```
-- Question 108
-- Given two tables as below, write a query to display the comparison
result (higher/lower/same) of the
-- average salary of employees in a department to the company's average
salary.


-- Table: salary
-- | id | employee_id | amount | pay_date   |
-- |----|-------------|--------|------------|
-- | 1  | 1           | 9000   | 2017-03-31 |
-- | 2  | 2           | 6000   | 2017-03-31 |
-- | 3  | 3           | 10000  | 2017-03-31 |
-- | 4  | 1           | 7000   | 2017-02-28 |
-- | 5  | 2           | 6000   | 2017-02-28 |
-- | 6  | 3           | 8000   | 2017-02-28 |


-- The employee_id column refers to the employee_id in the following
table employee.


-- | employee_id | department_id |
-- |-------------|---------------|
-- | 1           | 1             |
-- | 2           | 2             |
-- | 3           | 2             |


-- So for the sample data above, the result is:


-- | pay_month | department_id | comparison  |
-- |-----------|---------------|-------------|
-- | 2017-03   | 1             | higher      |
-- | 2017-03   | 2             | lower       |
-- | 2017-02   | 1             | same        |
-- | 2017-02   | 2             | same        |


-- Explanation


-- In March, the company's average salary is (9000+6000+10000)/3 =
8333.33...


-- The average salary for department '1' is 9000, which is the salary of
employee_id '1' since there is only one employee in this department. So
the comparison result is 'higher' since 9000 > 8333.33 obviously.


-- The average salary of department '2' is (6000 + 10000)/2 = 8000, which
is the average of employee_id '2' and '3'. So the comparison result is
'lower' since 8000 < 8333.33.
```

```
-- With he same formula for the average salary comparison in February,
the result is 'same' since both the department '1' and '2' have the same
average salary with the company, which is 7000.

-- Solution
with t1 as(
select date_format(pay_date,'%Y-%m') as pay_month, department_id,
avg(amount) over(partition by month(pay_date),department_id) as dept_avg,
avg(amount) over(partition by month(pay_date)) as comp_avg
from salary s join employee e
using (employee_id))

select distinct pay_month, department_id,
case when dept_avg>comp_avg then "higher"
when dept_avg = comp_avg then "same"
else "lower"
end as comparison
from t1
order by 1 desc
```

```
-- Question 102
-- The Employee table holds the salary information in a year.

-- Write a SQL to get the cumulative sum of an employee's salary over a
period of 3 months but exclude the most recent month.

-- The result should be displayed by 'Id' ascending, and then by 'Month'
descending.

-- Example
-- Input

-- | Id | Month | Salary |
-- |----|-------|--------|
-- | 1  | 1     | 20     |
-- | 2  | 1     | 20     |
-- | 1  | 2     | 30     |
-- | 2  | 2     | 30     |
-- | 3  | 2     | 40     |
-- | 1  | 3     | 40     |
-- | 3  | 3     | 60     |
-- | 1  | 4     | 60     |
-- | 3  | 4     | 70     |
-- Output

-- | Id | Month | Salary |
-- |----|-------|--------|
-- | 1  | 3     | 90     |
-- | 1  | 2     | 50     |
-- | 1  | 1     | 20     |
-- | 2  | 1     | 20     |
-- | 3  | 3     | 100    |
-- | 3  | 2     | 40     |


-- Explanation
-- Employee '1' has 3 salary records for the following 3 months except
the most recent month '4': salary 40 for month '3', 30 for month '2' and
20 for month '1'
-- So the cumulative sum of salary of this employee over 3 months is
90(40+30+20), 50(30+20) and 20 respectively.

-- | Id | Month | Salary |
-- |----|-------|--------|
-- | 1  | 3     | 90     |
-- | 1  | 2     | 50     |
-- | 1  | 1     | 20     |
-- Employee '2' only has one salary record (month '1') except its most
recent month '2'.
-- | Id | Month | Salary |
-- |----|-------|--------|
-- | 2  | 1     | 20     |


-- Employ '3' has two salary records except its most recent pay month
'4': month '3' with 60 and month '2' with 40. So the cumulative salary is
as following.
-- | Id | Month | Salary |
```

```
-- |----|-------|--------|
-- | 3  | 3     | 100    |
-- | 3  | 2     | 40     |

-- Solution
with t1 as(
select *, max(month) over(partition by id) as recent_month
from employee)

select id, month, sum(salary) over(partition by id order by month rows
between 2 preceding and current row) as salary
from t1
where month<recent_month
order by 1, 2 desc
```

```
-- Question 14
-- The Employee table holds all employees. Every employee has an Id, and
there is also a column for the department Id.

-- +----+-------+--------+--------------+
-- | Id | Name  | Salary | DepartmentId |
-- +----+-------+--------+--------------+
-- | 1  | Joe   | 85000  | 1            |
-- | 2  | Henry | 80000  | 2            |
-- | 3  | Sam   | 60000  | 2            |
-- | 4  | Max   | 90000  | 1            |
-- | 5  | Janet | 69000  | 1            |
-- | 6  | Randy | 85000  | 1            |
-- | 7  | Will  | 70000  | 1            |
-- +----+-------+--------+--------------+
-- The Department table holds all departments of the company.

-- +----+----------+
-- | Id | Name     |
-- +----+----------+
-- | 1  | IT       |
-- | 2  | Sales    |
-- +----+----------+
-- Write a SQL query to find employees who earn the top three salaries in
each of the department. 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         | Randy    | 85000  |
-- | IT         | Joe      | 85000  |
-- | IT         | Will     | 70000  |
-- | Sales      | Henry    | 80000  |
-- | Sales      | Sam      | 60000  |
-- +------------+----------+--------+
-- Explanation:

-- In IT department, Max earns the highest salary, both Randy and Joe
earn the second highest salary,
-- and Will earns the third highest salary.
-- There are only two employees in the Sales department,
-- Henry earns the highest salary while Sam earns the second highest
salary.

-- Solution
select a.department, a.employee, a.salary
from (
select d.name as department, e.name as employee, salary,
    dense_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<4
```

```
-- Question 107
-- The Numbers table keeps the value of number and its frequency.

-- +----------+-------------+
-- |  Number  |  Frequency  |
-- +----------+-------------|
-- |  0       |  7          |
-- |  1       |  1          |
-- |  2       |  3          |
-- |  3       |  1          |
-- +----------+-------------+
-- In this table, the numbers are 0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 3, so
the median is (0 + 0) / 2 = 0.

-- +--------+
-- | median |
-- +--------|
-- | 0.0000 |
-- +--------+
-- Write a query to find the median of all numbers and name the result as
median.

-- Solution
with t1 as(
select *,
sum(frequency) over(order by number) as cum_sum, (sum(frequency)
over())/2 as middle
from numbers)

select avg(number) as median
from t1
where middle between (cum_sum - frequency) and cum_sum
```

```
-- Question 106
-- Table: Student

-- +----------------------+---------+
-- | Column Name          | Type    |
-- +----------------------+---------+
-- | student_id           | int     |
-- | student_name         | varchar |
-- +----------------------+---------+
-- student_id is the primary key for this table.
-- student_name is the name of the student.


-- Table: Exam

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | exam_id       | int     |
-- | student_id    | int     |
-- | score         | int     |
-- +---------------+---------+
-- (exam_id, student_id) is the primary key for this table.
-- Student with student_id got score points in exam with id exam_id.


-- A "quite" student is the one who took at least one exam and didn't
score neither the high score nor the low score.

-- Write an SQL query to report the students (student_id, student_name)
being "quiet" in ALL exams.

-- Don't return the student who has never taken any exam. Return the
result table ordered by student_id.

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



-- Student table:
-- +-------------+---------------+
-- | student_id  | student_name  |
-- +-------------+---------------+
-- | 1           | Daniel        |
-- | 2           | Jade          |
-- | 3           | Stella        |
-- | 4           | Jonathan      |
-- | 5           | Will          |
-- +-------------+---------------+

-- Exam table:
-- +------------+--------------+-----------+
-- | exam_id    | student_id   | score     |
-- +------------+--------------+-----------+
-- | 10         |     1        |    70     |
-- | 10         |     2        |    80     |
-- | 10         |     3        |    90     |
-- | 20         |     1        |    80     |
```

```
-- | 30          |     1         |    70     |
-- | 30          |     3         |    80     |
-- | 30          |     4         |    90     |
-- | 40          |     1         |    60     |
-- | 40          |     2         |    70     |
-- | 40          |     4         |    80     |
-- +-------------+---------------+-----------+

-- Result table:
-- +-------------+---------------+
-- | student_id  | student_name  |
-- +-------------+---------------+
-- | 2           | Jade          |
-- +-------------+---------------+

-- For exam 1: Student 1 and 3 hold the lowest and high score
respectively.
-- For exam 2: Student 1 hold both highest and lowest score.
-- For exam 3 and 4: Studnet 1 and 4 hold the lowest and high score
respectively.
-- Student 2 and 5 have never got the highest or lowest in any of the
exam.
-- Since student 5 is not taking any exam, he is excluded from the
result.
-- So, we only return the information of Student 2.

-- Solution
with t1 as(
select student_id
from
(select *,
min(score) over(partition by exam_id) as least,
max(score) over(partition by exam_id) as most
from exam) a
where least = score or most = score)


select distinct student_id, student_name
from exam join student
using (student_id)
where student_id != all(select student_id from t1)
order by 1
```

```
-- Question 111
-- 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.


-- We define the install date of a player to be the first login day of
that player.

-- We also define day 1 retention of some date X to be the number of
players whose install date is X and they logged back in on the day right
after X, divided by the number of players whose install date is X,
rounded to 2 decimal places.

-- Write an SQL query that reports for each install date, the number of
players that installed the game on that day and the day 1 retention.

-- 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         | 2016-03-01 | 0            |
-- | 3         | 4         | 2016-07-03 | 5            |
-- +-----------+-----------+------------+--------------+

-- Result table:
-- +------------+----------+----------------+
-- | install_dt | installs | Day1_retention |
-- +------------+----------+----------------+
-- | 2016-03-01 | 2        | 0.50           |
-- | 2017-06-25 | 1        | 0.00           |
-- +------------+----------+----------------+
-- Player 1 and 3 installed the game on 2016-03-01 but only player 1
logged back in on 2016-03-02 so the
-- day 1 retention of 2016-03-01 is 1 / 2 = 0.50
-- Player 2 installed the game on 2017-06-25 but didn't log back in on
2017-06-26 so the day 1 retention of 2017-06-25 is 0 / 1 = 0.00

-- Solution
with t1 as(
select *,
row_number() over(partition by player_id order by event_date) as rnk,
```

```
min(event_date) over(partition by player_id) as install_dt,
lead(event_date,1) over(partition by player_id order by event_date) as
nxt
from Activity)

select distinct install_dt,
count(distinct player_id) as installs,
round(sum(case when nxt=event_date+1 then 1 else 0 end)/count(distinct
player_id),2) as Day1_retention
from t1
where rnk = 1
group by 1
order by 1
```

```
-- Question 99
-- X city built a new stadium, each day many people visit it and the
stats are saved as these columns: id, visit_date, people

-- Please write a query to display the records which have 3 or more
consecutive rows and the amount of people more than 100(inclusive).

-- For example, the table stadium:
-- +------+------------+-----------+
-- | id   | visit_date | people    |
-- +------+------------+-----------+
-- | 1    | 2017-01-01 | 10        |
-- | 2    | 2017-01-02 | 109       |
-- | 3    | 2017-01-03 | 150       |
-- | 4    | 2017-01-04 | 99        |
-- | 5    | 2017-01-05 | 145       |
-- | 6    | 2017-01-06 | 1455      |
-- | 7    | 2017-01-07 | 199       |
-- | 8    | 2017-01-08 | 188       |
-- +------+------------+-----------+
-- For the sample data above, the output is:

-- +------+------------+-----------+
-- | id   | visit_date | people    |
-- +------+------------+-----------+
-- | 5    | 2017-01-05 | 145       |
-- | 6    | 2017-01-06 | 1455      |
-- | 7    | 2017-01-07 | 199       |
-- | 8    | 2017-01-08 | 188       |
-- +------+------------+-----------+
-- Note:
-- Each day only have one row record, and the dates are increasing with
id increasing.

-- Solution
WITH t1 AS (
            SELECT id,
                   visit_date,
                   people,
                   id - ROW_NUMBER() OVER(ORDER BY visit_date) AS dates
              FROM stadium
            WHERE people >= 100)

SELECT t1.id,
       t1.visit_date,
       t1people
FROM t1
LEFT JOIN (
            SELECT dates,
                   COUNT(*) as total
              FROM t1
            GROUP BY dates) AS b
USING (dates)
WHERE b.total > 2
```

```
-- Question 103
-- 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, whether the brand of the
second item (by date) they sold is their favorite brand. If a user sold
less than two items, report the answer for that user as no.

-- It is guaranteed that no seller sold more than one item on a day.

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

-- Users table:
-- +---------+------------+----------------+
-- | user_id | join_date  | favorite_brand |
-- +---------+------------+----------------+
-- | 1       | 2019-01-01 | Lenovo         |
-- | 2       | 2019-02-09 | Samsung        |
-- | 3       | 2019-01-19 | LG             |
-- | 4       | 2019-05-21 | HP             |
-- +---------+------------+----------------+

-- Orders table:
-- +----------+------------+---------+----------+-----------+
```

```
-- | order_id | order_date | item_id | buyer_id | seller_id |
-- +----------+------------+---------+----------+-----------+
-- | 1        | 2019-08-01 | 4       | 1        | 2         |
-- | 2        | 2019-08-02 | 2       | 1        | 3         |
-- | 3        | 2019-08-03 | 3       | 2        | 3         |
-- | 4        | 2019-08-04 | 1       | 4        | 2         |
-- | 5        | 2019-08-04 | 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:
-- +-----------+--------------------+
-- | seller_id | 2nd_item_fav_brand |
-- +-----------+--------------------+
-- | 1         | no                 |
-- | 2         | yes                |
-- | 3         | yes                |
-- | 4         | no                 |
-- +-----------+--------------------+

-- The answer for the user with id 1 is no because they sold nothing.
-- The answer for the users with id 2 and 3 is yes because the brands of
their second sold items are their favorite brands.
-- The answer for the user with id 4 is no because the brand of their
second sold item is not their favorite brand.

-- Solution
with t1 as(
select user_id,
case when favorite_brand = item_brand then "yes"
else "no"
end as 2nd_item_fav_brand
from users u left join
(select o.item_id, seller_id, item_brand, rank() over(partition by
seller_id order by order_date) as rk
from orders o join items i
using (item_id)) a
on u.user_id = a.seller_id
where a.rk = 2)

select u.user_id as seller_id, coalesce(2nd_item_fav_brand,"no") as
2nd_item_fav_brand
from users u left join t1
using(user_id)
```

```
-- Question 105
-- The Employee table holds all employees. The employee table has three
columns: Employee Id, Company Name, and Salary.

-- +-----+------------+--------+
-- |Id   | Company    | Salary |
-- +-----+------------+--------+
-- |1    | A          | 2341   |
-- |2    | A          | 341    |
-- |3    | A          | 15     |
-- |4    | A          | 15314  |
-- |5    | A          | 451    |
-- |6    | A          | 513    |
-- |7    | B          | 15     |
-- |8    | B          | 13     |
-- |9    | B          | 1154   |
-- |10   | B          | 1345   |
-- |11   | B          | 1221   |
-- |12   | B          | 234    |
-- |13   | C          | 2345   |
-- |14   | C          | 2645   |
-- |15   | C          | 2645   |
-- |16   | C          | 2652   |
-- |17   | C          | 65     |
-- +-----+------------+--------+
-- Write a SQL query to find the median salary of each company. Bonus
points if you can solve it without using any built-in SQL functions.

-- +-----+------------+--------+
-- |Id   | Company    | Salary |
-- +-----+------------+--------+
-- |5    | A          | 451    |
-- |6    | A          | 513    |
-- |12   | B          | 234    |
-- |9    | B          | 1154   |
-- |14   | C          | 2645   |
-- +-----+------------+--------+

-- Solution
select id, company, salary
from
(select *,
row_number() over(partition by company order by salary) as rn,
count(*) over(partition by company) as cnt
from employee) a
where rn between cnt/2 and cnt/2+1
```

```
--Question 101
-- Table: Visits

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | user_id       | int     |
-- | visit_date    | date    |
-- +---------------+---------+
-- (user_id, visit_date) is the primary key for this table.
-- Each row of this table indicates that user_id has visited the bank in
visit_date.


-- Table: Transactions

-- +------------------+---------+
-- | Column Name      | Type    |
-- +------------------+---------+
-- | user_id          | int     |
-- | transaction_date | date    |
-- | amount           | int     |
-- +------------------+---------+
-- There is no primary key for this table, it may contain duplicates.
-- Each row of this table indicates that user_id has done a transaction
of amount in transaction_date.
-- It is guaranteed that the user has visited the bank in the
transaction_date.(i.e The Visits table contains (user_id,
transaction_date) in one row)


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

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

-- The result table will contain two columns:

-- transactions_count which is the number of transactions done in one
visit.
-- visits_count which is the corresponding number of users who did
transactions_count in one visit to the bank.
-- transactions_count should take all values from 0 to
max(transactions_count) done by one or more users.

-- Order the result table by transactions_count.

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

-- Visits table:
-- +---------+------------+
-- | user_id | visit_date |
-- +---------+------------+
-- | 1       | 2020-01-01 |
-- | 2       | 2020-01-02 |
```

```
-- | 12      | 2020-01-01 |
-- | 19      | 2020-01-03 |
-- | 1       | 2020-01-02 |
-- | 2       | 2020-01-03 |
-- | 1       | 2020-01-04 |
-- | 7       | 2020-01-11 |
-- | 9       | 2020-01-25 |
-- | 8       | 2020-01-28 |
-- +---------+------------+
-- Transactions table:
-- +---------+------------------+--------+
-- | user_id | transaction_date | amount |
-- +---------+------------------+--------+
-- | 1       | 2020-01-02       | 120    |
-- | 2       | 2020-01-03       | 22     |
-- | 7       | 2020-01-11       | 232    |
-- | 1       | 2020-01-04       | 7      |
-- | 9       | 2020-01-25       | 33     |
-- | 9       | 2020-01-25       | 66     |
-- | 8       | 2020-01-28       | 1      |
-- | 9       | 2020-01-25       | 99     |
-- +---------+------------------+--------+
-- Result table:
-- +--------------------+--------------+
-- | transactions_count | visits_count |
-- +--------------------+--------------+
-- | 0                  | 4            |
-- | 1                  | 5            |
-- | 2                  | 0            |
-- | 3                  | 1            |
-- +--------------------+--------------+
-- * For transactions_count = 0, The visits (1, "2020-01-01"), (2, "2020-
01-02"), (12, "2020-01-01") and (19, "2020-01-03") did no transactions so
visits_count = 4.
-- * For transactions_count = 1, The visits (2, "2020-01-03"), (7, "2020-
01-11"), (8, "2020-01-28"), (1, "2020-01-02") and (1, "2020-01-04") did
one transaction so visits_count = 5.
-- * For transactions_count = 2, No customers visited the bank and did
two transactions so visits_count = 0.
-- * For transactions_count = 3, The visit (9, "2020-01-25") did three
transactions so visits_count = 1.
-- * For transactions_count >= 4, No customers visited the bank and did
more than three transactions so we will stop at transactions_count = 3

-- Solution
WITH RECURSIVE t1 AS(
                    SELECT visit_date,
                           COALESCE(num_visits,0) as num_visits,
                           COALESCE(num_trans,0) as num_trans
                    FROM ((
                          SELECT visit_date, user_id, COUNT(*) as
num_visits
                          FROM visits
                          GROUP BY 1, 2) AS a
                         LEFT JOIN
                          (
                           SELECT transaction_date,
                                  user_id,
```

```
                                count(*) as num_trans
                           FROM transactions
                          GROUP BY 1, 2) AS b
                         ON a.visit_date = b.transaction_date and
a.user_id = b.user_id)
                      ),

              t2 AS (
                      SELECT MAX(num_trans) as trans
                        FROM t1
                      UNION ALL
                      SELECT trans-1
                        FROM t2
                      WHERE trans >= 1)

SELECT trans as transactions_count,
       COALESCE(visits_count,0) as visits_count
  FROM t2 LEFT JOIN (
                    SELECT num_trans as transactions_count,
COALESCE(COUNT(*),0) as visits_count
                    FROM t1
                    GROUP BY 1
                    ORDER BY 1) AS a
ON a.transactions_count = t2.trans
ORDER BY 1
```

```
-- Question 104
-- Table: Failed

-- +--------------+---------+
-- | Column Name  | Type    |
-- +--------------+---------+
-- | fail_date    | date    |
-- +--------------+---------+
-- Primary key for this table is fail_date.
-- Failed table contains the days of failed tasks.
-- Table: Succeeded

-- +--------------+---------+
-- | Column Name  | Type    |
-- +--------------+---------+
-- | success_date | date    |
-- +--------------+---------+
-- Primary key for this table is success_date.
-- Succeeded table contains the days of succeeded tasks.


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

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

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

-- Order result by start_date.

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

-- Failed table:
-- +-------------------+
-- | fail_date         |
-- +-------------------+
-- | 2018-12-28        |
-- | 2018-12-29        |
-- | 2019-01-04        |
-- | 2019-01-05        |
-- +-------------------+

-- Succeeded table:
-- +-------------------+
-- | success_date      |
-- +-------------------+
-- | 2018-12-30        |
-- | 2018-12-31        |
-- | 2019-01-01        |
-- | 2019-01-02        |
-- | 2019-01-03        |
-- | 2019-01-06        |
-- +-------------------+
```

```
-- Result table:
-- +--------------+--------------+--------------+
-- | period_state | start_date   | end_date     |
-- +--------------+--------------+--------------+
-- | succeeded    | 2019-01-01   | 2019-01-03   |
-- | failed       | 2019-01-04   | 2019-01-05   |
-- | succeeded    | 2019-01-06   | 2019-01-06   |
-- +--------------+--------------+--------------+

-- The report ignored the system state in 2018 as we care about the
system in the period 2019-01-01 to 2019-12-31.
-- From 2019-01-01 to 2019-01-03 all tasks succeeded and the system state
was "succeeded".
-- From 2019-01-04 to 2019-01-05 all tasks failed and system state was
"failed".
-- From 2019-01-06 to 2019-01-06 all tasks succeeded and system state was
"succeeded".

-- Solution
with t1 as(
select min(success_date) as start_date, max(success_date) as end_date,
state
from(
select *, date_sub(success_date, interval row_number() over(order by
success_date) day) as diff, 1 as state
from succeeded
where success_date between "2019-01-01" and "2019-12-31") a
group by diff),

t2 as(
select min(fail_date) as start_date, max(fail_date) as end_date, state
from(
select *, date_sub(fail_date, interval row_number() over(order by
fail_date) day) as diff, 0 as state
from failed
where fail_date between "2019-01-01" and "2019-12-31") b
group by diff)


select
case when c.state = 1 then "succeeded"
else "failed"
end as period_state,start_date, end_date
from(
select *
from t1
union all
select *
from t2) c
order by start_date
```

```
-- Question 112
-- Table: Orders

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | order_id      | int     |
-- | customer_id   | int     |
-- | order_date    | date    |
-- | item_id       | varchar |
-- | quantity      | int     |
-- +---------------+---------+
-- (ordered_id, item_id) is the primary key for this table.
-- This table contains information of the orders placed.
-- order_date is the date when item_id was ordered by the customer with
id customer_id.


-- Table: Items

-- +---------------------+---------+
-- | Column Name         | Type    |
-- +---------------------+---------+
-- | item_id             | varchar |
-- | item_name           | varchar |
-- | item_category       | varchar |
-- +---------------------+---------+
-- item_id is the primary key for this table.
-- item_name is the name of the item.
-- item_category is the category of the item.


-- You are the business owner and would like to obtain a sales report for
category items and day of the week.

-- Write an SQL query to report how many units in each category have been
ordered on each day of the week.

-- Return the result table ordered by category.

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



-- Orders table:
-- +------------+--------------+-------------+--------------+------------
-+
-- | order_id   | customer_id  | order_date  | item_id      | quantity
|
-- +------------+--------------+-------------+--------------+------------
-+
-- | 1          | 1            | 2020-06-01  | 1            | 10
|
-- | 2          | 1            | 2020-06-08  | 2            | 10
|
-- | 3          | 2            | 2020-06-02  | 1            | 5
|
```

```
-- | 4           | 3             | 2020-06-03  | 3             | 5
|
-- | 5           | 4             | 2020-06-04  | 4             | 1
|
-- | 6           | 4             | 2020-06-05  | 5             | 5
|
-- | 7           | 5             | 2020-06-05  | 1             | 10
|
-- | 8           | 5             | 2020-06-14  | 4             | 5
|
-- | 9           | 5             | 2020-06-21  | 3             | 5
|
-- +-------------+---------------+-------------+---------------+------------
-+

-- Items table:
-- +-------------+-----------------+---------------+
-- | item_id     | item_name       | item_category |
-- +-------------+-----------------+---------------+
-- | 1           | LC Alg. Book    | Book          |
-- | 2           | LC DB. Book     | Book          |
-- | 3           | LC SmarthPhone  | Phone         |
-- | 4           | LC Phone 2020   | Phone         |
-- | 5           | LC SmartGlass   | Glasses       |
-- | 6           | LC T-Shirt XL   | T-Shirt       |
-- +-------------+-----------------+---------------+

-- Result table:
-- +-------------+-----------+-----------+-----------+-----------+----------
---+-----------+-----------+
-- | Category    | Monday    | Tuesday   | Wednesday | Thursday  | Friday
| Saturday  | Sunday    |
-- +-------------+-----------+-----------+-----------+-----------+----------
---+-----------+-----------+
-- | Book        | 20        | 5         | 0         | 0         | 10
| 0         | 0         |
-- | Glasses     | 0         | 0         | 0         | 0         | 5
| 0         | 0         |
-- | Phone       | 0         | 0         | 5         | 1         | 0
| 0         | 10        |
-- | T-Shirt     | 0         | 0         | 0         | 0         | 0
| 0         | 0         |
-- +-------------+-----------+-----------+-----------+-----------+----------
---+-----------+-----------+
-- On Monday (2020-06-01, 2020-06-08) were sold a total of 20 units (10 +
10) in the category Book (ids: 1, 2).
-- On Tuesday (2020-06-02) were sold a total of 5 units  in the category
Book (ids: 1, 2).
-- On Wednesday (2020-06-03) were sold a total of 5 units in the category
Phone (ids: 3, 4).
-- On Thursday (2020-06-04) were sold a total of 1 unit in the category
Phone (ids: 3, 4).
-- On Friday (2020-06-05) were sold 10 units in the category Book (ids:
1, 2) and 5 units in Glasses (ids: 5).
-- On Saturday there are no items sold.
-- On Sunday (2020-06-14, 2020-06-21) were sold a total of 10 units (5
+5) in the category Phone (ids: 3, 4).
-- There are no sales of T-Shirt.
```

```
-- Solution
with t1 as(
select distinct item_category,
case when dayname(order_date)='Monday' then sum(quantity) over(partition
by item_category,dayname(order_date)) else 0 end as Monday,
Case when dayname(order_date)='Tuesday' then sum(quantity) over(partition
by item_category,dayname(order_date)) else 0 end as Tuesday,
Case when dayname(order_date)='Wednesday' then sum(quantity)
over(partition by item_category,dayname(order_date)) else 0 end as
Wednesday,
Case when dayname(order_date)='Thursday' then sum(quantity)
over(partition by item_category,dayname(order_date)) else 0 end as
Thursday,
Case when dayname(order_date)='Friday' then sum(quantity) over(partition
by item_category,dayname(order_date)) else 0 end as Friday,
Case when dayname(order_date)='Saturday' then sum(quantity)
over(partition by item_category,dayname(order_date)) else 0 end as
Saturday,
Case when dayname(order_date)='Sunday' then sum(quantity) over(partition
by item_category,dayname(order_date)) else 0 end as Sunday
from orders o
right join items i
using (item_id))

select item_category as category, sum(Monday) as Monday, sum(Tuesday) as
Tuesday, sum(Wednesday) Wednesday, sum(Thursday) Thursday,
sum(Friday) Friday, sum(Saturday) Saturday, sum(Sunday) Sunday
from t1
group by item_category
```

```
-- Question 105
-- A U.S graduate school has students from Asia, Europe and America. The
students' location information are stored in table student as below.


-- | name   | continent |
-- |--------|-----------|
-- | Jack   | America   |
-- | Pascal | Europe    |
-- | Xi     | Asia      |
-- | Jane   | America   |


-- Pivot the continent column in this table so that each name is sorted
alphabetically and displayed underneath its corresponding continent. The
output headers should be America, Asia and Europe respectively. It is
guaranteed that the student number from America is no less than either
Asia or Europe.


-- For the sample input, the output is:


-- | America | Asia | Europe |
-- |---------|------|--------|
-- | Jack    | Xi   | Pascal |
-- | Jane    |      |        |

-- Solution
select min(case when continent = 'America' then name end) as America,
min(case when continent = 'Asia' then name end) as Asia,
min(case when continent = 'Europe' then name end) as Europe
from
(select *, row_number() over(partition by continent order by name) as rn
from student) a
group by rn
```

```
-- Question 114
-- Table: Product

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


-- Table: Sales

-- +----------------------+---------+
-- | Column Name          | Type    |
-- +----------------------+---------+
-- | product_id           | int     |
-- | period_start         | varchar |
-- | period_end           | date    |
-- | average_daily_sales | int     |
-- +----------------------+---------+
-- product_id is the primary key for this table.
-- period_start and period_end indicates the start and end date for sales
period, both dates are inclusive.
-- The average_daily_sales column holds the average daily sales amount of
the items for the period.

-- Write an SQL query to report the Total sales amount of each item for
each year, with corresponding product name, product_id, product_name and
report_year.

-- Dates of the sales years are between 2018 to 2020. Return the result
table ordered by product_id and report_year.

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


-- Product table:
-- +------------+--------------+
-- | product_id | product_name |
-- +------------+--------------+
-- | 1          | LC Phone     |
-- | 2          | LC T-Shirt   |
-- | 3          | LC Keychain  |
-- +------------+--------------+

-- Sales table:
-- +------------+--------------+-------------+---------------------+
-- | product_id | period_start | period_end  | average_daily_sales |
-- +------------+--------------+-------------+---------------------+
-- | 1          | 2019-01-25   | 2019-02-28  | 100                 |
-- | 2          | 2018-12-01   | 2020-01-01  | 10                  |
-- | 3          | 2019-12-01   | 2020-01-31  | 1                   |
-- +------------+--------------+-------------+---------------------+

-- Result table:
```

```
-- +------------+--------------+-------------+--------------+
-- | product_id | product_name | report_year | total_amount |
-- +------------+--------------+-------------+--------------+
-- | 1          | LC Phone     |    2019     | 3500         |
-- | 2          | LC T-Shirt   |    2018     | 310          |
-- | 2          | LC T-Shirt   |    2019     | 3650         |
-- | 2          | LC T-Shirt   |    2020     | 10           |
-- | 3          | LC Keychain  |    2019     | 31           |
-- | 3          | LC Keychain  |    2020     | 31           |
-- +------------+--------------+-------------+--------------+
-- LC Phone was sold for the period of 2019-01-25 to 2019-02-28, and
there are 35 days for this period. Total amount 35*100 = 3500.
-- LC T-shirt was sold for the period of 2018-12-01 to 2020-01-01, and
there are 31, 365, 1 days for years 2018, 2019 and 2020 respectively.
-- LC Keychain was sold for the period of 2019-12-01 to 2020-01-31, and
there are 31, 31 days for years 2019 and 2020 respectively.

-- Solution
SELECT
    b.product_id,
    a.product_name,
    a.yr AS report_year,
    CASE
        WHEN YEAR(b.period_start)=YEAR(b.period_end) AND
a.yr=YEAR(b.period_start) THEN DATEDIFF(b.period_end,b.period_start)+1
        WHEN a.yr=YEAR(b.period_start) THEN
DATEDIFF(DATE_FORMAT(b.period_start,'%Y-12-31'),b.period_start)+1
        WHEN a.yr=YEAR(b.period_end) THEN DAYOFYEAR(b.period_end)
        WHEN a.yr>YEAR(b.period_start) AND a.yr<YEAR(b.period_end) THEN
365
        ELSE 0
    END * average_daily_sales AS total_amount
FROM
    (SELECT product_id,product_name,'2018' AS yr FROM Product
    UNION
    SELECT product_id,product_name,'2019' AS yr FROM Product
    UNION
    SELECT product_id,product_name,'2020' AS yr FROM Product) a
    JOIN
    Sales b
    ON a.product_id=b.product_id
HAVING total_amount > 0
ORDER BY b.product_id,a.yr
```

```
-- Question 109
-- Table: Players

-- +-------------+-------+
-- | Column Name | Type  |
-- +-------------+-------+
-- | player_id   | int   |
-- | group_id    | int   |
-- +-------------+-------+
-- player_id is the primary key of this table.
-- Each row of this table indicates the group of each player.
-- Table: Matches

-- +---------------+---------+
-- | Column Name   | Type    |
-- +---------------+---------+
-- | match_id      | int     |
-- | first_player  | int     |
-- | second_player | int     |
-- | first_score   | int     |
-- | second_score  | int     |
-- +---------------+---------+
-- match_id is the primary key of this table.
-- Each row is a record of a match, first_player and second_player
contain the player_id of each match.
-- first_score and second_score contain the number of points of the
first_player and second_player respectively.
-- You may assume that, in each match, players belongs to the same group.


-- The winner in each group is the player who scored the maximum total
points within the group. In the case of a tie,
-- the lowest player_id wins.

-- Write an SQL query to find the winner in each group.

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

-- Players table:
-- +-----------+------------+
-- | player_id | group_id   |
-- +-----------+------------+
-- | 15        | 1          |
-- | 25        | 1          |
-- | 30        | 1          |
-- | 45        | 1          |
-- | 10        | 2          |
-- | 35        | 2          |
-- | 50        | 2          |
-- | 20        | 3          |
-- | 40        | 3          |
-- +-----------+------------+

-- Matches table:
-- +------------+--------------+--------------+-------------+-----------
---+
-- | match_id   | first_player | second_player | first_score |
second_score |
```

```
-- +------------+--------------+---------------+-------------+-----------
---+
-- | 1          | 15           | 45            | 3           | 0
|
-- | 2          | 30           | 25            | 1           | 2
|
-- | 3          | 30           | 15            | 2           | 0
|
-- | 4          | 40           | 20            | 5           | 2
|
-- | 5          | 35           | 50            | 1           | 1
|
-- +------------+--------------+---------------+-------------+-----------
---+

-- Result table:
-- +-----------+------------+
-- | group_id  | player_id  |
-- +-----------+------------+
-- | 1         | 15         |
-- | 2         | 35         |
-- | 3         | 40         |
-- +-----------+------------+

-- Solution
with t1 as(
select first_player, sum(first_score) as total
from
(select first_player, first_score
from matches
union all
select second_player, second_score
from matches) a
group by 1),

t2 as(
select *, coalesce(total,0) as score
from players p left join t1
on p.player_id = t1.first_player)

select group_id, player_id
from
(select *, row_number() over(partition by group_id order by group_id,
score desc) as rn
from t2) b
where b.rn = 1
```

-- Question 98
-- The Trips table holds all taxi trips. Each trip has a unique Id, while Client_Id and Driver_Id are both foreign keys to the Users_Id at the Users table. Status is an ENUM type of ('completed', 'cancelled_by_driver', 'cancelled_by_client').

```
-- +----+-----------+-----------+---------+--------------------+---------
-+
-- | Id | Client_Id | Driver_Id | City_Id |        Status
|Request_at|
-- +----+-----------+-----------+---------+--------------------+---------
-+
-- | 1  |     1     |    10     |    1    |     completed      |2013-10-
01|
-- | 2  |     2     |    11     |    1    | cancelled_by_driver|2013-10-
01|
-- | 3  |     3     |    12     |    6    |     completed      |2013-10-
01|
-- | 4  |     4     |    13     |    6    | cancelled_by_client|2013-10-
01|
-- | 5  |     1     |    10     |    1    |     completed      |2013-10-
02|
-- | 6  |     2     |    11     |    6    |     completed      |2013-10-
02|
-- | 7  |     3     |    12     |    6    |     completed      |2013-10-
02|
-- | 8  |     2     |    12     |    12   |     completed      |2013-10-
03|
-- | 9  |     3     |    10     |    12   |     completed      |2013-10-
03|
-- | 10 |     4     |    13     |    12   | cancelled_by_driver|2013-10-
03|
-- +----+-----------+-----------+---------+--------------------+---------
-+
```

-- The Users table holds all users. Each user has an unique Users_Id, and Role is an ENUM type of ('client', 'driver', 'partner').

```
-- +----------+--------+--------+
-- | Users_Id | Banned |  Role  |
-- +----------+--------+--------+
-- |    1     |   No   | client |
-- |    2     |   Yes  | client |
-- |    3     |   No   | client |
-- |    4     |   No   | client |
-- |    10    |   No   | driver |
-- |    11    |   No   | driver |
-- |    12    |   No   | driver |
-- |    13    |   No   | driver |
-- +----------+--------+--------+
```

-- Write a SQL query to find the cancellation rate of requests made by unbanned users (both client and driver must be unbanned) between Oct 1, 2013 and Oct 3, 2013. The cancellation rate is computed by dividing the number of canceled (by client or driver) requests made by unbanned users by the total number of requests made by unbanned users.

-- For the above tables, your SQL query should return the following rows with the cancellation rate being rounded to two decimal places.

```
-- +------------+-------------------+
-- |     Day    | Cancellation Rate |
-- +------------+-------------------+
-- | 2013-10-01 |       0.33        |
-- | 2013-10-02 |       0.00        |
-- | 2013-10-03 |       0.50        |
-- +------------+-------------------+
-- Credits:
-- Special thanks to @cak1erlizhou for contributing this question,
writing the problem description and adding part of the test cases.

-- Solution
with t1 as(
select request_at, count(status) as total
from trips
where client_id = any(select users_id
from users
where banned != 'Yes')
and driver_id = any(select users_id
from users
where banned != 'Yes')
and request_at between '2013-10-01' and '2013-10-03'
group by request_at),

t2 as
( select request_at, count(status) as cancel
from trips
where client_id = any(select users_id
from users
where banned != 'Yes')
and driver_id = any(select users_id
from users
where banned != 'Yes')
and request_at between '2013-10-01' and '2013-10-03'
and status != 'completed'
group by request_at
)

select request_at as Day, coalesce(round((cancel+0.00)/(total+0.00),2),0)
as "Cancellation Rate"
from t1 left join t2
using(request_at)
```

```
-- Question 113
-- Table: Spending

-- +-------------+---------+
-- | Column Name | Type    |
-- +-------------+---------+
-- | user_id     | int     |
-- | spend_date  | date    |
-- | platform    | enum    |
-- | amount      | int     |
-- +-------------+---------+
-- The table logs the spendings history of users that make purchases from
an online shopping website which has a desktop and a mobile application.
-- (user_id, spend_date, platform) is the primary key of this table.
-- The platform column is an ENUM type of ('desktop', 'mobile').
-- Write an SQL query to find the total number of users and the total
amount spent using mobile only, desktop only and both mobile and desktop
together for each date.

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

-- Spending table:
-- +---------+------------+----------+--------+
-- | user_id | spend_date | platform | amount |
-- +---------+------------+----------+--------+
-- | 1       | 2019-07-01 | mobile   | 100    |
-- | 1       | 2019-07-01 | desktop  | 100    |
-- | 2       | 2019-07-01 | mobile   | 100    |
-- | 2       | 2019-07-02 | mobile   | 100    |
-- | 3       | 2019-07-01 | desktop  | 100    |
-- | 3       | 2019-07-02 | desktop  | 100    |
-- +---------+------------+----------+--------+

-- Result table:
-- +------------+----------+--------------+-------------+
-- | spend_date | platform | total_amount | total_users |
-- +------------+----------+--------------+-------------+
-- | 2019-07-01 | desktop  | 100          | 1           |
-- | 2019-07-01 | mobile   | 100          | 1           |
-- | 2019-07-01 | both     | 200          | 1           |
-- | 2019-07-02 | desktop  | 100          | 1           |
-- | 2019-07-02 | mobile   | 100          | 1           |
-- | 2019-07-02 | both     | 0            | 0           |
-- +------------+----------+--------------+-------------+
-- On 2019-07-01, user 1 purchased using both desktop and mobile, user 2
purchased using mobile only and user 3 purchased using desktop only.
-- On 2019-07-02, user 2 purchased using mobile only, user 3 purchased
using desktop only and no one purchased using both platforms.

-- Solution
SELECT p.spend_date, p.platform, IFNULL(SUM(amount), 0) total_amount,
COUNT(DISTINCT u.user_id) total_users
FROM
(
SELECT DISTINCT(spend_date), 'desktop' platform FROM Spending
UNION
SELECT DISTINCT(spend_date), 'mobile' platform FROM Spending
UNION
```

```
SELECT DISTINCT(spend_date), 'both' platform FROM Spending
) p LEFT JOIN

(SELECT user_id, spend_date, SUM(amount) amount, (CASE WHEN
COUNT(DISTINCT platform)>1 THEN "both" ELSE platform END) platform
FROM Spending
GROUP BY spend_date, user_id) u

ON p.platform = u.platform AND p.spend_date=u.spend_date

GROUP BY p.spend_date, p.platform
```