```
-- Question 21
-- Table: ActorDirector

-- +-------------+---------+
-- | Column Name | Type    |
-- +-------------+---------+
-- | actor_id    | int     |
-- | director_id | int     |
-- | timestamp   | int     |
-- +-------------+---------+
-- timestamp is the primary key column for this table.


-- Write a SQL query for a report that provides the pairs (actor_id,
director_id) where the actor have cooperated with the director at least 3
times.

-- Example:

-- ActorDirector table:
-- +-------------+-------------+-------------+
-- | actor_id    | director_id | timestamp   |
-- +-------------+-------------+-------------+
-- | 1           | 1           | 0           |
-- | 1           | 1           | 1           |
-- | 1           | 1           | 2           |
-- | 1           | 2           | 3           |
-- | 1           | 2           | 4           |
-- | 2           | 1           | 5           |
-- | 2           | 1           | 6           |
-- +-------------+-------------+-------------+

-- Result table:
-- +-------------+-------------+
-- | actor_id    | director_id |
-- +-------------+-------------+
-- | 1           | 1           |
-- +-------------+-------------+
-- The only pair is (1, 1) where they cooperated exactly 3 times.

-- Solution
Select actor_id, director_id
from actordirector
group by actor_id, director_id
having count(*)>=3
```

```
-- Question 13
-- 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:



-- 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
with t1 as(
select ad_id, sum(case when action in ('Clicked') then 1 else 0 end) as
clicked
from ads
group by ad_id
)

, t2 as
(
Select ad_id as ad, sum(case when action in ('Clicked','Viewed') then 1
else 0 end) as total
from ads
group by ad_id
)

Select a.ad_id, coalesce(round((clicked +0.0)/nullif((total
+0.0),0)*100,2),0) as ctr
from
(
select *
from t1 join t2
on t1.ad_id = t2.ad) a
order by ctr desc, ad_id
```

```
-- Question 42
-- 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
select distinct author_id as id
from views
where author_id = viewer_id
order by author_id
```

```
-- Question 39
-- 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
Select d.product_id, round((sum(price*units)+0.00)/(sum(units)+0.00),2)
as average_price
from(
Select *
from prices p
natural join
unitssold u
where u.purchase_date between p.start_date and p.end_date) d
group by d.product_id
```

```
-- Question 5
-- There is a table World

-- +-----------------+------------+------------+--------------+----------
-----+
-- | name            | continent  | area       | population   | gdp
|
-- +-----------------+------------+------------+--------------+----------
-----+
-- | Afghanistan     | Asia       | 652230     | 25500100     | 20343000
|
-- | Albania         | Europe     | 28748      | 2831741      | 12960000
|
-- | Algeria         | Africa     | 2381741    | 37100000     | 188681000
|
-- | Andorra         | Europe     | 468        | 78115        | 3712000
|
-- | Angola          | Africa     | 1246700    | 20609294     | 100990000
|
-- +-----------------+------------+------------+--------------+----------
-----+
-- A country is big if it has an area of bigger than 3 million square km
or a population of more than 25 million.

-- Write a SQL solution to output big countries' name, population and
area.

-- For example, according to the above table, we should output:

-- +--------------+-------------+--------------+
-- | name         | population  | area         |
-- +--------------+-------------+--------------+
-- | Afghanistan  | 25500100    | 652230       |
-- | Algeria      | 37100000    | 2381741      |
-- +--------------+-------------+--------------+


-- Solution
Select name, population, area
from world
where population > 25000000 OR area>3000000
```

```
-- Question 24
-- Table my_numbers contains many numbers in column num including
duplicated ones.
-- Can you write a SQL query to find the biggest number, which only
appears once.

-- +---+
-- |num|
-- +---+
-- | 8 |
-- | 8 |
-- | 3 |
-- | 3 |
-- | 1 |
-- | 4 |
-- | 5 |
-- | 6 |
-- For the sample data above, your query should return the following
result:
-- +---+
-- |num|
-- +---+
-- | 6 |
-- Note:
-- If there is no such number, just output null.

-- Solution
Select max(a.num) as num
from
(
    select num, count(*)
    from my_numbers
    group by num
    having count(*)=1
) a
```

```
-- Question7
-- There is a table courses with columns: student and class

-- Please list out all classes which have more than or equal to 5
students.

-- For example, the table:

-- +---------+------------+
-- | student | class      |
-- +---------+------------+
-- | A       | Math       |
-- | B       | English    |
-- | C       | Math       |
-- | D       | Biology    |
-- | E       | Math       |
-- | F       | Computer   |
-- | G       | Math       |
-- | H       | Math       |
-- | I       | Math       |
-- +---------+------------+

-- Solution
select class
from courses
group by class
having count(distinct student)>=5
```

```
-- Question 14
-- Table: Person

-- +-------------+---------+
-- | Column Name | Type    |
-- +-------------+---------+
-- | PersonId    | int     |
-- | FirstName   | varchar |
-- | LastName    | varchar |
-- +-------------+---------+
-- PersonId is the primary key column for this table.
-- Table: Address

-- +-------------+---------+
-- | Column Name | Type    |
-- +-------------+---------+
-- | AddressId   | int     |
-- | PersonId    | int     |
-- | City        | varchar |
-- | State       | varchar |
-- +-------------+---------+
-- AddressId is the primary key column for this table.


-- Write a SQL query for a report that provides the following information
for each person in the Person table,
-- regardless if there is an address for each of those people:

-- FirstName, LastName, City, State


-- Solution
select FirstName, LastName, City, State
from Person P left join Address A
on P.PersonId = A.PersonId
```

```
-- Question 37
-- Several friends at a cinema ticket office would like to reserve
consecutive available seats.
-- Can you help to query all the consecutive available seats order by the
seat_id using the following cinema table?
-- | seat_id | free |
-- |---------|------|
-- | 1       | 1    |
-- | 2       | 0    |
-- | 3       | 1    |
-- | 4       | 1    |
-- | 5       | 1    |


-- Your query should return the following result for the sample case
above.


-- | seat_id |
-- |---------|
-- | 3       |
-- | 4       |
-- | 5       |
-- Note:
-- The seat_id is an auto increment int, and free is bool ('1' means
free, and '0' means occupied.).
-- Consecutive available seats are more than 2(inclusive) seats
consecutively available.

-- Solution
Select seat_id
from(
select seat_id, free,
lead(free,1) over() as next,
lag(free,1) over() as prev
from cinema) a
where a.free=True and (next = True or prev=True)
order by seat_id
```

```
-- Question 2
-- Table: Sessions

-- +---------------------+---------+
-- | Column Name         | Type    |
-- +---------------------+---------+
-- | session_id          | int     |
-- | duration            | int     |
-- +---------------------+---------+
-- session_id is the primary key for this table.
-- duration is the time in seconds that a user has visited the
application.


-- You want to know how long a user visits your application. You decided
to create bins of "[0-5>", "[5-10>", "[10-15>" and "15 minutes or more"
and count the number of sessions on it.

-- Write an SQL query to report the (bin, total) in any order.

-- The query result format is in the following example.

-- Sessions table:
-- +-------------+---------------+
-- | session_id  | duration      |
-- +-------------+---------------+
-- | 1           | 30            |
-- | 2           | 199           |
-- | 3           | 299           |
-- | 4           | 580           |
-- | 5           | 1000          |
-- +-------------+---------------+

-- Result table:
-- +--------------+--------------+
-- | bin          | total        |
-- +--------------+--------------+
-- | [0-5>        | 3            |
-- | [5-10>       | 1            |
-- | [10-15>      | 0            |
-- | 15 or more   | 1            |
-- +--------------+--------------+

-- For session_id 1, 2 and 3 have a duration greater or equal than 0
minutes and less than 5 minutes.
-- For session_id 4 has a duration greater or equal than 5 minutes and
less than 10 minutes.
-- There are no session with a duration greater or equial than 10 minutes
and less than 15 minutes.
-- For session_id 5 has a duration greater or equal than 15 minutes.

-- Solution 2
(Select '[0-5>' as bin,
 sum(case when duration/60 < 5 then 1 else 0 end) as total from Sessions)
 union
(Select '[5-10>' as bin,
 sum(case when ((duration/60 >= 5) and (duration/60 < 10)) then 1 else 0
end) as total from Sessions)
```

```
 union
(Select '[10-15>' as bin,
 sum(case when ((duration/60 >= 10) and (duration/60 < 15)) then 1 else 0
end) as total from Sessions)
 union
(Select '15 or more' as bin,
 sum(case when duration/60 >= 15 then 1 else 0 end) as total from
Sessions)
```

```
-- Question 8
-- Query the customer_number from the orders table for the customer who
has placed the largest number of orders.

-- It is guaranteed that exactly one customer will have placed more
orders than any other customer.

-- The orders table is defined as follows:

-- | Column            | Type      |
-- |-------------------|-----------|
-- | order_number (PK) | int       |
-- | customer_number   | int       |
-- | order_date        | date      |
-- | required_date     | date      |
-- | shipped_date      | date      |
-- | status            | char(15)  |
-- | comment           | char(200) |
-- Sample Input

-- | order_number | customer_number | order_date | required_date |
shipped_date | status | comment |
-- |--------------|-----------------|------------|---------------|-------
-------|--------|---------|
-- | 1            | 1               | 2017-04-09 | 2017-04-13    | 2017-
04-12   | Closed |         |
-- | 2            | 2               | 2017-04-15 | 2017-04-20    | 2017-
04-18   | Closed |         |
-- | 3            | 3               | 2017-04-16 | 2017-04-25    | 2017-
04-20   | Closed |         |
-- | 4            | 3               | 2017-04-18 | 2017-04-28    | 2017-
04-25   | Closed |         |
-- Sample Output

-- | customer_number |
-- |-----------------|
-- | 3               |
-- Explanation

-- The customer with number '3' has two orders,
-- which is greater than either customer '1' or '2' because each of them
only has one order.
-- So the result is customer_number '3'.


-- Solution
-- Ranking them according to the number of orders to have same rank for
-- customers with same number of orders
With t1 as
(
  Select customer_number,
  Rank() over(order by count(customer_number) desc) as rk
  from orders
  group by customer_number
)

Select t1.customer_number
from t1
```

```
where t1.rk=1
```

```
-- Question 13
-- Suppose that a website contains two tables,
-- the Customers table and the Orders table. Write a SQL query to find
all customers who never order anything.

-- Table: Customers.

-- +----+-------+
-- | Id | Name  |
-- +----+-------+
-- | 1  | Joe   |
-- | 2  | Henry |
-- | 3  | Sam   |
-- | 4  | Max   |
-- +----+-------+
-- Table: Orders.

-- +----+------------+
-- | Id | CustomerId |
-- +----+------------+
-- | 1  | 3          |
-- | 2  | 1          |
-- +----+------------+
-- Using the above tables as example, return the following:

-- +-----------+
-- | Customers |
-- +-----------+
-- | Henry     |
-- | Max       |
-- +-----------+


-- Solution
Select Name as Customers
from Customers
where id != All(select c.id
                from Customers c, Orders o
                where c.id = o.Customerid)
```

```
-- Question 32
-- Write a SQL query to delete all duplicate email entries in a table
named Person, keeping only unique emails based on its smallest Id.

-- +----+------------------+
-- | Id | Email            |
-- +----+------------------+
-- | 1  | john@example.com |
-- | 2  | bob@example.com  |
-- | 3  | john@example.com |
-- +----+------------------+
-- Id is the primary key column for this table.
-- For example, after running your query, the above Person table should
have the following rows:

-- +----+------------------+
-- | Id | Email            |
-- +----+------------------+
-- | 1  | john@example.com |
-- | 2  | bob@example.com  |
-- +----+------------------+

-- Solution
With t1 as
(
 Select *,
    row_number() over(partition by email order by id) as rk
    from person
)
Delete from person
where id in (Select t1.id from t1 where t1.rk>1)
```

```
-- Question 11
-- Write a SQL query to find all duplicate emails in a table named
Person.

-- +----+---------+
-- | Id | Email   |
-- +----+---------+
-- | 1  | a@b.com |
-- | 2  | c@d.com |
-- | 3  | a@b.com |
-- +----+---------+
-- For example, your query should return the following for the above
table:

-- +---------+
-- | Email   |
-- +---------+
-- | a@b.com |
-- +---------+


-- Solution
Select Email
from
(Select Email, count(Email)
from person
group by Email
having count(Email)>1) a
```

```
-- Question 4
-- Select all employee's name and bonus whose bonus is < 1000.

-- Table:Employee

-- +-------+--------+-----------+--------+
-- | empId |  name  | supervisor| salary |
-- +-------+--------+-----------+--------+
-- |   1   | John   |  3        | 1000   |
-- |   2   | Dan    |  3        | 2000   |
-- |   3   | Brad   |  null     | 4000   |
-- |   4   | Thomas |  3        | 4000   |
-- +-------+--------+-----------+--------+
-- empId is the primary key column for this table.
-- Table: Bonus

-- +-------+-------+
-- | empId | bonus |
-- +-------+-------+
-- | 2     | 500   |
-- | 4     | 2000  |
-- +-------+-------+
-- empId is the primary key column for this table.
-- Example ouput:

-- +-------+-------+
-- | name  | bonus |
-- +-------+-------+
-- | John  | null  |
-- | Dan   | 500   |
-- | Brad  | null  |
-- +-------+-------+


-- Solution
Select E.name, B.bonus
From Employee E left join Bonus B
on E.empId = B.empId
where B.bonus< 1000 or B.Bonus IS NULL
```

```
-- Question 15
-- The Employee table holds all employees including their managers.
-- Every employee has an Id, and there is also a column for the manager
Id.

-- +----+-------+--------+-----------+
-- | Id | Name  | Salary | ManagerId |
-- +----+-------+--------+-----------+
-- | 1  | Joe   | 70000  | 3         |
-- | 2  | Henry | 80000  | 4         |
-- | 3  | Sam   | 60000  | NULL      |
-- | 4  | Max   | 90000  | NULL      |
-- +----+-------+--------+-----------+
-- Given the Employee table, write a SQL query that finds out employees
who earn more than their managers.
-- For the above table, Joe is the only employee who earns more than his
manager.

-- +----------+
-- | Employee |
-- +----------+
-- | Joe      |
-- +----------+

-- Solution
select a.Name as Employee
from employee a, employee b
where a.salary>b.salary and a.managerid=b.id
```

```
-- Question 10
-- Given a table customer holding customers information and the referee.

-- +------+------+-----------+
-- | id   | name | referee_id|
-- +------+------+-----------+
-- |    1 | Will |      NULL |
-- |    2 | Jane |      NULL |
-- |    3 | Alex |         2 |
-- |    4 | Bill |      NULL |
-- |    5 | Zack |         1 |
-- |    6 | Mark |         2 |
-- +------+------+-----------+
-- Write a query to return the list of customers NOT referred by the
person with id '2'.

-- For the sample data above, the result is:

-- +------+
-- | name |
-- +------+
-- | Will |
-- | Jane |
-- | Bill |
-- | Zack |
-- +------+

-- Solution
Select name
from customer
where referee_id != 2
or referee_id is NULL
```

```
-- Question 47
-- 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
Select employee_id, b.team_size
from employee e
join
(
Select team_id, count(team_id) as team_size
from employee
group by team_id) b
on e.team_id = b.team_id
```

```
-- Question 49
-- In social network like Facebook or Twitter, people send friend
requests and accept others' requests as well. Now given two tables as
below:


-- Table: friend_request
-- | sender_id | send_to_id |request_date|
-- |-----------|------------|------------|
-- | 1         | 2          | 2016_06-01 |
-- | 1         | 3          | 2016_06-01 |
-- | 1         | 4          | 2016_06-01 |
-- | 2         | 3          | 2016_06-02 |
-- | 3         | 4          | 2016-06-09 |


-- Table: request_accepted
-- | requester_id | accepter_id |accept_date |
-- |--------------|-------------|------------|
-- | 1            | 2           | 2016_06-03 |
-- | 1            | 3           | 2016-06-08 |
-- | 2            | 3           | 2016-06-08 |
-- | 3            | 4           | 2016-06-09 |
-- | 3            | 4           | 2016-06-10 |


-- Write a query to find the overall acceptance rate of requests rounded
to 2 decimals, which is the number of acceptance divide the number of
requests.


-- For the sample data above, your query should return the following
result.


-- |accept_rate|
-- |-----------|
-- |       0.80|


-- Note:
-- The accepted requests are not necessarily from the table
friend_request. In this case, you just need to simply count the total
accepted requests (no matter whether they are in the original requests),
and divide it by the number of requests to get the acceptance rate.
-- It is possible that a sender sends multiple requests to the same
receiver, and a request could be accepted more than once. In this case,
the 'duplicated' requests or acceptances are only counted once.
-- If there is no requests at all, you should return 0.00 as the
accept_rate.


-- Explanation: There are 4 unique accepted requests, and there are 5
requests in total.
-- So the rate is 0.80.

-- Solution
with t1 as
```

```
(
      select distinct sender_id, send_to_id
      from friend_request
), t2 as
(
      select distinct requester_id, accepter_id
      from request_accepted
)

Select
ifnull((
        select distinct
        round((select count(*) from t2) / ( select count(*) from t1),2)
from t1,t2
    ),0) 'accept_rate'
```

```
-- Question 115
-- Write an SQL query to report the distinct titles of the kid-friendly
movies streamed in June 2020.

-- Return the result table in any order.

-- The query result format is in the following example.



-- TVProgram table:
-- +--------------------+--------------+-------------+
-- | program_date       | content_id   | channel     |
-- +--------------------+--------------+-------------+
-- | 2020-06-10 08:00   | 1            | LC-Channel  |
-- | 2020-05-11 12:00   | 2            | LC-Channel  |
-- | 2020-05-12 12:00   | 3            | LC-Channel  |
-- | 2020-05-13 14:00   | 4            | Disney Ch   |
-- | 2020-06-18 14:00   | 4            | Disney Ch   |
-- | 2020-07-15 16:00   | 5            | Disney Ch   |
-- +--------------------+--------------+-------------+

-- Content table:
-- +------------+----------------+---------------+---------------+
-- | content_id | title          | Kids_content  | content_type  |
-- +------------+----------------+---------------+---------------+
-- | 1          | Leetcode Movie | N             | Movies        |
-- | 2          | Alg. for Kids  | Y             | Series        |
-- | 3          | Database Sols  | N             | Series        |
-- | 4          | Aladdin        | Y             | Movies        |
-- | 5          | Cinderella     | Y             | Movies        |
-- +------------+----------------+---------------+---------------+

-- Result table:
-- +--------------+
-- | title        |
-- +--------------+
-- | Aladdin      |
-- +--------------+
-- "Leetcode Movie" is not a content for kids.
-- "Alg. for Kids" is not a movie.
-- "Database Sols" is not a movie
-- "Alladin" is a movie, content for kids and was streamed in June 2020.
-- "Cinderella" was not streamed in June 2020.

-- Solution
select distinct title
from
(select content_id, title
from content
where kids_content = 'Y' and content_type = 'Movies') a
join
tvprogram using (content_id)
where month(program_date) = 6
```

```
-- Question 3
-- Table: Activity

-- +--------------+---------+
-- | Column Name  | Type    |
-- +--------------+---------+
-- | player_id    | int     |
-- | device_id    | int     |
-- | event_date   | date    |
-- | games_played | int     |
-- +--------------+---------+
-- (player_id, event_date) is the primary key of this table.
-- This table shows the activity of players of some game.
-- Each row is a record of a player who logged in and played a number of
games (possibly 0) before logging out on some day using some device.


-- Write an SQL query that reports the first login date for each player.

-- The query result format is in the following example:

-- Activity table:
-- +-----------+-----------+------------+--------------+
-- | player_id | device_id | event_date | games_played |
-- +-----------+-----------+------------+--------------+
-- | 1         | 2         | 2016-03-01 | 5            |
-- | 1         | 2         | 2016-05-02 | 6            |
-- | 2         | 3         | 2017-06-25 | 1            |
-- | 3         | 1         | 2016-03-02 | 0            |
-- | 3         | 4         | 2018-07-03 | 5            |
-- +-----------+-----------+------------+--------------+

-- Result table:
-- +-----------+-------------+
-- | player_id | first_login |
-- +-----------+-------------+
-- | 1         | 2016-03-01  |
-- | 2         | 2017-06-25  |
-- | 3         | 2016-03-02  |
-- +-----------+-------------+

-- Solution
Select player_id, min(event_date) as first_login
from Activity
Group by player_id
```

```
-- Question 9
-- Table: Activity

-- +--------------+---------+
-- | Column Name  | Type    |
-- +--------------+---------+
-- | player_id    | int     |
-- | device_id    | int     |
-- | event_date   | date    |
-- | games_played | int     |
-- +--------------+---------+
-- (player_id, event_date) is the primary key of this table.
-- This table shows the activity of players of some game.
-- Each row is a record of a player who logged in and played a number of
games (possibly 0) before logging out on some day using some device.


-- Write a SQL query that reports the device that is first logged in for
each player.

-- The query result format is in the following example:

-- Activity table:
-- +-----------+-----------+------------+--------------+
-- | player_id | device_id | event_date | games_played |
-- +-----------+-----------+------------+--------------+
-- | 1         | 2         | 2016-03-01 | 5            |
-- | 1         | 2         | 2016-05-02 | 6            |
-- | 2         | 3         | 2017-06-25 | 1            |
-- | 3         | 1         | 2016-03-02 | 0            |
-- | 3         | 4         | 2018-07-03 | 5            |
-- +-----------+-----------+------------+--------------+

-- Result table:
-- +-----------+-----------+
-- | player_id | device_id |
-- +-----------+-----------+
-- | 1         | 2         |
-- | 2         | 3         |
-- | 3         | 1         |
-- +-----------+-----------+


-- Solution
With table1 as
(
   Select player_id, device_id,
   Rank() OVER(partition by player_id
               order by event_date) as rk
   From Activity
)
Select t.player_id, t.device_id
from table1 as t
where t.rk=1
```

```
-- Question 116
-- Table Activities:

-- +-------------+---------+
-- | Column Name | Type    |
-- +-------------+---------+
-- | sell_date   | date    |
-- | product     | varchar |
-- +-------------+---------+
-- There is no primary key for this table, it may contains duplicates.
-- Each row of this table contains the product name and the date it was
sold in a market.


-- Write an SQL query to find for each date, the number of distinct
products sold and their names.

-- The sold-products names for each date should be sorted
lexicographically.

-- Return the result table ordered by sell_date.

-- The query result format is in the following example.

-- Activities table:
-- +------------+-------------+
-- | sell_date  | product     |
-- +------------+-------------+
-- | 2020-05-30 | Headphone   |
-- | 2020-06-01 | Pencil      |
-- | 2020-06-02 | Mask        |
-- | 2020-05-30 | Basketball  |
-- | 2020-06-01 | Bible       |
-- | 2020-06-02 | Mask        |
-- | 2020-05-30 | T-Shirt     |
-- +------------+-------------+

-- Result table:
-- +------------+----------+------------------------------+
-- | sell_date  | num_sold | products                     |
-- +------------+----------+------------------------------+
-- | 2020-05-30 | 3        | Basketball,Headphone,T-shirt |
-- | 2020-06-01 | 2        | Bible,Pencil                 |
-- | 2020-06-02 | 1        | Mask                         |
-- +------------+----------+------------------------------+
-- For 2020-05-30, Sold items were (Headphone, Basketball, T-shirt), we
sort them lexicographically and separate them by comma.
-- For 2020-06-01, Sold items were (Pencil, Bible), we sort them
lexicographically and separate them by comma.
-- For 2020-06-02, Sold item is (Mask), we just return it.

-- Solution
select sell_date, count(distinct product) as num_sold,
group_concat(distinct product) as products
from activities
group by 1
order by 1
```

```
-- Question 38
-- 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
```

```
Select
Round(avg(case when order_date=customer_pref_delivery_date then 1 else 0
end)*100,2) as immediate_percentage
from delivery
```

```
-- Question 45
-- 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
Select a.product_name, a.unit
from
(select p.product_name, sum(unit) as unit
from orders o
join products p
on o.product_id = p.product_id
where month(order_date)=2 and year(order_date) = 2020
group by o.product_id) a
where a.unit>=100
```

```
-- Question 6
-- X city opened a new cinema, many people would like to go to this
cinema.
-- The cinema also gives out a poster indicating the movies’ ratings and
descriptions.
-- Please write a SQL query to output movies with an odd numbered ID and
a description that is not 'boring'.
-- Order the result by rating.



-- For example, table cinema:

-- +---------+-----------+--------------+-----------+
-- |   id    | movie     |  description |  rating   |
-- +---------+-----------+--------------+-----------+
-- |   1     | War       |   great 3D   |   8.9     |
-- |   2     | Science   |   fiction    |   8.5     |
-- |   3     | irish     |   boring     |   6.2     |
-- |   4     | Ice song  |   Fantacy    |   8.6     |
-- |   5     | House card|   Interesting|   9.1     |
-- +---------+-----------+--------------+-----------+
-- For the example above, the output should be:
-- +---------+-----------+--------------+-----------+
-- |   id    | movie     |  description |  rating   |
-- +---------+-----------+--------------+-----------+
-- |   5     | House card|   Interesting|   9.1     |
-- |   1     | War       |   great 3D   |   8.9     |
-- +---------+-----------+--------------+-----------+


-- Solution
Select *
from cinema
where id%2=1 and description not in ('boring')
order by rating desc
```

```
-- Question 31
-- 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
Select a.sub_id as post_id, coalesce(b.number_of_comments,0) as
number_of_comments
from(
select distinct sub_id from submissions where parent_id is null) a
left join(
select parent_id, count(distinct(sub_id)) as number_of_comments
from submissions
group by parent_id
having parent_id = any(select sub_id from submissions where parent_id is
null)) b
on a.sub_id = b.parent_id
order by post_id
```

```
-- Question 30
-- Table: Sales

-- +-------------+-------+
-- | Column Name | Type  |
-- +-------------+-------+
-- | sale_id     | int   |
-- | product_id  | int   |
-- | year        | int   |
-- | quantity    | int   |
-- | price       | int   |
-- +-------------+-------+
-- (sale_id, year) is the primary key of this table.
-- product_id is a foreign key to Product table.
-- Note that the price is per unit.
-- Table: Product

-- +--------------+---------+
-- | Column Name  | Type    |
-- +--------------+---------+
-- | product_id   | int     |
-- | product_name | varchar |
-- +--------------+---------+
-- product_id is the primary key of this table.


-- Write an SQL query that reports all product names of the products in
the Sales table along with their selling year and price.

-- For example:

-- Sales table:
-- +---------+------------+------+----------+-------+
-- | sale_id | product_id | year | quantity | price |
-- +---------+------------+------+----------+-------+
-- | 1       | 100        | 2008 | 10       | 5000  |
-- | 2       | 100        | 2009 | 12       | 5000  |
-- | 7       | 200        | 2011 | 15       | 9000  |
-- +---------+------------+------+----------+-------+

-- Product table:
-- +------------+--------------+
-- | product_id | product_name |
-- +------------+--------------+
-- | 100        | Nokia        |
-- | 200        | Apple        |
-- | 300        | Samsung      |
-- +------------+--------------+

-- Result table:
-- +--------------+-------+-------+
-- | product_name | year  | price |
-- +--------------+-------+-------+
-- | Nokia        | 2008  | 5000  |
-- | Nokia        | 2009  | 5000  |
-- | Apple        | 2011  | 9000  |
-- +--------------+-------+-------+
```

```sql
-- Solution
Select a.product_name, b.year, b.price
from product as a
join
sales as b
on a.product_id = b.product_id
```

```
-- Question 29
-- Table: Sales

-- +-------------+-------+
-- | Column Name | Type  |
-- +-------------+-------+
-- | sale_id     | int   |
-- | product_id  | int   |
-- | year        | int   |
-- | quantity    | int   |
-- | price       | int   |
-- +-------------+-------+
-- sale_id is the primary key of this table.
-- product_id is a foreign key to Product table.
-- Note that the price is per unit.
-- Table: Product

-- +--------------+---------+
-- | Column Name  | Type    |
-- +--------------+---------+
-- | product_id   | int     |
-- | product_name | varchar |
-- +--------------+---------+
-- product_id is the primary key of this table.


-- Write an SQL query that reports the total quantity sold for every
product id.

-- The query result format is in the following example:

-- Sales table:
-- +---------+------------+------+----------+-------+
-- | sale_id | product_id | year | quantity | price |
-- +---------+------------+------+----------+-------+
-- | 1       | 100        | 2008 | 10       | 5000  |
-- | 2       | 100        | 2009 | 12       | 5000  |
-- | 7       | 200        | 2011 | 15       | 9000  |
-- +---------+------------+------+----------+-------+

-- Product table:
-- +------------+--------------+
-- | product_id | product_name |
-- +------------+--------------+
-- | 100        | Nokia        |
-- | 200        | Apple        |
-- | 300        | Samsung      |
-- +------------+--------------+

-- Result table:
-- +--------------+----------------+
-- | product_id   | total_quantity |
-- +--------------+----------------+
-- | 100          | 22             |
-- | 200          | 15             |
-- +--------------+----------------+

-- Solution
```

```
Select a.product_id, sum(a.quantity) as total_quantity
from sales a
join
product b
on a.product_id = b.product_id
group by a.product_id
```

```
-- Question 26
-- 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 average experience years of all
the employees for each project, rounded to 2 digits.

-- 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   | 1                |
-- | 4           | Doe    | 2                |
-- +-------------+--------+------------------+

-- Result table:
-- +-------------+---------------+
-- | project_id  | average_years |
-- +-------------+---------------+
-- | 1           | 2.00          |
-- | 2           | 2.50          |
-- +-------------+---------------+
-- The average experience years for the first project is (3 + 2 + 1) / 3
= 2.00 and for the second project is (3 + 2) / 2 = 2.50
```

```sql
-- Solution
Select a.project_id,
round(sum(b.experience_years)/count(b.employee_id),2) as average_years
from project as a
join
employee as b
on a.employee_id=b.employee_id
group by a.project_id
```

```
-- Question 28
-- 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 all the projects that have the most
employees.

-- 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   | 1                |
-- | 4           | Doe    | 2                |
-- +-------------+--------+------------------+

-- Result table:
-- +-------------+
-- | project_id  |
-- +-------------+
-- | 1           |
-- +-------------+
-- The first project has 3 employees while the second one has 2.

-- Solution
```

```
select a.project_id
from(
select project_id,
rank() over(order by count(employee_id) desc) as rk
from project
group by project_id) a
where a.rk = 1
```

```
-- Question 41
-- 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
Select query_name, round(sum(rating/position)/count(*),2) as quality,
round(avg(case when rating<3 then 1 else 0 end)*100,2) as
poor_query_percentage
from queries
group by query_name
```

```
-- Question 44
-- 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
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
```

```
-- Question 48
-- Table: Employees

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | id            | int     |
-- | name          | varchar |
-- +---------------+---------+
-- id is the primary key for this table.
-- Each row of this table contains the id and the name of an employee in
a company.


-- Table: EmployeeUNI

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | id            | int     |
-- | unique_id     | int     |
-- +---------------+---------+
-- (id, unique_id) is the primary key for this table.
-- Each row of this table contains the id and the corresponding unique id
of an employee in the company.


-- Write an SQL query to show the unique ID of each user, If a user
doesn't have a unique ID replace just show null.

-- Return the result table in any order.

-- The query result format is in the following example:

-- Employees table:
-- +----+----------+
-- | id | name     |
-- +----+----------+
-- | 1  | Alice    |
-- | 7  | Bob      |
-- | 11 | Meir     |
-- | 90 | Winston  |
-- | 3  | Jonathan |
-- +----+----------+

-- EmployeeUNI table:
-- +----+-----------+
-- | id | unique_id |
-- +----+-----------+
-- | 3  | 1         |
-- | 11 | 2         |
-- | 90 | 3         |
-- +----+-----------+

-- EmployeeUNI table:
-- +-----------+----------+
-- | unique_id | name     |
-- +-----------+----------+
```

```
-- | null      | Alice    |
-- | null      | Bob      |
-- | 2         | Meir     |
-- | 3         | Winston  |
-- | 1         | Jonathan |
-- +-----------+----------+

-- Alice and Bob don't have a unique ID, We will show null instead.
-- The unique ID of Meir is 2.
-- The unique ID of Winston is 3.
-- The unique ID of Jonathan is 1.

-- Solution
select unique_id, name
from employees e
left join
employeeuni u
on e.id = u.id
order by e.id
```

```
-- Question 43
-- 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
Select extra as report_reason, count(distinct post_id) as report_count
from actions
where action_date = DATE_SUB("2019-07-5", INTERVAL 1 DAY) and
action='report'
```

```
group by extra
```

```
-- Question 12
-- Given a Weather table, write a SQL query to find all dates' Ids with
higher temperature compared to its previous (yesterday's) dates.

-- +---------+------------------+------------------+
-- | Id(INT) | RecordDate(DATE) | Temperature(INT) |
-- +---------+------------------+------------------+
-- |       1 |       2015-01-01 |               10 |
-- |       2 |       2015-01-02 |               25 |
-- |       3 |       2015-01-03 |               20 |
-- |       4 |       2015-01-04 |               30 |
-- +---------+------------------+------------------+
-- For example, return the following Ids for the above Weather table:

-- +----+
-- | Id |
-- +----+
-- |  2 |
-- |  4 |
-- +----+

-- Solution
select a.Id
from weather a, weather b
where a.Temperature>b.Temperature and
datediff(a.recorddate,b.recorddate)=1
```

```
-- Question 27
-- 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
Select a.seller_id
from
(select seller_id,
rank() over(order by sum(price) desc) as rk
from sales
group by seller_id) a
where a.rk=1
```

```
-- Question 33
-- 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
Select distinct a.buyer_id
from sales a join
product b
on a.product_id = b.product_id
where a.buyer_id in
(Select a.buyer_id from sales a join product b on a.product_id =
b.product_id where b.product_name = 'S8')
and
a.buyer_id not in (Select a.buyer_id from sales a join product b on
a.product_id = b.product_id where b.product_name = 'iPhone')
```

```
-- Question 34
-- 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
select distinct a.product_id, product_name from sales a join product b on
a.product_id = b.product_id where a.product_id
in
(select product_id from sales where sale_date >= '2019-01-01' and
sale_date <= '2019-03-31')
and
a.product_id not in
(select product_id from sales where sale_date > '2019-03-31' or sale_date
< '2019-01-01')
```

```
-- Question 12
-- Description

-- Given three tables: salesperson, company, orders.
-- Output all the names in the table salesperson, who didn’t have sales
to company 'RED'.

-- Example
-- Input

-- Table: salesperson

-- +----------+------+--------+-----------------+-----------+
-- | sales_id | name | salary | commission_rate | hire_date |
-- +----------+------+--------+-----------------+-----------+
-- |   1      | John | 100000 |     6           | 4/1/2006  |
-- |   2      | Amy  | 120000 |     5           | 5/1/2010  |
-- |   3      | Mark | 65000  |     12          | 12/25/2008|
-- |   4      | Pam  | 25000  |     25          | 1/1/2005  |
-- |   5      | Alex | 50000  |     10          | 2/3/2007  |
-- +----------+------+--------+-----------------+-----------+
-- The table salesperson holds the salesperson information. Every
salesperson has a sales_id and a name.
-- Table: company

-- +---------+--------+------------+
-- | com_id  |  name  |    city    |
-- +---------+--------+------------+
-- |   1     |  RED   |   Boston   |
-- |   2     | ORANGE |   New York |
-- |   3     | YELLOW |   Boston   |
-- |   4     | GREEN  |   Austin   |
-- +---------+--------+------------+
-- The table company holds the company information. Every company has a
com_id and a name.
-- Table: orders

-- +----------+------------+---------+----------+--------+
-- | order_id | order_date | com_id  | sales_id | amount |
-- +----------+------------+---------+----------+--------+
-- | 1        |   1/1/2014 |    3    |    4     | 100000 |
-- | 2        |   2/1/2014 |    4    |    5     | 5000   |
-- | 3        |   3/1/2014 |    1    |    1     | 50000  |
-- | 4        |   4/1/2014 |    1    |    4     | 25000  |
-- +----------+----------+---------+----------+--------+
-- The table orders holds the sales record information, salesperson and
customer company are represented by sales_id and com_id.
-- output

-- +------+
-- | name |
-- +------+
-- | Amy  |
-- | Mark |
-- | Alex |
-- +------+
-- Explanation
```

```
-- According to order '3' and '4' in table orders, it is easy to tell
only salesperson 'John' and 'Pam' have sales to company 'RED',
-- so we need to output all the other names in the table salesperson.

-- Solution
# Takes higher time
# Select distinct a.name
# from(
# select s.sales_id as sales, name
# from salesperson s left join orders o
# on s.sales_id = o.sales_id) a
# where a.sales != all(select distinct sales_id from orders o join
company c on o.com_id = c.com_id where o.com_id = any (select com_id from
company where name = 'RED'))

# Faster solution
SELECT name
FROM salesperson
WHERE sales_id NOT IN (SELECT DISTINCT sales_id
FROM orders
WHERE com_id = (SELECT com_id
FROM company
WHERE name = 'RED')) ;
```

```
-- Question 15
-- Write a SQL query to get the second highest salary from the Employee
table.

-- +----+--------+
-- | Id | Salary |
-- +----+--------+
-- | 1  | 100    |
-- | 2  | 200    |
-- | 3  | 300    |
-- +----+--------+
-- For example, given the above Employee table, the query should return
200 as the second highest salary.
-- If there is no second highest salary, then the query should return
null.

-- +---------------------+
-- | SecondHighestSalary |
-- +---------------------+
-- | 200                 |
-- +---------------------+


-- Solution
select max(salary) as SecondHighestSalary
from employee
where salary ! = (Select max(salary)
                  from employee)
```

```
-- Question 25
-- Table point holds the x coordinate of some points on x-axis in a
plane, which are all integers.


-- Write a query to find the shortest distance between two points in
these points.


-- | x   |
-- |-----|
-- | -1  |
-- | 0   |
-- | 2   |


-- The shortest distance is '1' obviously, which is from point '-1' to
'0'. So the output is as below:


-- | shortest|
-- |---------|
-- | 1       |


-- Note: Every point is unique, which means there is no duplicates in
table point

-- Solution
select min(abs(abs(a.x)-abs(a.next_closest))) as shortest
from(
select *,
lead(x) over(order by x) as next_closest
from point) a
```

```
-- Question 23
-- 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
Select a.student_id as student_id, a.student_name as student_name,
a.subject_name as subject_name, coalesce(attended_exams,0) as
attended_exams
from(
select *
from students
cross join subjects
group by student_id, student_name, subject_name) a
left join
```

```
(Select e.student_id, student_name, subject_name, count(*) as
attended_exams
from examinations e join students s
on e.student_id = s.student_id
group by e.student_id, student_name, subject_name) b
on a.student_id = b.student_id and a.subject_name =b.subject_name
order by a.student_id asc, a.subject_name asc
```

```
-- Question 36
-- Table: Departments

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | id            | int     |
-- | name          | varchar |
-- +---------------+---------+
-- id is the primary key of this table.
-- The table has information about the id of each department of a
university.


-- Table: Students

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | id            | int     |
-- | name          | varchar |
-- | department_id | int     |
-- +---------------+---------+
-- id is the primary key of this table.
-- The table has information about the id of each student at a university
and the id of the department he/she studies at.


-- Write an SQL query to find the id and the name of all students who are
enrolled in departments that no longer exists.

-- Return the result table in any order.

-- The query result format is in the following example:

-- Departments table:
-- +------+--------------------------+
-- | id   | name                     |
-- +------+--------------------------+
-- | 1    | Electrical Engineering   |
-- | 7    | Computer Engineering     |
-- | 13   | Bussiness Administration |
-- +------+--------------------------+

-- Students table:
-- +------+----------+---------------+
-- | id   | name     | department_id |
-- +------+----------+---------------+
-- | 23   | Alice    | 1             |
-- | 1    | Bob      | 7             |
-- | 5    | Jennifer | 13            |
-- | 2    | John     | 14            |
-- | 4    | Jasmine  | 77            |
-- | 3    | Steve    | 74            |
-- | 6    | Luis     | 1             |
-- | 8    | Jonathan | 7             |
-- | 7    | Daiana   | 33            |
-- | 11   | Madelynn | 1             |
```

```
-- +------+----------+---------------+

-- Result table:
-- +------+----------+
-- | id   | name     |
-- +------+----------+
-- | 2    | John     |
-- | 7    | Daiana   |
-- | 4    | Jasmine  |
-- | 3    | Steve    |
-- +------+----------+

-- John, Daiana, Steve and Jasmine are enrolled in departments 14, 33, 74
and 77 respectively.
-- department 14, 33, 74 and 77 doesn't exist in the Departments table.

-- Solution
Select s.id, s.name
from students s left join
departments d
on s.department_id = d.id
where d.name is null
```

```
-- Question 22
-- Given a table salary, such as the one below, that has m=male and
f=female values.
-- Swap all f and m values (i.e., change all f values to m and vice
versa) with
-- a single update statement and no intermediate temp table.

-- Note that you must write a single update statement, DO NOT write any
select statement for this problem.


-- Example:

-- | id | name | sex | salary |
-- |----|------|-----|--------|
-- | 1  | A    | m   | 2500   |
-- | 2  | B    | f   | 1500   |
-- | 3  | C    | m   | 5500   |
-- | 4  | D    | f   | 500    |
-- After running your update statement, the above salary table should
have the following rows:
-- | id | name | sex | salary |
-- |----|------|-----|--------|
-- | 1  | A    | f   | 2500   |
-- | 2  | B    | m   | 1500   |
-- | 3  | C    | f   | 5500   |
-- | 4  | D    | m   | 500    |

-- Solution
Update salary
set sex = Case when sex = 'm' then 'f'
when sex = 'f' then 'm'
end;
```

```
-- Question 1
-- Table: Users

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | id            | int     |
-- | name          | varchar |
-- +---------------+---------+
-- id is the primary key for this table.
-- name is the name of the user.


-- Table: Rides

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | id            | int     |
-- | user_id       | int     |
-- | distance      | int     |
-- +---------------+---------+
-- id is the primary key for this table.
-- user_id is the id of the user who travelled the distance "distance".


-- Write an SQL query to report the distance travelled by each user.

-- Return the result table ordered by travelled_distance in descending
order,
-- if two or more users travelled the same distance, order them by their
name in ascending order.

-- The query result format is in the following example.



-- Users table:
-- +------+-----------+
-- | id   | name      |
-- +------+-----------+
-- | 1    | Alice     |
-- | 2    | Bob       |
-- | 3    | Alex      |
-- | 4    | Donald    |
-- | 7    | Lee       |
-- | 13   | Jonathan  |
-- | 19   | Elvis     |
-- +------+-----------+

-- Rides table:
-- +------+----------+----------+
-- | id   | user_id  | distance |
-- +------+----------+----------+
-- | 1    | 1        | 120      |
-- | 2    | 2        | 317      |
-- | 3    | 3        | 222      |
-- | 4    | 7        | 100      |
```

```
-- | 5    | 13       | 312      |
-- | 6    | 19       | 50       |
-- | 7    | 7        | 120      |
-- | 8    | 19       | 400      |
-- | 9    | 7        | 230      |
-- +------+----------+----------+

-- Result table:
-- +----------+--------------------+
-- | name     | travelled_distance |
-- +----------+--------------------+
-- | Elvis    | 450                |
-- | Lee      | 450                |
-- | Bob      | 317                |
-- | Jonathan | 312                |
-- | Alex     | 222                |
-- | Alice    | 120                |
-- | Donald   | 0                  |
-- +----------+--------------------+
-- Elvis and Lee travelled 450 miles, Elvis is the top traveller as his
name is alphabetically smaller than Lee.
-- Bob, Jonathan, Alex and Alice have only one ride and we just order
them by the total distances of the ride.
-- Donald didn't have any rides, the distance travelled by him is 0.

-- Solution
Select U.name as name, coalesce(sum(R.distance),0) as travelled_distance
from Users U left join Rides R
on R.user_id = U.id
group by name
Order by travelled_distance desc, name
```

```
-- Question 16
-- A pupil Tim gets homework to identify whether three line segments
could possibly form a triangle.


-- However, this assignment is very heavy because there are hundreds of
records to calculate.


-- Could you help Tim by writing a query to judge whether these three
sides can form a triangle,
-- assuming table triangle holds the length of the three sides x, y and
z.


-- | x  | y  | z  |
-- |----|----|----|
-- | 13 | 15 | 30 |
-- | 10 | 20 | 15 |
-- For the sample data above, your query should return the follow result:
-- | x  | y  | z  | triangle |
-- |----|----|----|----------|
-- | 13 | 15 | 30 | No       |
-- | 10 | 20 | 15 | Yes      |


-- Solution
select x, y, z,
case
when x+y > z and x+z > y and  y+z > x  then 'Yes'
when x=y and y=z then 'Yes'
else 'No'
end as Triangle
from triangle
```

```
-- Question 40
-- 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
Select activity_date as day, count(distinct user_id) as active_users
from activity
where activity_date > '2019-06-26' and activity_date < '2019-07-27'
group by activity_date
```

```
-- Question 35
-- 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
select ifnull(round(avg(a.num),2),0) as average_sessions_per_user
from (
```

```
select count(distinct session_id) as num
from activity
where activity_date between '2019-06-28' and '2019-07-27'
group by user_id) a
```

```
-- Question 46
-- 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.

-- Solution
Select c.country_name,
case when avg(w.weather_state)<=15 then 'Cold'
     when avg(w.weather_state)>=25 then 'Hot'
else 'Warm'
end as weather_type
from weather w join
countries c
on w.country_id = c.country_id
where month(day) = 11
group by c.country_name
```