```sql
-- Question 65
-- 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 occured
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 occurences greater than the average occurences of that event type
among all businesses.

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

-- Events table:
-- +-------------+------------+------------+
-- | business_id | event_type | occurences |
-- +-------------+------------+------------+
-- | 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
select c.business_id
from(
select *
from events e
join
(select event_type as event, round(avg(occurences),2) as average from
events group by event_type) b
on e.event_type = b.event) c
where c.occurences>c.average
group by c.business_id
```

```
having count(*) > 1
```

```
--Question 94
-- Table Accounts:

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


-- Table Logins:

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | id            | int     |
-- | login_date    | date    |
-- +---------------+---------+
-- There is no primary key for this table, it may contain duplicates.
-- This table contains the account id of the user who logged in and the
login date. A user may log in multiple times in the day.


-- Write an SQL query to find the id and the name of active users.

-- Active users are those who logged in to their accounts for 5 or more
consecutive days.

-- Return the result table ordered by the id.

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

-- Accounts table:
-- +----+----------+
-- | id | name     |
-- +----+----------+
-- | 1  | Winston  |
-- | 7  | Jonathan |
-- +----+----------+

-- Logins table:
-- +----+------------+
-- | id | login_date |
-- +----+------------+
-- | 7  | 2020-05-30 |
-- | 1  | 2020-05-30 |
-- | 7  | 2020-05-31 |
-- | 7  | 2020-06-01 |
-- | 7  | 2020-06-02 |
-- | 7  | 2020-06-02 |
-- | 7  | 2020-06-03 |
-- | 1  | 2020-06-07 |
-- | 7  | 2020-06-10 |
-- +----+------------+
-- Result table:
-- +----+----------+
-- | id | name     |
-- +----+----------+
-- | 7  | Jonathan |
-- +----+----------+
-- User Winston with id = 1 logged in 2 times only in 2 different days,
so, Winston is not an active user.
-- User Jonathan with id = 7 logged in 7 times in 6 different days, five
of them were consecutive days, so, Jonathan is an active user.

-- Solution
with t1 as (
select id,login_date,
lead(login_date,4) over(partition by id order by login_date) date_5
from (select distinct * from Logins) b
)

select distinct a.id, a.name from t1
inner join accounts a
on t1.id = a.id
where datediff(t1.date_5,login_date) = 4
order by id
```

```
-- Question 77
-- Table: Friends

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | id            | int     |
-- | name          | varchar |
-- | activity      | varchar |
-- +---------------+---------+
-- id is the id of the friend and primary key for this table.
-- name is the name of the friend.
-- activity is the name of the activity which the friend takes part in.
-- Table: Activities

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


-- Write an SQL query to find the names of all the activities with
neither maximum, nor minimum number of participants.

-- Return the result table in any order. Each activity in table
Activities is performed by any person in the table Friends.

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

-- Friends table:
-- +------+--------------+---------------+
-- | id   | name         | activity      |
-- +------+--------------+---------------+
-- | 1    | Jonathan D.  | Eating        |
-- | 2    | Jade W.      | Singing       |
-- | 3    | Victor J.    | Singing       |
-- | 4    | Elvis Q.     | Eating        |
-- | 5    | Daniel A.    | Eating        |
-- | 6    | Bob B.       | Horse Riding  |
-- +------+--------------+---------------+

-- Activities table:
-- +------+--------------+
-- | id   | name         |
-- +------+--------------+
-- | 1    | Eating       |
-- | 2    | Singing      |
-- | 3    | Horse Riding |
-- +------+--------------+

-- Result table:
-- +--------------+
-- | activity     |
-- +--------------+
-- | Singing      |
-- +--------------+

-- Eating activity is performed by 3 friends, maximum number of
participants, (Jonathan D. , Elvis Q. and Daniel A.)
-- Horse Riding activity is performed by 1 friend, minimum number of
participants, (Bob B.)
-- Singing is performed by 2 friends (Victor J. and Jade W.)

-- Solution
with t1 as(
select max(a.total) as total
from(
    select activity, count(*) as total
    from friends
    group by activity) a
      union all
      select min(b.total) as low
    from(
    select activity, count(*) as total
    from friends
    group by activity) b),
t2 as
(
    select activity, count(*) as total
    from friends
    group by activity
)

select activity
from t1 right join t2
on t1.total = t2.total
where t1.total is null
```

```
-- Question 55
-- 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
select employee_id
from employees
where manager_id = 1 and employee_id != 1
union
select employee_id
from employees
where manager_id = any (select employee_id
from employees
where manager_id = 1 and employee_id != 1)
union
select employee_id
from employees
where manager_id = any (select employee_id
from employees
where manager_id = any (select employee_id
from employees
where manager_id = 1 and employee_id != 1))
```

```
-- Question 66
-- Table: Sales

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | sale_date     | date    |
-- | fruit         | enum    |
-- | sold_num      | int     |
-- +---------------+---------+
-- (sale_date,fruit) is the primary key for this table.
-- This table contains the sales of "apples" and "oranges" sold each day.


-- Write an SQL query to report the difference between number of apples
and oranges sold each day.

-- Return the result table ordered by sale_date in format ('YYYY-MM-DD').

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



-- Sales table:
-- +------------+------------+-------------+
-- | sale_date  | fruit      | sold_num    |
-- +------------+------------+-------------+
-- | 2020-05-01 | apples     | 10          |
-- | 2020-05-01 | oranges    | 8           |
-- | 2020-05-02 | apples     | 15          |
-- | 2020-05-02 | oranges    | 15          |
-- | 2020-05-03 | apples     | 20          |
-- | 2020-05-03 | oranges    | 0           |
-- | 2020-05-04 | apples     | 15          |
-- | 2020-05-04 | oranges    | 16          |
-- +------------+------------+-------------+

-- Result table:
-- +------------+--------------+
-- | sale_date  | diff         |
-- +------------+--------------+
-- | 2020-05-01 | 2            |
-- | 2020-05-02 | 0            |
-- | 2020-05-03 | 20           |
-- | 2020-05-04 | -1           |
-- +------------+--------------+

-- Day 2020-05-01, 10 apples and 8 oranges were sold (Difference  10 - 8
= 2).
-- Day 2020-05-02, 15 apples and 15 oranges were sold (Difference 15 - 15
= 0).
-- Day 2020-05-03, 20 apples and 0 oranges were sold (Difference 20 - 0 =
20).
-- Day 2020-05-04, 15 apples and 16 oranges were sold (Difference 15 - 16
= -1).

-- Solution
Select sale_date, sold_num-sold as diff
```

```
from
((select *
from sales
where fruit = 'apples') a
join
(select sale_date as sale, fruit, sold_num as sold
from sales
where fruit = 'oranges') b
on a.sale_date = b.sale)
```

```
-- Question 81
-- 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 people who viewed more than one
article on the same date, 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 |
-- | 3          | 4         | 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   |
-- +------+
-- | 5    |
-- | 6    |
-- +------+

-- Solution
select distinct viewer_id as id#, count(distinct article_id) as total
from views
group by viewer_id, view_date
having count(distinct article_id)>1
order by 1
```

```
-- Question 74
-- Table Salaries:

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | company_id    | int     |
-- | employee_id   | int     |
-- | employee_name | varchar |
-- | salary        | int     |
-- +---------------+---------+
-- (company_id, employee_id) is the primary key for this table.
-- This table contains the company id, the id, the name and the salary
for an employee.


-- Write an SQL query to find the salaries of the employees after
applying taxes.

-- The tax rate is calculated for each company based on the following
criteria:

-- 0% If the max salary of any employee in the company is less than
1000$.
-- 24% If the max salary of any employee in the company is in the range
[1000, 10000] inclusive.
-- 49% If the max salary of any employee in the company is greater than
10000$.
-- Return the result table in any order. Round the salary to the nearest
integer.

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

-- Salaries table:
-- +------------+-------------+---------------+--------+
-- | company_id | employee_id | employee_name | salary |
-- +------------+-------------+---------------+--------+
-- | 1          | 1           | Tony          | 2000   |
-- | 1          | 2           | Pronub        | 21300  |
-- | 1          | 3           | Tyrrox        | 10800  |
-- | 2          | 1           | Pam           | 300    |
-- | 2          | 7           | Bassem        | 450    |
-- | 2          | 9           | Hermione      | 700    |
-- | 3          | 7           | Bocaben       | 100    |
-- | 3          | 2           | Ognjen        | 2200   |
-- | 3          | 13          | Nyancat       | 3300   |
-- | 3          | 15          | Morninngcat   | 1866   |
-- +------------+-------------+---------------+--------+

-- Result table:
-- +------------+-------------+---------------+--------+
-- | company_id | employee_id | employee_name | salary |
-- +------------+-------------+---------------+--------+
-- | 1          | 1           | Tony          | 1020   |
-- | 1          | 2           | Pronub        | 10863  |
-- | 1          | 3           | Tyrrox        | 5508   |
-- | 2          | 1           | Pam           | 300    |
-- | 2          | 7           | Bassem        | 450    |
-- | 2          | 9           | Hermione      | 700    |
-- | 3          | 7           | Bocaben       | 76     |
-- | 3          | 2           | Ognjen        | 1672   |
-- | 3          | 13          | Nyancat       | 2508   |
-- | 3          | 15          | Morninngcat   | 5911   |
-- +------------+-------------+---------------+--------+
-- For company 1, Max salary is 21300. Employees in company 1 have taxes
= 49%
-- For company 2, Max salary is 700. Employees in company 2 have taxes =
0%
-- For company 3, Max salary is 7777. Employees in company 3 have taxes =
24%
-- The salary after taxes = salary - (taxes percentage / 100) * salary
-- For example, Salary for Morninngcat (3, 15) after taxes = 7777 - 7777
* (24 / 100) = 7777 - 1866.48 = 5910.52, which is rounded to 5911.

-- Solution
with t1 as (
select company_id, employee_id, employee_name, salary as sa, max(salary)
over(partition by company_id) as maximum
from salaries)

select company_id, employee_id, employee_name,
case when t1.maximum<1000 then t1.sa
when t1.maximum between 1000 and 10000 then round(t1.sa*.76,0)
else round(t1.sa*.51,0)
end as salary
from t1
```

```
-- Question 61
-- Table: Stocks

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | stock_name    | varchar |
-- | operation     | enum    |
-- | operation_day | int     |
-- | price         | int     |
-- +---------------+---------+
-- (stock_name, day) is the primary key for this table.
-- The operation column is an ENUM of type ('Sell', 'Buy')
-- Each row of this table indicates that the stock which has stock_name had an operation on the day operation_day with the price.
-- It is guaranteed that each 'Sell' operation for a stock has a corresponding 'Buy' operation in a previous day.


-- Write an SQL query to report the Capital gain/loss for each stock.

-- The capital gain/loss of a stock is total gain or loss after buying and selling the stock one or many times.

-- Return the result table in any order.

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

-- Stocks table:
-- +---------------+-----------+---------------+--------+
-- | stock_name    | operation | operation_day | price  |
-- +---------------+-----------+---------------+--------+
-- | Leetcode      | Buy       | 1             | 1000   |
-- | Corona Masks  | Buy       | 2             | 10     |
-- | Leetcode      | Sell      | 5             | 9000   |
-- | Handbags      | Buy       | 17            | 30000  |
-- | Corona Masks  | Sell      | 3             | 1010   |
-- | Corona Masks  | Buy       | 4             | 1000   |
-- | Corona Masks  | Sell      | 5             | 500    |
-- | Corona Masks  | Buy       | 6             | 1000   |
-- | Handbags      | Sell      | 29            | 7000   |
-- | Corona Masks  | Sell      | 10            | 10000  |
-- +---------------+-----------+---------------+--------+

-- Result table:
-- +---------------+-------------------+
-- | stock_name    | capital_gain_loss |
-- +---------------+-------------------+
-- | Corona Masks  | 9500              |
-- | Leetcode      | 8000              |
-- | Handbags      | -23000            |
-- +---------------+-------------------+
-- Leetcode stock was bought at day 1 for 1000$ and was sold at day 5 for 9000$. Capital gain = 9000 - 1000 = 8000$.
-- Handbags stock was bought at day 17 for 30000$ and was sold at day 29 for 7000$. Capital loss = 7000 - 30000 = -23000$.
-- Corona Masks stock was bought at day 1 for 10$ and was sold at day 3 for 1010$. It was bought again at day 4 for 1000$ and was sold at day 5
```

```
for 500$. At last, it was bought at day 6 for 1000$ and was sold at day
10 for 10000$. Capital gain/loss is the sum of capital gains/losses for
each ('Buy' --> 'Sell')
-- operation = (1010 - 10) + (500 - 1000) + (10000 - 1000) = 1000 - 500 +
9000 = 9500$.

-- Solution
select stock_name, (one-two) as capital_gain_loss
from(
(select stock_name, sum(price) as one
from stocks
where operation = 'Sell'
group by stock_name) b
left join
(select stock_name as name, sum(price) as two
from stocks
where operation = 'Buy'
group by stock_name) c
on b.stock_name = c.name)
order by capital_gain_loss desc
```

```
-- Question 52
-- Write a SQL query to find all numbers that appear at least three times
consecutively.

-- +----+-----+
-- | Id | Num |
-- +----+-----+
-- | 1  |  1  |
-- | 2  |  1  |
-- | 3  |  1  |
-- | 4  |  2  |
-- | 5  |  1  |
-- | 6  |  2  |
-- | 7  |  2  |
-- +----+-----+
-- For example, given the above Logs table, 1 is the only number that
appears consecutively for at least three times.

-- +-----------------+
-- | ConsecutiveNums |
-- +-----------------+
-- | 1               |
-- +-----------------+

-- Solution
select distinct a.num as ConsecutiveNums
from(
select *,
lag(num) over() as prev,
lead(num) over() as next
from logs) a
where a.num = a.prev and a.num=a.next
```

```
-- Question 87
-- A university uses 2 data tables, student and department, to store data
about its students
-- and the departments associated with each major.

-- Write a query to print the respective department name and number of
students majoring in each
-- department for all departments in the department table (even ones with
no current students).

-- Sort your results by descending number of students; if two or more
departments have the same number of students,
-- then sort those departments alphabetically by department name.

-- The student is described as follow:

-- | Column Name  | Type      |
-- |--------------|-----------|
-- | student_id   | Integer   |
-- | student_name | String    |
-- | gender       | Character |
-- | dept_id      | Integer   |
-- where student_id is the student's ID number, student_name is the
student's name, gender is their gender, and dept_id is the department ID
associated with their declared major.

-- And the department table is described as below:

-- | Column Name | Type    |
-- |-------------|---------|
-- | dept_id     | Integer |
-- | dept_name   | String  |
-- where dept_id is the department's ID number and dept_name is the
department name.

-- Here is an example input:
-- student table:

-- | student_id | student_name | gender | dept_id |
-- |------------|--------------|--------|---------|
-- | 1          | Jack         | M      | 1       |
-- | 2          | Jane         | F      | 1       |
-- | 3          | Mark         | M      | 2       |
-- department table:

-- | dept_id | dept_name   |
-- |---------|-------------|
-- | 1       | Engineering |
-- | 2       | Science     |
-- | 3       | Law         |
-- The Output should be:

-- | dept_name   | student_number |
-- |-------------|----------------|
-- | Engineering | 2              |
-- | Science     | 1              |
-- | Law         | 0              |
-- Solution
select dept_name, count(s.dept_id) as student_number
from department d
left join student s
on d.dept_id = s.dept_id
group by d.dept_id
order by count(s.dept_id) desc, dept_name
```

```
-- Question 110
-- Table Person:

-- +----------------+---------+
-- | Column Name    | Type    |
-- +----------------+---------+
-- | id             | int     |
-- | name           | varchar |
-- | phone_number   | varchar |
-- +----------------+---------+
-- id is the primary key for this table.
-- Each row of this table contains the name of a person and their phone
number.
-- Phone number will be in the form 'xxx-yyyyyyy' where xxx is the
country code (3 characters) and yyyyyyy is the
-- phone number (7 characters) where x and y are digits. Both can contain
leading zeros.
-- Table Country:

-- +----------------+---------+
-- | Column Name    | Type    |
-- +----------------+---------+
-- | name           | varchar |
-- | country_code   | varchar |
-- +----------------+---------+
-- country_code is the primary key for this table.
-- Each row of this table contains the country name and its code.
country_code will be in the form 'xxx' where x is digits.


-- Table Calls:

-- +-------------+------+
-- | Column Name | Type |
-- +-------------+------+
-- | caller_id   | int  |
-- | callee_id   | int  |
-- | duration    | int  |
-- +-------------+------+
-- There is no primary key for this table, it may contain duplicates.
-- Each row of this table contains the caller id, callee id and the
duration of the call in minutes. caller_id != callee_id
-- A telecommunications company wants to invest in new countries. The
country intends to invest in the countries where the average call
duration of the calls in this country is strictly greater than the global
average call duration.

-- Write an SQL query to find the countries where this company can
invest.

-- Return the result table in any order.

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

-- Person table:
-- +----+----------+--------------+
-- | id | name     | phone_number |
-- +----+----------+--------------+
-- | 3  | Jonathan | 051-1234567  |
-- | 12 | Elvis    | 051-7654321  |
-- | 1  | Moncef   | 212-1234567  |
-- | 2  | Maroua   | 212-6523651  |
-- | 7  | Meir     | 972-1234567  |
-- | 9  | Rachel   | 972-0011100  |
-- +----+----------+--------------+

-- Country table:
-- +----------+--------------+
-- | name     | country_code |
-- +----------+--------------+
-- | Peru     | 051          |
-- | Israel   | 972          |
-- | Morocco  | 212          |
-- | Germany  | 049          |
-- | Ethiopia | 251          |
-- +----------+--------------+

-- Calls table:
-- +-----------+-----------+----------+
-- | caller_id | callee_id | duration |
-- +-----------+-----------+----------+
-- | 1         | 9         | 33       |
-- | 2         | 9         | 4        |
-- | 1         | 2         | 59       |
-- | 3         | 12        | 102      |
-- | 3         | 12        | 330      |
-- | 12        | 3         | 5        |
-- | 7         | 9         | 13       |
-- | 7         | 1         | 3        |
-- | 9         | 7         | 1        |
-- | 1         | 7         | 7        |
-- +-----------+-----------+----------+

-- Result table:
-- +----------+
-- | country  |
-- +----------+
-- | Peru     |
-- +----------+
-- The average call duration for Peru is (102 + 102 + 330 + 330 + 5 + 5)
/ 6 = 145.666667
-- The average call duration for Israel is (33 + 4 + 13 + 13 + 3 + 1 + 1
+ 7) / 8 = 9.37500
-- The average call duration for Morocco is (33 + 4 + 59 + 59 + 3 + 7) /
6 = 27.5000
-- Global call duration average = (2 * (33 + 3 + 59 + 102 + 330 + 5 + 13
+ 3 + 1 + 7)) / 20 = 55.70000
-- Since Peru is the only country where average call duration is greater
than the global average, it's the only recommended country.

-- Solution
with t1 as(
select caller_id as id, duration as total
from
(select caller_id, duration
from calls
union all
select callee_id, duration
from calls) a
)
select name as country
from
(select distinct avg(total) over(partition by code) as avg_call,
avg(total) over() as global_avg, c.name
from
((select *, coalesce(total,0) as duration, substring(phone_number from 1
for 3) as code
from person right join t1
using (id)) b
join country c
on c.country_code = b.code)) d
where avg_call > global_avg
```

```
-- Question 72
-- Table: Customers

-- +---------------------+---------+
-- | Column Name         | Type    |
-- +---------------------+---------+
-- | customer_id         | int     |
-- | customer_name       | varchar |
-- +---------------------+---------+
-- customer_id is the primary key for this table.
-- customer_name is the name of the customer.


-- Table: Orders

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | order_id      | int     |
-- | customer_id   | int     |
-- | product_name  | varchar |
-- +---------------+---------+
-- order_id is the primary key for this table.
-- customer_id is the id of the customer who bought the product
"product_name".


-- Write an SQL query to report the customer_id and customer_name of
customers who bought products "A", "B" but did not buy the product "C"
since we want to recommend them buy this product.

-- Return the result table ordered by customer_id.

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



-- Customers table:
-- +-------------+---------------+
-- | customer_id | customer_name |
-- +-------------+---------------+
-- | 1           | Daniel        |
-- | 2           | Diana         |
-- | 3           | Elizabeth     |
-- | 4           | Jhon          |
-- +-------------+---------------+

-- Orders table:
-- +------------+--------------+---------------+
-- | order_id   | customer_id  | product_name  |
-- +------------+--------------+---------------+
-- | 10         |     1        |     A         |
-- | 20         |     1        |     B         |
-- | 30         |     1        |     D         |
-- | 40         |     1        |     C         |
-- | 50         |     2        |     A         |
-- | 60         |     3        |     A         |
-- | 70         |     3        |     B         |
-- | 80         |     3        |     D         |
-- | 90         |     4        |     C         |
-- +------------+--------------+---------------+

-- Result table:
-- +-------------+---------------+
-- | customer_id | customer_name |
-- +-------------+---------------+
-- | 3           | Elizabeth     |
-- +-------------+---------------+
-- Only the customer_id with id 3 bought the product A and B but not the
product C.

-- Solution
with t1 as
(
select customer_id
from orders
where product_name = 'B' and
customer_id in (select customer_id
from orders
where product_name = 'A'))

Select t1.customer_id, c.customer_name
from t1 join customers c
on t1.customer_id = c.customer_id
where t1.customer_id != all(select customer_id
from orders
where product_name = 'C')
```

```
-- Question 93
-- Table: Customer

-- +-------------+---------+
-- | Column Name | Type    |
-- +-------------+---------+
-- | customer_id | int     |
-- | product_key | int     |
-- +-------------+---------+
-- product_key is a foreign key to Product table.
-- Table: Product

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


-- Write an SQL query for a report that provides the customer ids from
the Customer table that bought all the products in the Product table.

-- For example:

-- Customer table:
-- +-------------+-------------+
-- | customer_id | product_key |
-- +-------------+-------------+
-- | 1           | 5           |
-- | 2           | 6           |
-- | 3           | 5           |
-- | 3           | 6           |
-- | 1           | 6           |
-- +-------------+-------------+

-- Product table:
-- +-------------+
-- | product_key |
-- +-------------+
-- | 5           |
-- | 6           |
-- +-------------+

-- Result table:
-- +-------------+
-- | customer_id |
-- +-------------+
-- | 1           |
-- | 3           |
-- +-------------+
-- The customers who bought all the products (5 and 6) are customers with
id 1 and 3.

-- Solution
select customer_id
from customer
group by customer_id
```

```
having count(distinct product_key) = (select COUNT(distinct product_key)
from product)
```

```
-- Question 57
-- The Employee table holds all employees. Every employee has an Id, a
salary, and there is also a column for the department Id.

-- +----+-------+--------+--------------+
-- | Id | Name  | Salary | DepartmentId |
-- +----+-------+--------+--------------+
-- | 1  | Joe   | 70000  | 1            |
-- | 2  | Jim   | 90000  | 1            |
-- | 3  | Henry | 80000  | 2            |
-- | 4  | Sam   | 60000  | 2            |
-- | 5  | Max   | 90000  | 1            |
-- +----+-------+--------+--------------+
-- The Department table holds all departments of the company.

-- +----+----------+
-- | Id | Name     |
-- +----+----------+
-- | 1  | IT       |
-- | 2  | Sales    |
-- +----+----------+
-- Write a SQL query to find employees who have the highest salary in
each of the departments.
-- For the above tables, your SQL query should return the following rows
(order of rows does not matter).

-- +------------+----------+--------+
-- | Department | Employee | Salary |
-- +------------+----------+--------+
-- | IT         | Max      | 90000  |
-- | IT         | Jim      | 90000  |
-- | Sales      | Henry    | 80000  |
-- +------------+----------+--------+
-- Explanation:

-- Max and Jim both have the highest salary in the IT department and
Henry has the highest salary in the Sales department.

-- Solution
select a.Department, a.Employee, a.Salary
from(
select d.name as Department, e.name as Employee, Salary,
rank() over(partition by d.name order by salary desc) as rk
from employee e
join department d
on e.departmentid = d.id) a
where a.rk=1
```

```
-- Question 78
-- Table Variables:

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | name          | varchar |
-- | value         | int     |
-- +---------------+---------+
-- name is the primary key for this table.
-- This table contains the stored variables and their values.


-- Table Expressions:

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | left_operand  | varchar |
-- | operator      | enum    |
-- | right_operand | varchar |
-- +---------------+---------+
-- (left_operand, operator, right_operand) is the primary key for this
table.
-- This table contains a boolean expression that should be evaluated.
-- operator is an enum that takes one of the values ('<', '>', '=')
-- The values of left_operand and right_operand are guaranteed to be in
the Variables table.


-- Write an SQL query to evaluate the boolean expressions in Expressions
table.

-- Return the result table in any order.

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

-- Variables table:
-- +------+-------+
-- | name | value |
-- +------+-------+
-- | x    | 66    |
-- | y    | 77    |
-- +------+-------+

-- Expressions table:
-- +--------------+----------+---------------+
-- | left_operand | operator | right_operand |
-- +--------------+----------+---------------+
-- | x            | >        | y             |
-- | x            | <        | y             |
-- | x            | =        | y             |
-- | y            | >        | x             |
-- | y            | <        | x             |
-- | x            | =        | x             |
-- +--------------+----------+---------------+

-- Result table:
```

```
-- +--------------+----------+---------------+-------+
-- | left_operand | operator | right_operand | value |
-- +--------------+----------+---------------+-------+
-- | x            | >        | y             | false |
-- | x            | <        | y             | true  |
-- | x            | =        | y             | false |
-- | y            | >        | x             | true  |
-- | y            | <        | x             | false |
-- | x            | =        | x             | true  |
-- +--------------+----------+---------------+-------+
-- As shown, you need find the value of each boolean exprssion in the
table using the variables table.

-- Solution
with t1 as(
select e.left_operand, e.operator, e.right_operand, v.value as left_val,
v_1.value as right_val
from expressions e
join variables v
on v.name = e.left_operand
join variables v_1
on v_1.name = e.right_operand)

select t1.left_operand, t1.operator, t1.right_operand,
case when t1.operator = '<' then (select t1.left_val< t1.right_val)
when t1.operator = '>' then (select t1.left_val > t1.right_val)
when t1.operator = '=' then (select t1.left_val = t1.right_val)
else FALSE
END AS VALUE
from t1
```

```sql
-- Question 56
-- Mary is a teacher in a middle school and she has a table seat storing
students' names and their corresponding seat ids.

-- The column id is continuous increment.


-- Mary wants to change seats for the adjacent students.


-- Can you write a SQL query to output the result for Mary?


-- +---------+---------+
-- |    id   | student |
-- +---------+---------+
-- |    1    | Abbot   |
-- |    2    | Doris   |
-- |    3    | Emerson |
-- |    4    | Green   |
-- |    5    | Jeames  |
-- +---------+---------+
-- For the sample input, the output is:


-- +---------+---------+
-- |    id   | student |
-- +---------+---------+
-- |    1    | Doris   |
-- |    2    | Abbot   |
-- |    3    | Green   |
-- |    4    | Emerson |
-- |    5    | Jeames  |
-- +---------+---------+

-- Solution
select row_number() over (order by (if(id%2=1,id+1,id-1))) as id, student
from seat
```

```
-- Question 80
-- 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
select min(log_id) as start_id, max(log_id) as end_id
from(
select log_id, log_id-row_number() over (order by log_id) as rk
from logs) a
group by rk
```

```
-- Question 60
-- In social network like Facebook or Twitter, people send friend
requests and accept others' requests as well.

-- 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 |
-- +--------------+-------------+------------+
-- This table holds the data of friend acceptance, while requester_id and
accepter_id both are the id of a person.


-- Write a query to find the the people who has most friends and the most
friends number under the following rules:

-- It is guaranteed there is only 1 people having the most friends.
-- The friend request could only been accepted once, which mean there is
no multiple records with the same requester_id and accepter_id value.
-- For the sample data above, the result is:

-- Result table:
-- +------+------+
-- | id   | num  |
-- |------|------|
-- | 3    | 3    |
-- +------+------+
-- The person with id '3' is a friend of people '1', '2' and '4', so he
has 3 friends in total, which is the most number than any others.

-- Solution
select requester_id as id, b.total as num
from(
select requester_id, sum(one) as total
from((
select requester_id, count(distinct accepter_id) as one
from request_accepted
group by requester_id)
union all
(select accepter_id, count(distinct requester_id) as two
from request_accepted
group by accepter_id)) a
group by requester_id
order by total desc) b
limit 1
```

```
-- Question 62
-- 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 for each player and date, how many
games played so far by the player. That is, the total number of games
played by the player until that date. Check the example for clarity.

-- 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            |
-- | 1         | 3         | 2017-06-25 | 1            |
-- | 3         | 1         | 2016-03-02 | 0            |
-- | 3         | 4         | 2018-07-03 | 5            |
-- +-----------+-----------+------------+--------------+

-- Result table:
-- +-----------+------------+---------------------+
-- | player_id | event_date | games_played_so_far |
-- +-----------+------------+---------------------+
-- | 1         | 2016-03-01 | 5                   |
-- | 1         | 2016-05-02 | 11                  |
-- | 1         | 2017-06-25 | 12                  |
-- | 3         | 2016-03-02 | 0                   |
-- | 3         | 2018-07-03 | 5                   |
-- +-----------+------------+---------------------+
-- For the player with id 1, 5 + 6 = 11 games played by 2016-05-02, and 5
+ 6 + 1 = 12 games played by 2017-06-25.
-- For the player with id 3, 0 + 5 = 5 games played by 2018-07-03.
-- Note that for each player we only care about the days when the player
logged in.

-- Solution
select player_id, event_date,
sum(games_played) over(partition by player_id order by event_date) as
games_played_so_far
from activity
order by 1,2
```

```
-- Question 91
-- 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 fraction of players that logged in
again
-- on the day after the day they first logged in, rounded to 2 decimal
places.
-- In other words, you need to count the number of players that logged in
for at least two consecutive
-- days starting from their first login date, then divide that number by
the total number of players.

-- 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         | 2016-03-02 | 6            |
-- | 2         | 3         | 2017-06-25 | 1            |
-- | 3         | 1         | 2016-03-02 | 0            |
-- | 3         | 4         | 2018-07-03 | 5            |
-- +-----------+-----------+------------+--------------+

-- Result table:
-- +-----------+
-- | fraction  |
-- +-----------+
-- | 0.33      |
-- +-----------+
-- Only the player with id 1 logged back in after the first day he had
logged in so the answer is 1/3 = 0.33

-- Solution
With t as
(select player_id,
 min(event_date) over(partition by player_id) as min_event_date,
 case when event_date- min(event_date) over(partition by player_id) = 1
then 1
 else 0
 end as s
 from Activity)
```

```
select round(sum(t.s)/count(distinct t.player_id),2) as fraction
from t
```

```
-- Question 86
-- Get the highest answer rate question from a table survey_log with
these columns: id, action, question_id, answer_id, q_num, timestamp.

-- id means user id; action has these kind of values: "show", "answer",
"skip"; answer_id is not null when action column is "answer",
-- while is null for "show" and "skip"; q_num is the numeral order of the
question in current session.

-- Write a sql query to identify the question which has the highest
answer rate.

-- Example:

-- Input:
-- +------+-----------+--------------+------------+-----------+----------
--+
-- | id   | action    | question_id  | answer_id  | q_num     | timestamp
|
-- +------+-----------+--------------+------------+-----------+----------
--+
-- | 5    | show      | 285          | null       | 1         | 123
|
-- | 5    | answer    | 285          | 124124     | 1         | 124
|
-- | 5    | show      | 369          | null       | 2         | 125
|
-- | 5    | skip      | 369          | null       | 2         | 126
|
-- +------+-----------+--------------+------------+-----------+----------
--+
-- Output:
-- +-------------+
-- | survey_log  |
-- +-------------+
-- |    285      |
-- +-------------+
-- Explanation:
-- question 285 has answer rate 1/1, while question 369 has 0/1 answer
rate, so output 285.


-- Note: The highest answer rate meaning is: answer number's ratio in
show number in the same question.

-- Solution
with t1 as(
select a.question_id, coalesce(b.answer/a.show_1,0) as rate
from
(select question_id, coalesce(count(*),0) as show_1
from survey_log
where action != 'answer'
group by question_id) a
left join
(select question_id, coalesce(count(*),0) as answer
from survey_log
where action = 'answer'
group by question_id) b
on a.question_id = b.question_id)

select a.question_id as survey_log
from
( select t1.question_id,
rank() over(order by rate desc) as rk
from t1) a
where a.rk = 1
```

```sql
-- Question 109
-- Table: UserActivity

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | username      | varchar |
-- | activity      | varchar |
-- | startDate     | Date    |
-- | endDate       | Date    |
-- +---------------+---------+
-- This table does not contain primary key.
-- This table contain information about the activity performed of each
user in a period of time.
-- A person with username performed a activity from startDate to endDate.

-- Write an SQL query to show the second most recent activity of each
user.

-- If the user only has one activity, return that one.

-- A user can't perform more than one activity at the same time. Return
the result table in any order.

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

-- UserActivity table:
-- +------------+--------------+-------------+-------------+
-- | username   | activity     | startDate   | endDate     |
-- +------------+--------------+-------------+-------------+
-- | Alice      | Travel       | 2020-02-12  | 2020-02-20  |
-- | Alice      | Dancing      | 2020-02-21  | 2020-02-23  |
-- | Alice      | Travel       | 2020-02-24  | 2020-02-28  |
-- | Bob        | Travel       | 2020-02-11  | 2020-02-18  |
-- +------------+--------------+-------------+-------------+

-- Result table:
-- +------------+--------------+-------------+-------------+
-- | username   | activity     | startDate   | endDate     |
-- +------------+--------------+-------------+-------------+
-- | Alice      | Dancing      | 2020-02-21  | 2020-02-23  |
-- | Bob        | Travel       | 2020-02-11  | 2020-02-18  |
-- +------------+--------------+-------------+-------------+

-- The most recent activity of Alice is Travel from 2020-02-24 to 2020-
02-28, before that she was dancing from 2020-02-21 to 2020-02-23.
-- Bob only has one record, we just take that one.

-- Solution
select username, activity, startdate, enddate
from
(select *,
rank() over(partition by username order by startdate desc) as rk,
count(username) over(partition by username) as cnt
from useractivity) a
where a.rk = 2 or cnt = 1
```

```
-- Question 63
-- 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
select student_id, course_id, grade
from(
select student_id, course_id, grade,
rank() over(partition by student_id order by grade desc, course_id) as rk
from enrollments) a
where a.rk = 1
```

```
-- Question 82
-- 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
select
round(avg(case when order_date = customer_pref_delivery_date then 1 else
0 end)*100,2) as
immediate_percentage
from
(select *,
 rank() over(partition by customer_id order by order_date) as rk
from delivery) a
where a.rk=1
```

```sql
-- Question 96
-- Write a query to print the sum of all total investment values in 2016
(TIV_2016), to a scale of 2 decimal places, for all policy holders who
meet the following criteria:

-- Have the same TIV_2015 value as one or more other policyholders.
-- Are not located in the same city as any other policyholder (i.e.: the
(latitude, longitude) attribute pairs must be unique).
-- Input Format:
-- The insurance table is described as follows:

-- | Column Name | Type          |
-- |-------------|---------------|
-- | PID         | INTEGER(11)   |
-- | TIV_2015    | NUMERIC(15,2) |
-- | TIV_2016    | NUMERIC(15,2) |
-- | LAT         | NUMERIC(5,2)  |
-- | LON         | NUMERIC(5,2)  |
-- where PID is the policyholder's policy ID, TIV_2015 is the total
investment value in 2015, TIV_2016 is the total investment value in 2016,
LAT is the latitude of the policy holder's city, and LON is the longitude
of the policy holder's city.

-- Sample Input

-- | PID | TIV_2015 | TIV_2016 | LAT | LON |
-- |-----|----------|----------|-----|-----|
-- | 1   | 10       | 5        | 10  | 10  |
-- | 2   | 20       | 20       | 20  | 20  |
-- | 3   | 10       | 30       | 20  | 20  |
-- | 4   | 10       | 40       | 40  | 40  |
-- Sample Output

-- | TIV_2016 |
-- |----------|
-- | 45.00    |
-- Explanation

-- The first record in the table, like the last record, meets both of the
two criteria.
-- The TIV_2015 value '10' is as the same as the third and forth record,
and its location unique.

-- The second record does not meet any of the two criteria. Its TIV_2015
is not like any other policyholders.

-- And its location is the same with the third record, which makes the
third record fail, too.

-- So, the result is the sum of TIV_2016 of the first and last record,
which is 45.

-- Solution
select sum(TIV_2016) TIV_2016
from
(select *, count(*) over (partition by TIV_2015) as c1, count(*) over
(partition by LAT, LON) as c2
from insurance ) t
```

```
where c1 > 1 and c2 = 1;
```

```
-- Question 68
-- 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 Johnliams    | 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
With t1 as
(
select *,
sum(weight) over(order by turn) as cum_weight
from queue
order by turn)
```

```
select t1.person_name
from t1
where turn = (select max(turn) from t1 where t1.cum_weight<=1000)
```

```
-- Question 75
-- 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    |Department |ManagerId |
-- +------+----------+-----------+----------+
-- |101   |John    |A         |null      |
-- |102   |Dan     |A         |101       |
-- |103   |James   |A         |101       |
-- |104   |Amy     |A         |101       |
-- |105   |Anne    |A         |101       |
-- |106   |Ron     |B         |101       |
-- +------+----------+-----------+----------+
-- Given the Employee table, write a SQL query that finds out managers
with at least 5 direct report. For the above table, your SQL query should
return:

-- +-------+
-- | Name  |
-- +-------+
-- | John  |
-- +-------+
-- Note:
-- No one would report to himself.

-- Solution
with t1 as
(
    select managerid, count(name) as total
    from employee
    group by managerid
)

select e.name
from t1
join employee e
on t1.managerid = e.id
where t1.total>=5
```

```
-- Question 69
-- Table: Users

-- +----------------+---------+
-- | Column Name    | Type    |
-- +----------------+---------+
-- | user_id        | int     |
-- | join_date      | date    |
-- | favorite_brand | varchar |
-- +----------------+---------+
-- user_id is the primary key of this table.
-- This table has the info of the users of an online shopping website
where users can sell and buy items.
-- Table: Orders

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | order_id      | int     |
-- | order_date    | date    |
-- | item_id       | int     |
-- | buyer_id      | int     |
-- | seller_id     | int     |
-- +---------------+---------+
-- order_id is the primary key of this table.
-- item_id is a foreign key to the Items table.
-- buyer_id and seller_id are foreign keys to the Users table.
-- Table: Items

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


-- Write an SQL query to find for each user, the join date and the number
of orders they made as a buyer in 2019.

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

-- Users table:
-- +---------+------------+----------------+
-- | user_id | join_date  | favorite_brand |
-- +---------+------------+----------------+
-- | 1       | 2018-01-01 | Lenovo         |
-- | 2       | 2018-02-09 | Samsung        |
-- | 3       | 2018-01-19 | LG             |
-- | 4       | 2018-05-21 | HP             |
-- +---------+------------+----------------+

-- Orders table:
-- +----------+------------+---------+----------+-----------+
-- | order_id | order_date | item_id | buyer_id | seller_id |
-- +----------+------------+---------+----------+-----------+
-- | 1        | 2019-08-01 | 4       | 1        | 2         |
-- | 2        | 2018-08-02 | 2       | 1        | 3         |
-- | 3        | 2019-08-03 | 3       | 2        | 3         |
-- | 4        | 2018-08-04 | 1       | 4        | 2         |
-- | 5        | 2018-08-04 | 1       | 3        | 4         |
-- | 6        | 2019-08-05 | 2       | 2        | 4         |
-- +----------+------------+---------+----------+-----------+

-- Items table:
-- +---------+------------+
-- | item_id | item_brand |
-- +---------+------------+
-- | 1       | Samsung    |
-- | 2       | Lenovo     |
-- | 3       | LG         |
-- | 4       | HP         |
-- +---------+------------+

-- Result table:
-- +-----------+------------+----------------+
-- | buyer_id  | join_date  | orders_in_2019 |
-- +-----------+------------+----------------+
-- | 1         | 2018-01-01 | 1              |
-- | 2         | 2018-02-09 | 2              |
-- | 3         | 2018-01-19 | 0              |
-- | 4         | 2018-05-21 | 0              |
-- +-----------+------------+----------------+

-- Solution
select user_id as buyer_id, join_date, coalesce(a.orders_in_2019,0)
from users
left join
(
select buyer_id, coalesce(count(*), 0) as orders_in_2019
from orders o
join users u
on u.user_id = o.buyer_id
where extract('year'from order_date) = 2019
group by buyer_id) a
on users.user_id = a.buyer_id
```

```
-- Question 95
-- 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"].
-- Table: Chargebacks

-- +----------------+---------+
-- | Column Name    | Type    |
-- +----------------+---------+
-- | trans_id       | int     |
-- | charge_date    | date    |
-- +----------------+---------+
-- Chargebacks contains basic information regarding incoming chargebacks
from some transactions placed in Transactions table.
-- trans_id is a foreign key to the id column of Transactions table.
-- Each chargeback corresponds to a transaction made previously even if
they were not approved.


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

-- Note: In your query, given the month and country, ignore rows with all
zeros.

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

-- Transactions table:
-- +------+---------+----------+--------+------------+
-- | id   | country | state    | amount | trans_date |
-- +------+---------+----------+--------+------------+
-- | 101  | US      | approved | 1000   | 2019-05-18 |
-- | 102  | US      | declined | 2000   | 2019-05-19 |
-- | 103  | US      | approved | 3000   | 2019-06-10 |
-- | 104  | US      | approved | 4000   | 2019-06-13 |
-- | 105  | US      | approved | 5000   | 2019-06-15 |
-- +------+---------+----------+--------+------------+

-- Chargebacks table:
-- +------------+------------+
-- | trans_id   | trans_date |
-- +------------+------------+
-- | 102        | 2019-05-29 |
-- | 101        | 2019-06-30 |
-- | 105        | 2019-09-18 |
-- +------------+------------+
-- Result table:
-- +----------+---------+---------------+----------------+-------------
------+--------------------+
-- | month    | country | approved_count | approved_amount |
chargeback_count  | chargeback_amount  |
-- +----------+---------+---------------+----------------+-------------
------+--------------------+
-- | 2019-05  | US      | 1             | 1000           | 1
| 2000               |
-- | 2019-06  | US      | 3             | 12000          | 1
| 1000               |
-- | 2019-09  | US      | 0             | 0              | 1
| 5000               |
-- +----------+---------+---------------+----------------+-------------
------+--------------------+

-- Solution
with t1 as
(select country, extract('month' from trans_date), state, count(*) as
approved_count, sum(amount) as approved_amount
from transactions
where state = 'approved'
group by 1, 2, 3),
t2 as(
select t.country, extract('month' from c.trans_date), sum(amount) as
chargeback_amount, count(*) as chargeback_count
from chargebacks c left join transactions t
on trans_id = id
group by t.country, extract('month' from c.trans_date)),

t3 as(
select t2.date_part, t2.country, coalesce(approved_count,0) as
approved_count, coalesce(approved_amount,0) as approved_amount,
coalesce(chargeback_count,0) as chargeback_count,
coalesce(chargeback_amount,0) as chargeback_amount
from t2 left join t1
on t2.date_part = t1.date_part and t2.country = t1.country),

t4 as(
select t1.date_part, t1.country, coalesce(approved_count,0) as
approved_count, coalesce(approved_amount,0) as approved_amount,
coalesce(chargeback_count,0) as chargeback_count,
coalesce(chargeback_amount,0) as chargeback_amount
from t2 right join t1
on t2.date_part = t1.date_part and t2.country = t1.country)

select *
from t3
union
select *
from t4
```

```
-- Question 83
-- 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
with t1 as(
select DATE_FORMAT(trans_date,'%Y-%m') as month, country, count(state) as
trans_count, sum(amount) as trans_total_amount
from transactions
group by country, month(trans_date)),

t2 as (
Select DATE_FORMAT(trans_date,'%Y-%m') as month, country, count(state) as
approved_count, sum(amount) as approved_total_amount
from transactions
where state = 'approved'
group by country, month(trans_date))

select t1.month, t1.country, coalesce(t1.trans_count,0) as trans_count,
coalesce(t2.approved_count,0) as approved_count,
coalesce(t1.trans_total_amount,0) as trans_total_amount,
coalesce(t2.approved_total_amount,0) as approved_total_amount
from t1 left join t2
on t1.country = t2.country and t1.month = t2.month
```

```
-- Question 59
-- Table: Movies

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | movie_id      | int     |
-- | title         | varchar |
-- +---------------+---------+
-- movie_id is the primary key for this table.
-- title is the name of the movie.
-- Table: Users

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | user_id       | int     |
-- | name          | varchar |
-- +---------------+---------+
-- user_id is the primary key for this table.
-- Table: Movie_Rating

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | movie_id      | int     |
-- | user_id       | int     |
-- | rating        | int     |
-- | created_at    | date    |
-- +---------------+---------+
-- (movie_id, user_id) is the primary key for this table.
-- This table contains the rating of a movie by a user in their review.
-- created_at is the user's review date.


-- Write the following SQL query:

-- Find the name of the user who has rated the greatest number of the
movies.
-- In case of a tie, return lexicographically smaller user name.

-- Find the movie name with the highest average rating in February 2020.
-- In case of a tie, return lexicographically smaller movie name.

-- Query is returned in 2 rows, the query result format is in the
folowing example:

-- Movies table:
-- +-------------+--------------+
-- | movie_id    |  title       |
-- +-------------+--------------+
-- | 1           | Avengers     |
-- | 2           | Frozen 2     |
-- | 3           | Joker        |
-- +-------------+--------------+

-- Users table:
-- +-------------+--------------+
-- | user_id     |  name        |
-- +-------------+--------------+
-- | 1           | Daniel       |
-- | 2           | Monica       |
-- | 3           | Maria        |
-- | 4           | James        |
-- +-------------+--------------+

-- Movie_Rating table:
-- +-------------+--------------+--------------+-------------+
-- | movie_id    | user_id      | rating       | created_at  |
-- +-------------+--------------+--------------+-------------+
-- | 1           | 1            | 3            | 2020-01-12  |
-- | 1           | 2            | 4            | 2020-02-11  |
-- | 1           | 3            | 2            | 2020-02-12  |
-- | 1           | 4            | 1            | 2020-01-01  |
-- | 2           | 1            | 5            | 2020-02-17  |
-- | 2           | 2            | 2            | 2020-02-01  |
-- | 2           | 3            | 2            | 2020-03-01  |
-- | 3           | 1            | 3            | 2020-02-22  |
-- | 3           | 2            | 4            | 2020-02-25  |
-- +-------------+--------------+--------------+-------------+

-- Result table:
-- +--------------+
-- | results      |
-- +--------------+
-- | Daniel       |
-- | Frozen 2     |
-- +--------------+

-- Daniel and Maria have rated 3 movies ("Avengers", "Frozen 2" and
"Joker") but Daniel is smaller lexicographically.
-- Frozen 2 and Joker have a rating average of 3.5 in February but Frozen
2 is smaller lexicographically.

-- Solution
select name as results
from(
(select a.name
from(
select name, count(*),
rank() over(order by count(*) desc) as rk
from movie_rating m
join users u
on m.user_id = u.user_id
group by name, m.user_id
order by rk, name) a
limit 1)
union
(select title
from(
select title, round(avg(rating),1) as rnd
from movie_rating m
join movies u
on m.movie_id = u.movie_id
where month(created_at) = 2
group by title
order by rnd desc, title) b
limit 1)) as d
```

```
-- Question 92
-- Table: Traffic

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | user_id       | int     |
-- | activity      | enum    |
-- | activity_date | date    |
-- +---------------+---------+
-- There is no primary key for this table, it may have duplicate rows.
-- The activity column is an ENUM type of ('login', 'logout', 'jobs',
'groups', 'homepage').


-- Write an SQL query that reports for every date within at most 90 days
from today,
-- the number of users that logged in for the first time on that date.
Assume today is 2019-06-30.

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

-- Traffic table:
-- +---------+----------+---------------+
-- | user_id | activity | activity_date |
-- +---------+----------+---------------+
-- | 1       | login    | 2019-05-01    |
-- | 1       | homepage | 2019-05-01    |
-- | 1       | logout   | 2019-05-01    |
-- | 2       | login    | 2019-06-21    |
-- | 2       | logout   | 2019-06-21    |
-- | 3       | login    | 2019-01-01    |
-- | 3       | jobs     | 2019-01-01    |
-- | 3       | logout   | 2019-01-01    |
-- | 4       | login    | 2019-06-21    |
-- | 4       | groups   | 2019-06-21    |
-- | 4       | logout   | 2019-06-21    |
-- | 5       | login    | 2019-03-01    |
-- | 5       | logout   | 2019-03-01    |
-- | 5       | login    | 2019-06-21    |
-- | 5       | logout   | 2019-06-21    |
-- +---------+----------+---------------+

-- Result table:
-- +------------+-------------+
-- | login_date | user_count  |
-- +------------+-------------+
-- | 2019-05-01 | 1           |
-- | 2019-06-21 | 2           |
-- +------------+-------------+
-- Note that we only care about dates with non zero user count.
-- The user with id 5 first logged in on 2019-03-01 so he's not counted
on 2019-06-21.

-- Solution
with t1 as
(
    select user_id, min(activity_date) as login_date
    from Traffic
    where activity = 'login'
    group by user_id
)

select login_date, count(distinct user_id) as user_count
from t1
where login_date between '2019-04-01' and '2019-06-30'
group by login_date
```

```
-- Question 54
-- Table: NPV

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | id            | int     |
-- | year          | int     |
-- | npv           | int     |
-- +---------------+---------+
-- (id, year) is the primary key of this table.
-- The table has information about the id and the year of each inventory
and the corresponding net present value.


-- Table: Queries

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | id            | int     |
-- | year          | int     |
-- +---------------+---------+
-- (id, year) is the primary key of this table.
-- The table has information about the id and the year of each inventory
query.


-- Write an SQL query to find the npv of all each query of queries table.

-- Return the result table in any order.

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

-- NPV table:
-- +------+--------+--------+
-- | id   | year   | npv    |
-- +------+--------+--------+
-- | 1    | 2018   | 100    |
-- | 7    | 2020   | 30     |
-- | 13   | 2019   | 40     |
-- | 1    | 2019   | 113    |
-- | 2    | 2008   | 121    |
-- | 3    | 2009   | 12     |
-- | 11   | 2020   | 99     |
-- | 7    | 2019   | 0      |
-- +------+--------+--------+

-- Queries table:
-- +------+--------+
-- | id   | year   |
-- +------+--------+
-- | 1    | 2019   |
-- | 2    | 2008   |
-- | 3    | 2009   |
-- | 7    | 2018   |
-- | 7    | 2019   |
-- | 7    | 2020   |
-- | 13   | 2019   |
-- +------+--------+

-- Result table:
-- +------+--------+--------+
-- | id   | year   | npv    |
-- +------+--------+--------+
-- | 1    | 2019   | 113    |
-- | 2    | 2008   | 121    |
-- | 3    | 2009   | 12     |
-- | 7    | 2018   | 0      |
-- | 7    | 2019   | 0      |
-- | 7    | 2020   | 30     |
-- | 13   | 2019   | 40     |
-- +------+--------+--------+

-- The npv value of (7, 2018) is not present in the NPV table, we
consider it 0.
-- The npv values of all other queries can be found in the NPV table.

-- Solution
select q.id, q.year, coalesce(n.npv,0) as npv
from queries q
left join npv n
on q.id = n.id and q.year=n.year
```

```
-- Question 50
-- Write a SQL query to get the nth highest salary from the Employee
table.

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

-- +------------------------+
-- | getNthHighestSalary(2) |
-- +------------------------+
-- | 200                    |
-- +------------------------+

-- Solution
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
  RETURN (
      # Write your MySQL query statement below.
      select distinct a.salary
      from
      (select salary,
      dense_rank() over(order by salary desc) as rk
      from Employee) a
      where a.rk = N
  );
END
```

```
-- Question 84
-- 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
select distinct page_id as recommended_page
from likes
where user_id =
any(select user2_id as id
from friendship
where user1_id = 1 or user2_id = 1 and user2_id !=1
union all
select user1_id
from friendship
where user2_id = 1)
and page_id != all(select page_id from likes where user_id = 1)
```

```sql
-- Question 67
-- 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
with t1 as (
select a.product_id, new_price
from(
Select product_id, max(change_date) as date
from products
where change_date<='2019-08-16'
group by product_id) a
join products p
on a.product_id = p.product_id and a.date = p.change_date),

t2 as (
select distinct product_id
      from products)

select t2.product_id, coalesce(new_price,10) as price
from t2 left join t1
```

```
on t2.product_id = t1.product_id
order by price desc
```

```
-- Question 90
-- 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 selects the product id, year, quantity, and
price for the first year of every product sold.

-- 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 | first_year | quantity | price |
-- +------------+------------+----------+-------+
-- | 100        | 2008       | 10       | 5000  |
-- | 200        | 2011       | 15       | 9000  |
-- +------------+------------+----------+-------+

-- Solution
```

```
select a.product_id, a.year as first_year, a.quantity, a.price
from
( select product_id, quantity, price, year,
  rank() over(partition by product_id order by year) as rk
  from sales
) a
where a.rk = 1
```

```
-- Question 85
-- 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
with t1 as(
select p.project_id, p.employee_id, e.experience_years,
rank() over(partition by project_id order by experience_years desc) as rk
from project p
join employee e
on p.employee_id = e.employee_id)

select t1.project_id, t1.employee_id
from t1
where t1.rk = 1
```

```
-- Question 51
-- Write a SQL query to rank scores.
-- If there is a tie between two scores, both should have the same
ranking.
-- Note that after a tie, the next ranking number should be the next
consecutive integer value.
-- In other words, there should be no "holes" between ranks.

-- +----+-------+
-- | Id | Score |
-- +----+-------+
-- | 1  | 3.50  |
-- | 2  | 3.65  |
-- | 3  | 4.00  |
-- | 4  | 3.85  |
-- | 5  | 4.00  |
-- | 6  | 3.65  |
-- +----+-------+
-- For example, given the above Scores table, your query should generate
the following report (order by highest score):

-- +-------+---------+
-- | score | Rank    |
-- +-------+---------+
-- | 4.00  | 1       |
-- | 4.00  | 1       |
-- | 3.85  | 2       |
-- | 3.65  | 3       |
-- | 3.65  | 3       |
-- | 3.50  | 4       |
-- +-------+---------+
-- Important Note: For MySQL solutions, to escape reserved words used as
column names,
-- you can use an apostrophe before and after the keyword. For example
`Rank`.

-- Solution
select Score,
dense_rank() over(order by score desc) as "Rank"
from scores
```

```sql
-- Question 79
-- Table: Points

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | id            | int     |
-- | x_value       | int     |
-- | y_value       | int     |
-- +---------------+---------+
-- id is the primary key for this table.
-- Each point is represented as a 2D Dimensional (x_value, y_value).
-- Write an SQL query to report of all possible rectangles which can be
formed by any two points of the table.

-- Each row in the result contains three columns (p1, p2, area) where:

-- p1 and p2 are the id of two opposite corners of a rectangle and p1 <
p2.
-- Area of this rectangle is represented by the column area.
-- Report the query in descending order by area in case of tie in
ascending order by p1 and p2.

-- Points table:
-- +----------+-------------+-------------+
-- | id       | x_value     | y_value     |
-- +----------+-------------+-------------+
-- | 1        | 2           | 8           |
-- | 2        | 4           | 7           |
-- | 3        | 2           | 10          |
-- +----------+-------------+-------------+

-- Result table:
-- +----------+-------------+-------------+
-- | p1       | p2          | area        |
-- +----------+-------------+-------------+
-- | 2        | 3           | 6           |
-- | 1        | 2           | 2           |
-- +----------+-------------+-------------+

-- p1 should be less than p2 and area greater than 0.
-- p1 = 1 and p2 = 2, has an area equal to |2-4| * |8-7| = 2.
-- p1 = 2 and p2 = 3, has an area equal to |4-2| * |7-10| = 6.
-- p1 = 1 and p2 = 3 It's not possible because the rectangle has an area
equal to 0.

-- Solution
select p1.id as p1, p2.id as p2, abs(p1.x_value-
p2.x_value)*abs(p1.y_value-p2.y_value) as area
from points p1 cross join points p2
where p1.x_value!=p2.x_value and p1.y_value!=p2.y_value and p1.id<p2.id
order by area desc, p1, p2
```

```
-- Question 73
-- 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.
-- Table: Removals

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | post_id       | int     |
-- | remove_date   | date    |
-- +---------------+---------+
-- post_id is the primary key of this table.
-- Each row in this table indicates that some post was removed as a
result of being reported or as a result of an admin review.


-- Write an SQL query to find the average for daily percentage of posts
that got removed after being reported as spam, rounded to 2 decimal
places.

-- 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       | 2       | 2019-07-04  | view   | null   |
-- | 2       | 2       | 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       | 2019-07-03  | view   | null   |
-- | 5       | 2       | 2019-07-03  | report | racism |
-- | 5       | 5       | 2019-07-03  | view   | null   |
-- | 5       | 5       | 2019-07-03  | report | racism |
-- +---------+---------+-------------+--------+--------+

-- Removals table:
-- +---------+-------------+
-- | post_id | remove_date |
-- +---------+-------------+
-- | 2       | 2019-07-20  |
-- | 3       | 2019-07-18  |
-- +---------+-------------+

-- Result table:
-- +-----------------------+
-- | average_daily_percent |
-- +-----------------------+
-- | 75.00                 |
-- +-----------------------+
-- The percentage for 2019-07-04 is 50% because only one post of two spam
reported posts was removed.
-- The percentage for 2019-07-02 is 100% because one post was reported as
spam and it was removed.
-- The other days had no spam reports so the average is (50 + 100) / 2 =
75%
-- Note that the output is only one number and that we do not care about
the remove dates.

-- Solution
with t1 as(
select a.action_date, (count(distinct r.post_id)+0.0)/(count(distinct
a.post_id)+0.0) as result
from (select action_date, post_id
from actions
where extra = 'spam' and action = 'report') a
left join
removals r
on a.post_id = r.post_id
group by a.action_date)

select round(avg(t1.result)*100,2) as  average_daily_percent
from t1
```

```
-- Question 71
-- 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
select visited_on, sum(amount) over(order by visited_on rows 6
preceding),
round(avg(amount) over(order by visited_on rows 6 preceding),2)
from
(
      select visited_on, sum(amount) as amount
      from customer
      group by visited_on
      order by visited_on
) a
order by visited_on offset 6 rows
```

```
-- Question 76
-- Table: Scores

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | player_name   | varchar |
-- | gender        | varchar |
-- | day           | date    |
-- | score_points  | int     |
-- +---------------+---------+
-- (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.
```

```sql
-- 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
select gender, day,
sum(score_points) over(partition by gender order by day) as total
from scores
group by 1,2
order by 1,2
```

```
-- Question 70
-- In facebook, there is a follow table with two columns: followee,
follower.

-- Please write a sql query to get the amount of each follower’s follower
if he/she has one.

-- For example:

-- +-------------+------------+
-- | followee    | follower   |
-- +-------------+------------+
-- |     A       |     B      |
-- |     B       |     C      |
-- |     B       |     D      |
-- |     D       |     E      |
-- +-------------+------------+
-- should output:
-- +-------------+------------+
-- | follower    | num        |
-- +-------------+------------+
-- |     B       |  2         |
-- |     D       |  1         |
-- +-------------+------------+
-- Explaination:
-- Both B and D exist in the follower list, when as a followee, B's
follower is C and D, and D's follower is E. A does not exist in follower
list.


-- Note:
-- Followee would not follow himself/herself in all cases.
-- Please display the result in follower's alphabet order.

-- Solution
select followee as follower, count(distinct(follower)) as num
from follow
where followee = any(select follower from follow)
group by followee
order by followee
```

```sql
-- Question 89
-- Table point_2d holds the coordinates (x,y) of some unique points (more
than two) in a plane.


-- Write a query to find the shortest distance between these points
rounded to 2 decimals.


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


-- The shortest distance is 1.00 from point (-1,-1) to (-1,2). So the
output should be:


-- | shortest |
-- |----------|
-- | 1.00     |


-- Note: The longest distance among all the points are less than 10000.

-- Solution
select round(a.shortest,2) as shortest
from(
select sqrt(pow((p1.x-p2.x),2)+pow((p1.y-p2.y),2)) as shortest
from point_2d p1
cross join point_2d p2
where p1.x!=p2.x or p1.y!=p2.y
order by sqrt(pow((p1.x-p2.x),2)+pow((p1.y-p2.y),2))
limit 1) a
```

```
-- Question 53
-- 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
with t1 as(
Select c.host_id, c.host_name, c.host_points
from(
select a.match_id, a.team_id as host_id, a.team_name as host_name,
b.team_id as guest_id, b.team_name as guest_name, a.host_goals,
a.guest_goals,
case
when a.host_goals > a.guest_goals then 3
when a.host_goals = a.guest_goals then 1
else 0
end as host_points,
case
when a.host_goals < a.guest_goals then 3
when a.host_goals = a.guest_goals then 1
else 0
end as guest_points
from(
select *
from matches m
join teams t
on t.team_id = m.host_team) a
join
(select *
from matches m
join teams t
on t.team_id = m.guest_team) b
on a.match_id = b.match_id) c
union all
Select d.guest_id, d.guest_name, d.guest_points
from(
select a.match_id, a.team_id as host_id, a.team_name as host_name,
b.team_id as guest_id, b.team_name as guest_name, a.host_goals,
a.guest_goals,
case
when a.host_goals > a.guest_goals then 3
when a.host_goals = a.guest_goals then 1
else 0
end as host_points,
case
when a.host_goals < a.guest_goals then 3
when a.host_goals = a.guest_goals then 1
else 0
end as guest_points
from(
select *
from matches m
join teams t
on t.team_id = m.host_team) a
join
(select *
from matches m
join teams t
on t.team_id = m.guest_team) b
on a.match_id = b.match_id) d)

Select team_id, team_name, coalesce(total,0) as num_points
from teams t2
left join(
select host_id, host_name, sum(host_points) as total
from t1
group by host_id, host_name) e
on t2.team_id = e.host_id
order by num_points desc, team_id
```

```
-- Question 58
-- Given a table tree, id is identifier of the tree node and p_id is its
parent node's id.

-- +----+------+
-- | id | p_id |
-- +----+------+
-- | 1  | null |
-- | 2  | 1    |
-- | 3  | 1    |
-- | 4  | 2    |
-- | 5  | 2    |
-- +----+------+
-- Each node in the tree can be one of three types:
-- Leaf: if the node is a leaf node.
-- Root: if the node is the root of the tree.
-- Inner: If the node is neither a leaf node nor a root node.


-- Write a query to print the node id and the type of the node. Sort your
output by the node id. The result for the above sample is:


-- +----+------+
-- | id | Type |
-- +----+------+
-- | 1  | Root |
-- | 2  | Inner|
-- | 3  | Leaf |
-- | 4  | Leaf |
-- | 5  | Leaf |
-- +----+------+


-- Explanation



-- Node '1' is root node, because its parent node is NULL and it has
child node '2' and '3'.
-- Node '2' is inner node, because it has parent node '1' and child node
'4' and '5'.
-- Node '3', '4' and '5' is Leaf node, because they have parent node and
they don't have child node.

-- And here is the image of the sample tree as below:


--                 1
--               /   \
--           2       3
--         /   \
--       4       5
-- Note

-- If there is only one node on the tree, you only need to output its
root attributes.
```

```sql
-- Solution
select id,
case when p_id is null then 'Root'
when id not in (select p_id from tree where p_id is not null group by
p_id) then 'Leaf'
else 'Inner'
end as Type
from tree
order by id
```

```
-- Question 64
-- Table: Books

-- +----------------+---------+
-- | Column Name    | Type    |
-- +----------------+---------+
-- | book_id        | int     |
-- | name           | varchar |
-- | available_from | date    |
-- +----------------+---------+
-- book_id is the primary key of this table.
-- Table: Orders

-- +----------------+---------+
-- | Column Name    | Type    |
-- +----------------+---------+
-- | order_id       | int     |
-- | book_id        | int     |
-- | quantity       | int     |
-- | dispatch_date  | date    |
-- +----------------+---------+
-- order_id is the primary key of this table.
-- book_id is a foreign key to the Books table.


-- Write an SQL query that reports the books that have sold less than 10
copies in the last year, excluding books that have been available for
less than 1 month from today. Assume today is 2019-06-23.

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

-- Books table:
-- +---------+--------------------+----------------+
-- | book_id | name               | available_from |
-- +---------+--------------------+----------------+
-- | 1       | "Kalila And Demna" | 2010-01-01     |
-- | 2       | "28 Letters"       | 2012-05-12     |
-- | 3       | "The Hobbit"       | 2019-06-10     |
-- | 4       | "13 Reasons Why"   | 2019-06-01     |
-- | 5       | "The Hunger Games" | 2008-09-21     |
-- +---------+--------------------+----------------+

-- Orders table:
-- +----------+---------+----------+----------------+
-- | order_id | book_id | quantity | dispatch_date |
-- +----------+---------+----------+----------------+
-- | 1        | 1       | 2        | 2018-07-26     |
-- | 2        | 1       | 1        | 2018-11-05     |
-- | 3        | 3       | 8        | 2019-06-11     |
-- | 4        | 4       | 6        | 2019-06-05     |
-- | 5        | 4       | 5        | 2019-06-20     |
-- | 6        | 5       | 9        | 2009-02-02     |
-- | 7        | 5       | 8        | 2010-04-13     |
-- +----------+---------+----------+----------------+

-- Result table:
-- +-----------+--------------------+
-- | book_id   | name               |
-- +-----------+--------------------+
-- | 1         | "Kalila And Demna" |
-- | 2         | "28 Letters"       |
-- | 5         | "The Hunger Games" |
-- +-----------+--------------------+

-- Solution
select b.book_id, name
from
(select *
from books
where available_from < '2019-05-23') b
left join
(select *
from orders
where dispatch_date > '2018-06-23') a
on a.book_id = b.book_id
group by b.book_id, name
having coalesce(sum(quantity),0)<10
```

```
-- Question 88
-- Table: Candidate

-- +-----+---------+
-- | id  | Name    |
-- +-----+---------+
-- | 1   | A       |
-- | 2   | B       |
-- | 3   | C       |
-- | 4   | D       |
-- | 5   | E       |
-- +-----+---------+
-- Table: Vote

-- +-----+--------------+
-- | id  | CandidateId  |
-- +-----+--------------+
-- | 1   |     2        |
-- | 2   |     4        |
-- | 3   |     3        |
-- | 4   |     2        |
-- | 5   |     5        |
-- +-----+--------------+
-- id is the auto-increment primary key,
-- CandidateId is the id appeared in Candidate table.
-- Write a sql to find the name of the winning candidate, the above
example will return the winner B.

-- +------+
-- | Name |
-- +------+
-- | B    |
-- +------+
-- Notes:

-- You may assume there is no tie, in other words there will be only one
winning candidate

-- Solution
with t1 as (
select *, rank() over(order by b.votes desc) as rk
from candidate c
join
(select candidateid, count(*) as votes
from vote
group by candidateid) b
on c.id = b.candidateid)

select t1.name
from t1
where t1.rk=1
```