_date <= f2.fail_date
    group by f1.fail_date
    union
    select 'succeeded' as period_state, s1.success_date as start_date,
s2.success_date as end_date
    from
        (
            select success_date
            from Succeeded
            where success_date between '2019-01-01' and '2019-12-31' and
date_sub(success_date, interval 1 day) not in (select * from Succeeded where
success_date between '2019-01-01' and '2019-12-31')
        ) as s1,
        (
            select success_date
            from Succeeded
            where success_date between '2019-01-01' and '2019-12-31' and
date_add(success_date, interval 1 day) not in (select * from Succeeded where
success_date between '2019-01-01' and '2019-12-31')
        ) as s2
    where s1.success_date <= s2.success_date
    group by s1.success_date
) as p
order by start_date;
```

1241. Number of Comments per Post

Description

Table: Submissions

Column Name	Type
sub_id	int
parent_id	int

There is no primary key for this table, it may have duplicate rows.

Each row can be a post or comment on the post.

parent_id is null for posts.

parent_id for comments is sub_id for another post in the table.

Write an SQL query to find number of comments per each post.

Result table should contain post_id and its corresponding number_of_comments, and must be sorted by post_id in ascending order.

Submissions may contain duplicate comments. You should count the number of unique comments per post.

Submissions may contain duplicate posts. You should treat them as one post.

The query result format is in the following example:

Submissions table:

sub_id	parent_id
1	Null
2	Null
1	Null
12	Null
3	1
5	2
3	1
4	1
9	1
10	2
6	7

+-----+-----+

Result table:

+-----+-----+	
post_id	number_of_comments
+-----+-----+	
1	3
2	2
12	0
+-----+-----+	

The post with id 1 has three comments in the table with id 3, 4 and 9. The comment with id 3 is repeated in the table, we counted it only once.
The post with id 2 has two comments in the table with id 5 and 10.
The post with id 12 has no comments in the table.
The comment with id 6 is a comment on a deleted post with id 7 so we ignored it.

Solution

01/13/2020:

```
# Write your MySQL query statement below
# select post_id, ifnull(number_of_comments, 0) as number_of_comments
# from (
#     select distinct sub_id as post_id
#     from Submissions
#     where parent_id is null
# ) as s1
# left join
# (
#     select parent_id, count(*) as number_of_comments
#     from (
#         select distinct sub_id, parent_id
#         from Submissions
#     ) as ds
#     where parent_id is not null
#     group by parent_id
# ) as s2
# on s1.post_id = s2.parent_id
# order by post_id;

select t.post_id, count(distinct s.sub_id) as number_of_comments
from (
    select distinct sub_id as post_id
    from Submissions
    where parent_id is null
) as t
left join
```

```
Submissions as s
on t.post_id = s.parent_id
group by t.post_id
order by t.post_id;
```

1251. Average Selling Price

Description

Table: Prices

Column Name	Type
product_id	int
start_date	date
end_date	date
price	int

(product_id, start_date, end_date) is the primary key for this table.

Each row of this table indicates the price of the product_id in the period from start_date to end_date.

For each product_id there will be no two overlapping periods. That means there will be no two intersecting periods for the same product_id.

Table: UnitsSold

Column Name	Type
product_id	int
purchase_date	date
units	int

There is no primary key for this table, it may contain duplicates.

Each row of this table indicates the date, units and product_id of each product sold.

Write an SQL query to find the average selling price for each product.

average_price should be rounded to 2 decimal places.

The query result format is in the following example:

Prices table:

product_id	start_date	end_date	price
1	2019-02-17	2019-02-28	5
1	2019-03-01	2019-03-22	20
2	2019-02-01	2019-02-20	15
2	2019-02-21	2019-03-31	30

UnitsSold table:

product_id	purchase_date	units
1	2019-02-25	100
1	2019-03-01	15
2	2019-02-10	200
2	2019-03-22	30

Result table:

product_id	average_price
1	6.96
2	16.96

Average selling price = Total Price of Product / Number of products sold.

Average selling price for product 1 = $((100 * 5) + (15 * 20)) / 115 = 6.96$

Average selling price for product 2 = $((200 * 15) + (30 * 30)) / 230 = 16.96$

Solution

01/13/2020:

```
# Write your MySQL query statement below
select distinct p.product_id, round(sum(price * units) / sum(units), 2) as
average_price
from Prices as p join UnitsSold as u
on p.product_id = u.product_id and u.purchase_date between p.start_date and
p.end_date
group by p.product_id
order by p.product_id;
```

Table: Friendship

Column Name	Type
user1_id	int
user2_id	int

(user1_id, user2_id) is the primary key for this table.

Each row of this table indicates that there is a friendship relation between user1_id and user2_id.

Table: Likes

Column Name	Type
user_id	int
page_id	int

(user_id, page_id) is the primary key for this table.

Each row of this table indicates that user_id likes page_id.

Write an SQL query to recommend pages to the user with user_id = 1 using the pages that your friends liked. It should not recommend pages you already liked.

Return result table in any order without duplicates.

The query result format is in the following example:

Friendship table:

user1_id	user2_id
1	2
1	3
1	4
2	3
2	4
2	5
6	1

Likes table:

user_id	page_id
1	88
2	23
3	24
4	56
5	11
6	33
2	77
3	77
6	88

Result table:

recommended_page
23
24
56
33
77

User one is friend with users 2, 3, 4 and 6.

Suggested pages are 23 from user 2, 24 from user 3, 56 from user 3 and 33 from user 6.

Page 77 is suggested from both user 2 and user 3.

Page 88 is not suggested because user 1 already likes it.

Solution

01/21/2020 (MySQL):

```
# Write your MySQL query statement below
select distinct page_id as recommended_page
from Likes as l left join Friendship as f on f.user2_id = l.user_id
where f.user1_id = 1 and page_id not in (
    select page_id from Likes where user_id = 1
)
union
select distinct page_id as recommended_page
from Likes as l left join Friendship as f on f.user1_id = l.user_id
where f.user2_id = 1 and page_id not in (
    select page_id from Likes where user_id = 1
);
```

1270. All People Report to the Given Manager

Description

Table: Employees

Column Name	Type
employee_id	int
employee_name	varchar
manager_id	int

employee_id is the primary key for this table.

Each row of this table indicates that the employee with ID employee_id and name employee_name reports his work to his/her direct manager with manager_id

The head of the company is the employee with employee_id = 1.

Write an SQL query to find employee_id of all employees that directly or indirectly report their work to the head of the company.

The indirect relation between managers will not exceed 3 managers as the company is small.

Return result table in any order without duplicates.

The query result format is in the following example:

Employees table:

employee_id	employee_name	manager_id
1	Boss	1
3	Alice	3
2	Bob	1
4	Daniel	2
7	Luis	4
8	Jhon	3
9	Angela	8
77	Robert	1

Result table:

employee_id
2


```

| 77      |
| 4       |
| 7       |
+-----+

```

The head of the company is the employee with employee_id 1.
The employees with employee_id 2 and 77 report their work directly to the head of the company.
The employee with employee_id 4 report his work indirectly to the head of the company 4 --> 2 --> 1.
The employee with employee_id 7 report his work indirectly to the head of the company 7 --> 4 --> 2 --> 1.
The employees with employee_id 3, 8 and 9 don't report their work to head of company directly or indirectly.

Solution

01/19/2020 (MySQL):

```

# Write your MySQL query statement below
select distinct e1.employee_id
from Employees as e1 inner join Employees as e2 inner join Employees as e3 on
e1.manager_id = e2.employee_id and e2.manager_id = e3.employee_id
where e1.employee_id <> 1 and (e1.manager_id = 1 or e2.manager_id = 1 or
e3.manager_id = 1);

```

1280. Students and Examinations

Description

Table: Students

```

+-----+-----+
| Column Name | Type   |
+-----+-----+
| student_id  | int    |
| student_name | varchar|
+-----+-----+

```

student_id is the primary key for this table.

Each row of this table contains the ID and the name of one student in the school.

Table: Subjects

Column Name	Type
subject_name	varchar

subject_name is the primary key for this table.

Each row of this table contains the name of one subject in the school.

Table: Examinations

Column Name	Type
student_id	int
subject_name	varchar

There is no primary key for this table. It may contain duplicates.

Each student from the Students table takes every course from Subjects table.

Each row of this table indicates that a student with ID student_id attended the exam of subject_name.

Write an SQL query to find the number of times each student attended each exam.

Order the result table by student_id and subject_name.

The query result format is in the following example:

Students table:

student_id	student_name
1	Alice
2	Bob
13	John
6	Alex

Subjects table:

subject_name
Math
Physics
Programming

Examinations table:

student_id	subject_name
------------	--------------

1	Math
1	Physics
1	Programming
2	Programming
1	Physics
1	Math
13	Math
13	Programming
13	Physics
2	Math
1	Math

Result table:

student_id	student_name	subject_name	attended_exams
1	Alice	Math	3
1	Alice	Physics	2
1	Alice	Programming	1
2	Bob	Math	1
2	Bob	Physics	0
2	Bob	Programming	1
6	Alex	Math	0
6	Alex	Physics	0
6	Alex	Programming	0
13	John	Math	1
13	John	Physics	1
13	John	Programming	1

The result table should contain all students and all subjects.

Alice attended Math exam 3 times, Physics exam 2 times and Programming exam 1 time.

Bob attended Math exam 1 time, Programming exam 1 time and didn't attend the Physics exam.

Alex didn't attend any exam.

John attended Math exam 1 time, Physics exam 1 time and Programming exam 1 time.

Solution

01/13/2020:

```
# Write your MySQL query statement below
select s.student_id, s.student_name, u.subject_name, count(e.subject_name) as
attended_exams
from
  Students as s join Subjects as u left join Examinations as e
  on
    s.student_id = e.student_id and u.subject_name = e.subject_name
group by s.student_id, u.subject_name
order by s.student_id, u.subject_name;
```

1285. Find the Start and End Number of Continuous Ranges

Description

Table: Logs

Column Name	Type
log_id	int

id is the primary key for this table.

Each row of this table contains the ID in a log Table.

Since some IDs have been removed from Logs. Write an SQL query to find the start and end number of continuous ranges in table Logs.

Order the result table by start_id.

The query result format is in the following example:

Logs table:

log_id
1
2
3
7
8
10

Result table:

|--|--|

start_id	end_id
1	3
7	8
10	10

The result table should contain all ranges in table Logs.
 From 1 to 3 is contained in the table.
 From 4 to 6 is missing in the table
 From 7 to 8 is contained in the table.
 Number 9 is missing in the table.
 Number 10 is contained in the table.

Solution

01/19/2020 (MySQL):

```
# Write your MySQL query statement below
select l1.log_id as start_id, l2.log_id as end_id
from
(
  select log_id
  from Logs
  where log_id - 1 not in (select * from Logs)
) as l1,
(
  select log_id
  from Logs
  where log_id + 1 not in (select * from Logs)
) as l2
where l1.log_id <= l2.log_id
group by l1.log_id;
```

01/19/2020 (MySQL, using variables):

```
# Write your MySQL query statement below
select min(log_id) as start_id, max(log_id) as end_id
from(
  select *, (@id:=@id+1) as id
  from logs, (select @id:= 0) as init
) tmp
group by log_id - id
```

1294. Weather Type in Each Country

Table: Countries

Column Name	Type
country_id	int
country_name	varchar

country_id is the primary key for this table.

Each row of this table contains the ID and the name of one country.

Table: Weather

Column Name	Type
country_id	int
weather_state	varchar
day	date

(country_id, day) is the primary key for this table.

Each row of this table indicates the weather state in a country for one day.

Write an SQL query to find the type of weather in each country for November 2019.

The type of weather is Cold if the average weather_state is less than or equal 15, Hot if the average weather_state is greater than or equal 25 and Warm otherwise.

Return result table in any order.

The query result format is in the following example:

Countries table:

country_id	country_name
2	USA
3	Australia
7	Peru
5	China
8	Morocco
9	Spain

Weather table:		
country_id	weather_state	day
2	15	2019-11-01
2	12	2019-10-28
2	12	2019-10-27
3	-2	2019-11-10
3	0	2019-11-11
3	3	2019-11-12
5	16	2019-11-07
5	18	2019-11-09
5	21	2019-11-23
7	25	2019-11-28
7	22	2019-12-01
7	20	2019-12-02
8	25	2019-11-05
8	27	2019-11-15
8	31	2019-11-25
9	7	2019-10-23
9	3	2019-12-23

Result table:

country_name	weather_type
USA	Cold
Austraila	Cold
Peru	Hot
China	Warm
Morocco	Hot

Average weather_state in USA in November is $(15) / 1 = 15$ so weather type is Cold.

Average weather_state in Austraila in November is $(-2 + 0 + 3) / 3 = 0.333$ so weather type is Cold.

Average weather_state in Peru in November is $(25) / 1 = 25$ so weather type is Hot.

Average weather_state in China in November is $(16 + 18 + 21) / 3 = 18.333$ so weather type is Warm.

Average weather_state in Morocco in November is $(25 + 27 + 31) / 3 = 27.667$ so weather type is Hot.

We know nothing about average weather_state in Spain in November so we don't include it in the result table.

01/14/2020:

```
# Write your MySQL query statement below
select
  country_name,
  case
    when avg(weather_state) <= 15 then 'Cold'
    when avg(weather_state) >= 25 then 'Hot'
    else 'Warm'
  end as weather_type
from
  Weather as w
  left join
  Countries as c
  on c.country_id = w.country_id
where day between '2019-11-01' and '2019-11-30'
group by w.country_id;
```

1303. Find the Team Size

Description

Table: Employee

Column Name	Type
employee_id	int
team_id	int

employee_id is the primary key for this table.

Each row of this table contains the ID of each employee and their respective team.

Write an SQL query to find the team size of each of the employees.

Return result table in any order.

The query result format is in the following example:

Employee Table:

employee_id	team_id
1	8
2	8

	3		8	
	4		7	
	5		9	
	6		9	
+-----+-----+				

Result table:

+-----+-----+				
	employee_id		team_size	
+-----+-----+				
	1		3	
	2		3	
	3		3	
	4		1	
	5		2	
	6		2	
+-----+-----+				

Employees with Id 1,2,3 are part of a team with team_id = 8.

Employees with Id 4 is part of a team with team_id = 7.

Employees with Id 5,6 are part of a team with team_id = 9.

Solution

01/13/2020:

```
# Write your MySQL query statement below
select employee_id, team_size
from Employee as e join (select team_id, count(*) as team_size from employee
group by team_id) as t
on e.team_id = t.team_id;
```

1308. Running Total for Different Genders

Description

Table: Scores

+-----+-----+		
	Column Name	
	Type	
+-----+-----+		
	player_name	
	gender	
	day	
	score_points	
+-----+-----+		

(gender, day) is the primary key for this table.

A competition is held between females team and males team.

Each row of this table indicates that a player_name and with gender has scored score_point in someday.

Gender is 'F' if the player is in females team and 'M' if the player is in males team.

Write an SQL query to find the total score for each gender at each day.

Order the result table by gender and day

The query result format is in the following example:

Scores table:

player_name	gender	day	score_points
Aron	F	2020-01-01	17
Alice	F	2020-01-07	23
Bajrang	M	2020-01-07	7
Khali	M	2019-12-25	11
Slaman	M	2019-12-30	13
Joe	M	2019-12-31	3
Jose	M	2019-12-18	2
Priya	F	2019-12-31	23
Priyanka	F	2019-12-30	17

Result table:

gender	day	total
F	2019-12-30	17
F	2019-12-31	40
F	2020-01-01	57
F	2020-01-07	80
M	2019-12-18	2
M	2019-12-25	13
M	2019-12-30	26
M	2019-12-31	29
M	2020-01-07	36

For females team:

First day is 2019-12-30, Priyanka scored 17 points and the total score for the team is 17.

Second day is 2019-12-31, Priya scored 23 points and the total score for the team is 40.

Third day is 2020-01-01, Aron scored 17 points and the total score for the team is 57.

Fourth day is 2020-01-07, Alice scored 23 points and the total score for the team is 80.

For males team:

First day is 2019-12-18, Jose scored 2 points and the total score for the team is 2.

Second day is 2019-12-25, Khali scored 11 points and the total score for the team is 13.

Third day is 2019-12-30, Slaman scored 13 points and the total score for the team is 26.

Fourth day is 2019-12-31, Joe scored 3 points and the total score for the team is 29.

Fifth day is 2020-01-07, Bajrang scored 7 points and the total score for the team is 36.

Solution

01/19/2020 (MS SQL):

```
/* Write your T-SQL query statement below */
select gender, day, sum(score_points) over(partition by gender order by day) as total
from Scores
order by gender, day;
```

01/19/2020 (MySQL):

```
# Write your MySQL query statement below
select s1.gender, s1.day, sum(s2.score_points) as total
from Scores as s1 join Scores as s2 on s1.gender = s2.gender and s1.day >= s2.day
group by s1.gender, s1.day
order by gender, day;
```

1321. Restaurant Growth

Description

Table: Customer

Column Name	Type
customer_id	int
name	varchar
visited_on	date

amount	int
--------	-----

(customer_id, visited_on) is the primary key for this table.

This table contains data about customer transactions in a restaurant.

visited_on is the date on which the customer with ID (customer_id) have visited the restaurant.

amount is the total paid by a customer.

You are the restaurant owner and you want to analyze a possible expansion (there will be at least one customer every day).

Write an SQL query to compute moving average of how much customer paid in a 7 days window (current day + 6 days before) .

The query result format is in the following example:

Return result table ordered by visited_on.

average_amount should be rounded to 2 decimal places, all dates are in the format ('YYYY-MM-DD').

Customer table:

customer_id	name	visited_on	amount
1	Jhon	2019-01-01	100
2	Daniel	2019-01-02	110
3	Jade	2019-01-03	120
4	Khaled	2019-01-04	130
5	Winston	2019-01-05	110
6	Elvis	2019-01-06	140
7	Anna	2019-01-07	150
8	Maria	2019-01-08	80
9	Jaze	2019-01-09	110
1	Jhon	2019-01-10	130
3	Jade	2019-01-10	150

Result table:

visited_on	amount	average_amount
2019-01-07	860	122.86
2019-01-08	840	120
2019-01-09	840	120
2019-01-10	1000	142.86

+-----+-----+-----+

1st moving average from 2019-01-01 to 2019-01-07 has an average_amount of $(100 + 110 + 120 + 130 + 110 + 140 + 150)/7 = 122.86$
2nd moving average from 2019-01-02 to 2019-01-08 has an average_amount of $(110 + 120 + 130 + 110 + 140 + 150 + 80)/7 = 120$
3rd moving average from 2019-01-03 to 2019-01-09 has an average_amount of $(120 + 130 + 110 + 140 + 150 + 80 + 110)/7 = 120$
4th moving average from 2019-01-04 to 2019-01-10 has an average_amount of $(130 + 110 + 140 + 150 + 80 + 110 + 130 + 150)/7 = 142.86$

Solution

01/21/2020 (MySQL, user defined variables):

```
# Write your MySQL query statement below
select
visited_on, amount, average_amount
from (
  select
    visited_on,
    @cnt := @cnt + 1 as cnt,
    @d7 := @d6,
    @d6 := @d5,
    @d5 := @d4,
    @d4 := @d3,
    @d3 := @d2,
    @d2 := @d1,
    @d1 := amount,
    @total := @d1 + @d2 + @d3 + @d4 + @d5 + @d6 + @d7 as amount,
    round(@total / 7, 2) as average_amount
  from
    (
      select visited_on, sum(amount) as amount
      from Customer
      group by visited_on
    ) as c,
    (
      select
        @cnt := 0,
        @total := 0,
        @d1 := 0,
        @d2 := 0,
        @d3 := 0,
        @d4 := 0,
        @d5 := 0,
        @d6 := 0,
        @d7 := 0
```

```

    ) as t
) as s
where cnt >= 7;

```

1322. Ads Performance

Description

Table: Ads

Column Name	Type
ad_id	int
user_id	int
action	enum

(ad_id, user_id) is the primary key for this table.

Each row of this table contains the ID of an Ad, the ID of a user and the action taken by this user regarding this Ad.

The action column is an ENUM type of ('Clicked', 'Viewed', 'Ignored').

A company is running Ads and wants to calculate the performance of each Ad.

Performance of the Ad is measured using Click-Through Rate (CTR) where:

CTR = 0, if Ad total clicks + Ad total views = 0

CTR = Ad total clicks / (Ad total clicks + Ad total views) * 100, otherwise.

Write an SQL query to find the ctr of each Ad.

Round ctr to 2 decimal points. Order the result table by ctr in descending order and by ad_id in ascending order in case of a tie.

The query result format is in the following example:

Ads table:

ad_id	user_id	action
1	1	Clicked
2	2	Clicked
3	3	Viewed
5	5	Ignored
1	7	Ignored

2	7	Viewed
3	5	Clicked
1	4	Viewed
2	11	Viewed
1	2	Clicked

+-----+

Result table:

+-----+

ad_id	ctr
-------	-----

+-----+

1	66.67
---	-------

3	50.00
---	-------

2	33.33
---	-------

5	0.00
---	------

+-----+

for ad_id = 1, ctr = (2/(2+1)) * 100 = 66.67

for ad_id = 2, ctr = (1/(1+2)) * 100 = 33.33

for ad_id = 3, ctr = (1/(1+1)) * 100 = 50.00

for ad_id = 5, ctr = 0.00, Note that ad_id = 5 has no clicks or views.

Note that we don't care about Ignored Ads.

Result table is ordered by the ctr. in case of a tie we order them by ad_id

Solution

01/18/2020:

```
# Write your MySQL query statement below
select ad_id, round(if(clicks + views = 0, 0, clicks / (clicks + views) * 100),
2) as ctr
from (
  select ad_id, sum(if(action='Clicked', 1, 0)) as clicks,
sum(if(action='Viewed', 1, 0)) as views
  from Ads
  group by ad_id
) as a
order by ctr desc, ad_id asc;
```

1327. List the Products Ordered in a Period

Description

Table: Products

Column Name	Type
-------------	------

product_id	int
product_name	varchar
product_category	varchar

product_id is the primary key for this table.

This table contains data about the company's products.

Table: Orders

Column Name	Type
product_id	int
order_date	date
unit	int

There is no primary key for this table. It may have duplicate rows.

product_id is a foreign key to Products table.

unit is the number of products ordered in order_date.

Write an SQL query to get the names of products with greater than or equal to 100 units ordered in February 2020 and their amount.

Return result table in any order.

The query result format is in the following example:

Products table:

product_id	product_name	product_category
1	Leetcode Solutions	Book
2	Jewels of Stringology	Book
3	HP	Laptop
4	Lenovo	Laptop
5	Leetcode Kit	T-shirt

Orders table:

product_id	order_date	unit
1	2020-02-05	60
1	2020-02-10	70
2	2020-01-18	30
2	2020-02-11	80

3	2020-02-17	2	
3	2020-02-24	3	
4	2020-03-01	20	
4	2020-03-04	30	
4	2020-03-04	60	
5	2020-02-25	50	
5	2020-02-27	50	
5	2020-03-01	50	
+-----+-----+-----+			

Result table:

+-----+-----+		
product_name	unit	
+-----+-----+		
Leetcode Solutions	130	
Leetcode Kit	100	
+-----+-----+		

Products with product_id = 1 is ordered in February a total of $(60 + 70) = 130$.
Products with product_id = 2 is ordered in February a total of 80.
Products with product_id = 3 is ordered in February a total of $(2 + 3) = 5$.
Products with product_id = 4 was not ordered in February 2020.
Products with product_id = 5 is ordered in February a total of $(50 + 50) = 100$.

Solution

01/30/2020 (MySQL):

```
# Write your MySQL query statement below
select product_name, sum(unit) as unit
from Orders as o left join Products as p on o.product_id = p.product_id
where order_date between '2020-02-01' and '2020-02-29'
group by o.product_id
having sum(unit) >= 100;
```

1336. Number of Transactions per Visit

Description

Table: Visits

+-----+-----+		
Column Name	Type	
+-----+-----+		
user_id	int	

```
| visit_date    | date    |
+-----+-----+
```

(user_id, visit_date) is the primary key for this table.

Each row of this table indicates that user_id has visited the bank in visit_date.

Table: Transactions

```
+-----+-----+
| Column Name    | Type    |
+-----+-----+
| user_id        | int     |
| transaction_date | date    |
| amount         | int     |
+-----+-----+
```

There is no primary key for this table, it may contain duplicates.

Each row of this table indicates that user_id has done a transaction of amount in transaction_date.

It is guaranteed that the user has visited the bank in the transaction_date.(i.e The Visits table contains (user_id, transaction_date) in one row)

A bank wants to draw a chart of the number of transactions bank visitors did in one visit to the bank and the corresponding number of visitors who have done this number of transaction in one visit.

Write an SQL query to find how many users visited the bank and didn't do any transactions, how many visited the bank and did one transaction and so on.

The result table will contain two columns:

transactions_count which is the number of transactions done in one visit.

visits_count which is the corresponding number of users who did

transactions_count in one visit to the bank.

transactions_count should take all values from 0 to max(transactions_count) done by one or more users.

Order the result table by transactions_count.

The query result format is in the following example:

Visits table:

```
+-----+-----+
| user_id | visit_date |
+-----+-----+
| 1       | 2020-01-01 |
| 2       | 2020-01-02 |
| 12      | 2020-01-01 |
```

19	2020-01-03	
1	2020-01-02	
2	2020-01-03	
1	2020-01-04	
7	2020-01-11	
9	2020-01-25	
8	2020-01-28	

+-----+

Transactions table:

user_id	transaction_date	amount	
1	2020-01-02	120	
2	2020-01-03	22	
7	2020-01-11	232	
1	2020-01-04	7	
9	2020-01-25	33	
9	2020-01-25	66	
8	2020-01-28	1	
9	2020-01-25	99	

+-----+

Result table:

transactions_count	visits_count	
0	4	
1	5	
2	0	
3	1	

+-----+

* For transactions_count = 0, The visits (1, "2020-01-01"), (2, "2020-01-02"), (12, "2020-01-01") and (19, "2020-01-03") did no transactions so visits_count = 4.

* For transactions_count = 1, The visits (2, "2020-01-03"), (7, "2020-01-11"), (8, "2020-01-28"), (1, "2020-01-02") and (1, "2020-01-04") did one transaction so visits_count = 5.

* For transactions_count = 2, No customers visited the bank and did two transactions so visits_count = 0.

* For transactions_count = 3, The visit (9, "2020-01-25") did three transactions so visits_count = 1.

* For transactions_count >= 4, No customers visited the bank and did more than three transactions so we will stop at transactions_count = 3

The chart drawn for this example is as follows:

01/30/2020 (MySQL):

```
# Write your MySQL query statement below
select (select 0) as transactions_count, count(*) as visits_count
from Visits
where (user_id, visit_date) not in (
    select user_id, transaction_date
    from Transactions
)
union
select s.transactions_count, if(visits_count is null, 0, visits_count) as
visits_count
from (
    select tc as transactions_count
    from (
        select t.user_id, @tc := @tc + 1 as tc
        from Transactions as t, (select @tc := 0) as u
    ) as s
    where tc <= (
        select ifnull(max(transactions_count), 0)
        from (
            select count(*) as transactions_count
            from Transactions
            group by user_id, transaction_date
        ) as t
    )
) as s left join (
    select transactions_count, count(*) as visits_count
    from (
        select count(*) as transactions_count
        from Transactions
        group by user_id, transaction_date
    ) as t
    group by transactions_count
) as t on s.transactions_count = t.transactions_count
order by transactions_count;
```