Questions:

1. **1757. Recyclable and Low Fat Products**

+-------------+---------+
| Column Name | Type |
+-------------+---------+
product_id	int
low_fats	enum
recyclable	enum
+-------------+---------+
product_id is the primary key (column with unique values) for this table.
low_fats is an ENUM (category) of type ('Y', 'N') where 'Y' means this product is low fat and 'N' means it is not.
recyclable is an ENUM (category) of types ('Y', 'N') where 'Y' means this product is recyclable and 'N' means it is not.

Answer:

```
Select Product_id
from products
where low_fats='Y' and recyclable='Y'
```

2. **584. Find Customer Referee**

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

| Column Name | Type |

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

| id | int |

| name | varchar |

| referee_id | int |

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

In SQL, id is the primary key column for this table.

Each row of this table indicates the id of a customer, their name, and the id of the customer who referred them.

Answer:

```
Select name
from customer
where referee_id<>2 or referee_id is NULL
```

3. 595. Big Countries

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

| Column Name | Type |

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

| name | varchar |

| continent | varchar |

| area | int |

| population | int |

| gdp | bigint |

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

name is the primary key (column with unique values) for this table.

Each row of this table gives information about the name of a country, the continent to which it belongs, its area, the population, and its GDP value.

Answer:

```
Select name
, population
, area
from World
where area>=3000000
or population>=25000000
```

4. **1148. Article Views I**

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

| Column Name | Type |

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

| article_id | int |

| author_id | int |

| viewer_id | int |

| view_date | date |

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

There is no primary key (column with unique values) for this table, the table 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 a solution to find all the authors that viewed at least one of their own articles.

Return the result table sorted by id in ascending order.

Answer:

```
Select distinct
Author_id as id
from Views
where author_id=viewer_id
order by author_id asc
```


5. **1683. Invalid Tweets**

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

| Column Name | Type |

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

| tweet_id | int |

| content | varchar |

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

tweet_id is the primary key (column with unique values) for this table.

content consists of characters on an American Keyboard, and no other special characters.

This table contains all the tweets in a social media app.

Write a solution to find the IDs of the invalid tweets. The tweet is invalid if the number of characters used in the content of the tweet is **strictly greater** than 15.

Return the result table in **any order**.

Answer:

```
Select tweet_id
from tweets
where len(content)>15
```

6. **1378. Replace Employee ID With The Unique Identifier**


```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| id            | int     |
| name          | varchar |
+---------------+---------+
```

id is the primary key (column with unique values) for this table.

Each row of this table contains the id and the name of an employee in a company.

Table: EmployeeUNI

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| id            | int     |
| unique_id     | int     |
+---------------+---------+
```

(id, unique_id) is the primary key (combination of columns with unique values) for this table.

Each row of this table contains the id and the corresponding unique id of an employee in the company.

Write a solution to show the **unique ID** of each user, If a user does not have a unique ID replace just show null.

Return the result table in **any** order.

Answer:

```
Select name,
unique_id
from employees a
left join
EmployeeUNI b
on a.id=b.id
```


7. **1068. Product Sales Analysis I**

Sales

```
+-------------+-------+
| Column Name | Type  |
+-------------+-------+
| sale_id     | int   |
| product_id  | int   |
| year        | int   |
| quantity    | int   |
| price       | int   |
+-------------+-------+
```

(sale_id, year) is the primary key (combination of columns with unique values) of this table.

product_id is a foreign key (reference column) to Product table.

Each row of this table shows a sale on the product product_id in a certain year.

Note that the price is per unit.

Table: Product

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

| Column Name | Type |

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

| product_id | int |

| product_name | varchar |

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

product_id is the primary key (column with unique values) of this table.

Each row of this table indicates the product name of each product.

Write a solution to report the product_name, year, and price for each sale_id in the Sales table.

Return the resulting table in **any order**.

Answer:

```
Select product_name,
year,
price
from sales a
join product b
on a.product_id=b.product_id
order by product_name,year asc
```

8.

1581. Customer Who Visited but Did Not Make Any Transactions

Table: Visits

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

| Column Name | Type |

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

| visit_id | int |

| customer_id | int |

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

visit_id is the column with unique values for this table.

This table contains information about the customers who visited the mall.

Table: Transactions

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

| Column Name | Type |

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

| transaction_id | int |

| visit_id | int |

| amount | int |

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

transaction_id is column with unique values for this table.

This table contains information about the transactions made during the visit_id.

Write a solution to find the IDs of the users who visited without making any transactions and the number of times they made these types of visits.

Return the result table sorted in **any order**.

Answer:

```
Select customer_id as customer_id,

count(a.visit_id) as count_no_trans

from visits a

left join transactions b
```

```
on a.visit_id=b.visit_id
where b.visit_id is null
group by customer_id
```

9.

197. Rising Temperature

Table: Weather

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| id            | int     |
| recordDate    | date    |
| temperature   | int     |
+---------------+---------+
```

id is the column with unique values for this table.

There are no different rows with the same recordDate.

This table contains information about the temperature on a certain day.

Write a solution to find all dates' id with higher temperatures compared to its previous dates (yesterday).

Return the result table in **any order**.

Answer:

```
With cte as(
   Select id,recorddate,
temperature,
lag(temperature,1,NULL) over(order by recorddate asc) as lag_temp,
lag(recorddate,1,NULL) over(order by recorddate asc) as lag_date
from weather
)
Select id  as Id
from cte
where temperature>lag_temp
and datediff(day,lag_date,recorddate)=1
```

10.

1661. Average Time of Process per Machine

Table: Activity

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| machine_id     | int     |
| process_id     | int     |
| activity_type  | enum    |
| timestamp      | float   |
+----------------+---------+
```

The table shows the user activities for a factory website.

(machine_id, process_id, activity_type) is the primary key (combination of columns with unique values) of this table.

machine_id is the ID of a machine.

process_id is the ID of a process running on the machine with ID machine_id.

activity_type is an ENUM (category) of type ('start', 'end').

timestamp is a float representing the current time in seconds.

'start' means the machine starts the process at the given timestamp and 'end' means the machine ends the process at the given timestamp.

The 'start' timestamp will always be before the 'end' timestamp for every (machine_id, process_id) pair.

It is guaranteed that each (machine_id, process_id) pair has a 'start' and 'end' timestamp.

There is a factory website that has several machines each running the **same number of processes**. Write a solution to find the **average time** each machine takes to complete a process.

The time to complete a process is the 'end' timestamp minus the 'start' timestamp. The average time is calculated by the total time to complete every process on the machine divided by the number of processes that were run.

The resulting table should have the machine_id along with the **average time** as processing_time, which should be **rounded to 3 decimal places**.

Return the result table in **any order**.

Answer:

```
With cte as
(
   Select machine_id,
   process_id,
   Sum(case when activity_type='start' then timestamp else 0 end)  as start_1,
  Sum( case when activity_type='end' then timestamp else 0 end ) as end_1
  from activity
  group by machine_id,
   process_id
)
```

```
Select machine_id,
round((Sum(end_1-start_1)*1.0)/(count(process_id)*1.0),3) as processing_time
from cte
group by machine_id
```


11. **577. Employee Bonus**

Table: Employee

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| empId       | int     |
| name        | varchar |
| supervisor  | int     |
| salary      | int     |
+-------------+---------+
```

empId is the column with unique values for this table.

Each row of this table indicates the name and the ID of an employee in addition to their salary and the id of their manager.

Table: Bonus

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

| Column Name | Type |

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

| empId | int |

| bonus | int |

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

empId is the column of unique values for this table.

empId is a foreign key (reference column) to empId from the Employee table.

Each row of this table contains the id of an employee and their respective bonus.

Write a solution to report the name and bonus amount of each employee with a bonus **less than** 1000.

Return the result table in **any order**.

Answer:

```
Select name,bonus
from employee a
left join
bonus b
 on a.empid=b.empid
 where bonus<1000 or bonus is null
```

12.

1280. Students and Examinations

Table: Students

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

student_id is the primary key (column with unique values) for this table.

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

Table: Subjects

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| subject_name | varchar |
+--------------+---------+
```

subject_name is the primary key (column with unique values) for this table.

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

Table: Examinations

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| student_id   | int     |
| subject_name | varchar |
+--------------+---------+
```

There is no primary key (column with unique values) for this table. It may contain duplicates.

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

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

Write a solution to find the number of times each student attended each exam.

Return the result table ordered by student_id and subject_name.

Answer :

```
With cte as
(
   Select s.Student_id,
s.student_name,
e.subject_name,
count(e.subject_name) as attended_exams
from students s
left join examinations e
on s.student_id=e.student_id
group by s.Student_id,
s.student_name,
e.subject_name
)
,
cte2 as
(
   Select Student_id,student_name,
subject_name
```

```
from students
cross join subjects
)
Select a.student_id,
a.student_name,
a.subject_name,
coalesce(attended_exams,0) as attended_exams
from cte2 a
left join cte b
on a.student_id=b.student_id
and a.subject_name=b.subject_name
order by a.student_id asc, a.subject_name asc
```


13.

570. Managers with at Least 5 Direct Reports

Table: Employee

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
| department  | varchar |
| managerId   | int     |
+-------------+---------+
```

id is the primary key (column with unique values) for this table.

Each row of this table indicates the name of an employee, their department, and the id of their manager.

If managerId is null, then the employee does not have a manager.

No employee will be the manager of themself.

Write a solution to find managers with at least **five direct reports**.

Return the result table in **any order**.

Answer:

```
Select
  e2.name as name
  from employee e1
  join employee e2
  on e1.managerid=e2.id
  group by e1.managerid,
  e2.name
  having count(e1.id)>=5
```


14.

1934. Confirmation Rate

Table: Signups

```
+----------------+----------+
| Column Name    | Type     |
+----------------+----------+
| user_id        | int      |
| time_stamp     | datetime |
+----------------+----------+
```

user_id is the column of unique values for this table.

Each row contains information about the signup time for the user with ID user_id.

Table: Confirmations

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

| Column Name | Type |

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

| user_id | int |

| time_stamp | datetime |

| action | ENUM |

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

(user_id, time_stamp) is the primary key (combination of columns with unique values) for this table.

user_id is a foreign key (reference column) to the Signups table.

action is an ENUM (category) of the type ('confirmed', 'timeout')

Each row of this table indicates that the user with ID user_id requested a confirmation message at time_stamp and that confirmation message was either confirmed ('confirmed') or expired without confirming ('timeout').

The **confirmation rate** of a user is the number of 'confirmed' messages divided by the total number of requested confirmation messages. The confirmation rate of a user that did not request any confirmation messages is 0. Round the confirmation rate to **two decimal** places.

Write a solution to find the **confirmation rate** of each user.

Return the result table in **any order**.

Answer:

```
With cte as(

Select  a.user_id,

case when action='confirmed' then 1 else 0 end as flag

from signups a
```

```
left join Confirmations b
on a.user_id=b.user_id
)
Select
user_id,
round((sum(flag)*1.00/count(flag)*1.00),2) as confirmation_rate
from cte
group by user_id
```

15. **620. Not Boring Movies**

Table: Cinema

```
+----------------+----------+
| Column Name    | Type     |
+----------------+----------+
| id             | int      |
| movie          | varchar  |
| description    | varchar  |
| rating         | float    |
+----------------+----------+
```

id is the primary key (column with unique values) for this table.

Each row contains information about the name of a movie, its genre, and its rating.

rating is a 2 decimal places float in the range [0, 10]

Write a solution to report the movies with an odd-numbered ID and a description that is not "boring".

Return the result table ordered by rating **in descending order**.

Answer:

```
Select *
from cinema
where id%2=1
and description <> 'boring'
order by rating desc
```


16. **1251. Average Selling Price**

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| product_id    | int     |
| start_date    | date    |
| end_date      | date    |
| price         | int     |
+---------------+---------+
```

(product_id, start_date, end_date) is the primary key (combination of columns with unique values) for this table.

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

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

Table: UnitsSold

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

| Column Name | Type |

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

| product_id | int |

| purchase_date | date |

| units | int |

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

This table may contain duplicate rows.

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

Write a solution to find the average selling price for each product. average_price should be **rounded to 2 decimal places**. If a product does not have any sold units, its average selling price is assumed to be 0.

Return the result table in **any order**.

Answer:

```
With cte as
(
Select a.product_id,
price,
units,
Price*units as sales
from prices a
left join
unitssold b
on a.product_id=b.product_id
```

```
and b.purchase_date between Start_date and end_Date

)

Select
 product_id,
 coalesce(round(sum(Sales)*1.00/sum(units)*1.00,2),0) as average_price
 from cte
 group by product_id
```


17. **1075. Project Employees I**

```
-------------+---------+
| 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.

Each row of this table indicates that the employee with employee_id is working on the project with project_id.

Table: Employee

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

| Column Name | Type |

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

| employee_id | int |

| name | varchar |

| experience_years | int |

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

employee_id is the primary key of this table. It's guaranteed that experience_years is not NULL.

Each row of this table contains information about one employee.

Write an SQL query that reports the **average** experience years of all the employees for each project, **rounded to 2 digits**.

Return the result table in **any order**.

Answer:

```
Select p.Project_id,
round((sum(experience_years)*1.00/count(p.employee_id)*1.00),2) average_years
from project p
left join
employee e
on p.employee_id=e.employee_id
```

group by p.project_id

18. **1633. Percentage of Users Attended a Contest**

Table: Users

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

| Column Name | Type |

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

| user_id | int |

| user_name | varchar |

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

user_id is the primary key (column with unique values) for this table.

Each row of this table contains the name and the id of a user.

Table: Register

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

| Column Name | Type |

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

| contest_id | int |

| user_id | int |

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

(contest_id, user_id) is the primary key (combination of columns with unique values) for this table.

Each row of this table contains the id of a user and the contest they registered into.

Write a solution to find the percentage of the users registered in each contest rounded to **two decimals**.

Return the result table ordered by percentage in **descending order**. In case of a tie, order it by contest_id in **ascending order**.

Answer:

```
Select Contest_id,
round((count(b.user_id)*1.00/(Select count(user_id) from users))*100,2) as percentage
from register b
group by contest_id
order by percentage desc,contest_id asc
```

19. **1211. Queries Quality and Percentage**

Table: Queries

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| query_name  | varchar |
| result      | varchar |
| position    | int     |
| rating      | int     |
+-------------+---------+
```

This table may have duplicate rows.

This table contains information collected from some queries on a database.

The position column has a value from **1** to **500**.

The rating column has a value from **1** to **5**. Query with rating less than 3 is a poor query.

We define query quality as:

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

We also define poor query percentage as:

The percentage of all queries with rating less than 3.

Write a solution to find each query_name, the quality and poor_query_percentage.

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

Return the result table in **any order**.

Answer:

```
Select Query_name,
round(Sum((rating*1.00/position))/count(result),2) as quality,
round(sum(case when rating<3 then 1 else 0 end)*1.00/count(rating) *100,2) as poor_query_percentage
from queries
group by query_name
```

20. **1193. Monthly Transactions I**

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.

Return the result table in **any order**.

Answer:

```
Select concat(datepart(year,trans_date),'-', right('0'+
convert(nvarchar(2),datepart(month,trans_date)),2)) month,
   country,
   count(id) as trans_count,
   sum(case when state='approved' then 1 else 0 end) as approved_count,
   sum(amount) as trans_total_amount,
   sum(case when state='approved' then amount else 0 end) as approved_total_amount
   from Transactions
   group by concat(datepart(year,trans_date),'-', right('0'+
convert(nvarchar(2),datepart(month,trans_date)),2)), country
  order by country desc
```


21. **1174. Immediate Food Delivery II**

Table: Delivery

```
+-----------------------------+---------+
| Column Name                 | Type    |
+-----------------------------+---------+
| delivery_id                 | int     |
| customer_id                 | int     |
| order_date                  | date    |
| customer_pref_delivery_date | date    |
+-----------------------------+---------+
```

delivery_id is the column of unique values 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 customer's preferred delivery date is the same as the order date, then the order is called **immediate;** otherwise, it is called **scheduled**.

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

Write a solution to find the percentage of immediate orders in the first orders of all customers, **rounded to 2 decimal places**.

Answer:

```
WIth cte as(
Select *,
case when order_date=customer_pref_delivery_Date then 'immediate' else 'scheduled' end as delivery_flag,
rank() over(partition by customer_id order by order_date asc) as rank
from delivery
)
Select
 round(sum(case when delivery_flag='immediate' then 1 else 0 end)*1.00/count(*)*100,2) as immediate_percentage
 from cte
 where rank=1
```

22. **550. Game Play Analysis IV**

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 (combination of columns with unique values) of this table.

This table shows the activity of players of some games.

Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

Write a solution to report 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.

Answer:

```
With cte as(

Select player_id,

rank() over( partition by player_id order by event_Date) as rank,

datediff(day,event_Date,lead(event_Date,1) over(partition by player_id order by event_Date)) as lead_Date_flag

from Activity

)
```

```
Select
round(Sum(case when rank=1 and lead_date_flag=1 then 1 else 0 end)*1.00/count(distinct player_id),2) as fraction
from cte
where rank=1
```

23. **2356. Number of Unique Subjects Taught by Each Teacher**

Table: Teacher

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| teacher_id  | int  |
| subject_id  | int  |
| dept_id     | int  |
+-------------+------+
```

(subject_id, dept_id) is the primary key (combinations of columns with unique values) of this table.

Each row in this table indicates that the teacher with teacher_id teaches the subject subject_id in the department dept_id.

Write a solution to calculate the number of unique subjects each teacher teaches in the university.

Return the result table in **any order**.

Answer:

```
Select teacher_id,
count(distinct subject_id) as cnt
from teacher
group by teacher_id
```

24. [1141. User Activity for the Past 30 Days I]

Table: Activity

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| user_id       | int     |
| session_id    | int     |
| activity_date | date    |
| activity_type | enum    |
+---------------+---------+
```

This table may have duplicate rows.

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

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

Note that each session belongs to exactly one user.

Write a solution to find the daily active user count for a period of 30 days ending 2019-07-27 inclusively. A user was active on someday if they made at least one activity on that day.

Return the result table in **any order**.

Answer:

```
With cte as
(
   Select
   activity_date,
   user_id,
   sum(case when activity_type is not null then 1 else 0 end) as flag
   from activity
   where activity_date  between dateadd(day,-29,'2019-07-27') and '2019-07-27'
   group by user_id,activity_date

)
Select activity_date  as day,
count(distinct user_id) as active_users
from  cte
where flag>=1
group by activity_date
```

25. [1070. Product Sales Analysis III]

Table: Sales

```
+-------------+-------+
| Column Name | Type  |
+-------------+-------+
| sale_id     | int   |
| product_id  | int   |
| year        | int   |
| quantity    | int   |
| price       | int   |
+-------------+-------+
```

(sale_id, year) is the primary key (combination of columns with unique values) of this table.
product_id is a foreign key (reference column) to Product table.
Each row of this table shows a sale on the product product_id in a certain year.
Note that the price is per unit.

Table: Product

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| product_id   | int     |
| product_name | varchar |
+--------------+---------+
```

product_id is the primary key (column with unique values) of this table.
Each row of this table indicates the product name of each product.

Write a solution to select the **product id**, **year**, **quantity**, and **price** for the **first year** of every product sold.
Return the resulting table in **any order**.

Answer:

```
Select a.product_id,
year as first_year,
quantity,
price
from sales a
```

```
join  (Select product_id,
min(year) as first_year
from sales
group by product_id
)b
on a.product_id=b.product_id
and a.year=b.first_year
```

26. **[596. Classes More Than 5 Students]**

Table: Courses

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| student     | varchar |
| class       | varchar |
+-------------+---------+
```

(student, class) is the primary key (combination of columns with unique values) for this table.
Each row of this table indicates the name of a student and the class in which they are enrolled.

Write a solution to find all the classes that have **at least five students**.
Return the result table in **any order**.

Answer:

```
Select class
from courses
group by class
having count(student)>=5
```


27. **[1729. Find Followers Count]**

Table: Followers

+-------------+------+
| Column Name | Type |
+-------------+------+
| user_id | int |
| follower_id | int |
+-------------+------+
(user_id, follower_id) is the primary key (combination of columns with unique values) for this table.
This table contains the IDs of a user and a follower in a social media app where the follower follows the user.

Write a solution that will, for each user, return the number of followers.
Return the result table ordered by user_id in ascending order.

Answer:

```
Select user_id,
count(follower_id) as followers_count
from followers
group by user_id
order by user_id asc
```

28. **619. Biggest Single Number**

Table: MyNumbers

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

| Column Name | Type |

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

| num | int |

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

This table may contain duplicates (In other words, there is no primary key for this table in SQL).

Each row of this table contains an integer.

A **single number** is a number that appeared only once in the MyNumbers table.

Find the largest **single number**. If there is no **single number**, report null.

Answer:

```
With cte as(
Select num
from mynumbers
group by num
having count(num)=1
)
Select
max(num) as num
from cte
```


29. **1045. Customers Who Bought All Products**

Table: Customer

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| customer_id | int     |
| product_key | int     |
+-------------+---------+
```

This table may contain duplicates rows.

customer_id is not NULL.

product_key is a foreign key (reference column) to Product table.

Table: Product

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

| Column Name | Type |

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

| product_key | int |

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

product_key is the primary key (column with unique values) for this table.

Write a solution to report the customer ids from the Customer table that bought all the products in the Product table.

Return the result table in **any order**.

Answer:


```
With cte as(
select customer_id,products,
count(distinct product_key) as count
from customer a
cross join
(Select count(distinct product_key) as products
   from product) b
group by customer_id
)
Select customer_id
from cte
where products=count
```

30. **1731. The Number of Employees Which Report to Each Employee**

Table: Employees

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

| Column Name | Type |

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

| employee_id | int |

| name | varchar |

| reports_to | int |

| age | int |

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

employee_id is the column with unique values for this table.

This table contains information about the employees and the id of the manager they report to. Some employees do not report to anyone (reports_to is null).

For this problem, we will consider a **manager** an employee who has at least 1 other employee reporting to them.

Write a solution to report the ids and the names of all **managers**, the number of employees who report **directly** to them, and the average age of the reports rounded to the nearest integer.

Return the result table ordered by employee_id.

Answer:

```
Select a.reports_to as employee_id,
b.name,
count(a.employee_id) as reports_count,
round(avg(a.age)) as average_age
from employees a
join employees b
```

```
on a.reports_to=b.employee_id
group by a.reports_to,
b.name
order by a.reports_to
```

31. **[1789. Primary Department for Each Employee]**

Table: Employee

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| employee_id   | int     |
| department_id | int     |
| primary_flag  | varchar |
+---------------+---------+
```

(employee_id, department_id) is the primary key (combination of columns with unique values) for this table.

employee_id is the id of the employee.

department_id is the id of the department to which the employee belongs.

primary_flag is an ENUM (category) of type ('Y', 'N'). If the flag is 'Y', the department is the primary department for the employee. If the flag is 'N', the department is not the primary.

Employees can belong to multiple departments. When the employee joins other departments, they need to decide which department is their primary department. Note that when an employee belongs to only one department, their primary column is 'N'.

Write a solution to report all the employees with their primary department. For employees who belong to one department, report their only department.

Return the result table in **any order**.

Answer:

```
With cte as(

   Select *,
   count(department_id) over(partition by employee_id ) as count
   from employee
)
Select employee_id,
department_id
from cte
where count=1
or (count>1 and primary_flag='Y')
```


32. **610. Triangle Judgement**

Table: Triangle

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

| Column Name | Type |

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

| x | int |

| y | int |

| z | int |

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

In SQL, (x, y, z) is the primary key column for this table.

Each row of this table contains the lengths of three line segments.

Report for every three line segments whether they can form a triangle.

Return the result table in **any order**.

Answer:

```
Select x,y,z,
case when (x+y>z and y+z>x and x+z>y) then 'Yes'
   else 'No' end as triangle
   from triangle
```


33. **180. Consecutive Numbers**

Table: Logs

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

| Column Name | Type |

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

| id | int |

| num | varchar |

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

In SQL, id is the primary key for this table.

id is an autoincrement column starting from 1.

Find all numbers that appear at least three times consecutively.

Return the result table in **any order**.

Answer:

```
With cte as
(
Select id,
num,
case when lag(num,1) over(order by id)=num and num=lead(num,1) over(order by id) then 1 else 0 end as flag
from logs
)
Select distinct num as consecutivenums
from cte
where flag=1
```


34. **1164. Product Price at a Given Date**

Table: Products

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

| Column Name | Type |

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

| product_id | int |

| new_price | int |

| change_date | date |

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

(product_id, change_date) is the primary key (combination of columns with unique values) 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 a solution to find the prices of all products on 2019-08-16. Assume the price of all products before any change is 10.

Return the result table in **any order**.

Answer:

```
With cte as(
   Select product_id,
   new_price,
   rank() over( partition by product_id order by change_date desc) as rank_product
   from products
   where change_date<='2019-08-16'
)
Select a.product_id,
coalesce(new_price,10) as price
from
(
   Select distinct product_id
   from products
) a
```

```
left join
cte b
on a.product_id=b.product_id
where b.rank_product=1
or b.rank_product is null
```


35. **1204. Last Person to Fit in the Bus**

Table: Queue

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| person_id   | int     |
| person_name | varchar |
| weight      | int     |
| turn        | int     |
+-------------+---------+
```

person_id column contains unique values.

This table has the information about all people waiting for a bus.

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

turn determines the order of which the people will board the bus, where turn=1 denotes the first person to board and turn=n denotes the last person to board.

weight is the weight of the person in kilograms.

There is a queue of people waiting to board a bus. However, the bus has a weight limit of 1000 **kilograms**, so there may be some people who cannot board.

Write a solution to find the person_name of the **last person** that can fit on the bus without exceeding the weight limit. The test cases are generated such that the first person does not exceed the weight limit.

Note that *only one* person can board the bus at any given turn.

Answer:

```
With cte as(
   Select person_name,
   turn,
   sum(weight) over(order by turn asc) as cum_sum
   from queue
)
Select person_name
from cte
where turn=(Select max(turn) from cte
where cum_sum<=1000)
```


36. **1907. Count Salary Categories**

Table: Accounts

```
+-------------+------+
| Column Name | Type |
+-------------+------+
```

| account_id | int |

| income | int |

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

account_id is the primary key (column with unique values) for this table.

Each row contains information about the monthly income for one bank account.

Write a solution to calculate the number of bank accounts for each salary category. The salary categories are:

- "Low Salary": All the salaries **strictly less** than $20000.
- "Average Salary": All the salaries in the **inclusive** range [$20000, $50000].
- "High Salary": All the salaries **strictly greater** than $50000.

The result table **must** contain all three categories. If there are no accounts in a category, return 0.

Return the result table in **any order**.

Answer:

```
Select a.category,
coalesce(b.accounts_count,0) as accounts_count
from
(
Select 'Low Salary' as category
union
Select 'Average Salary' as category
union
Select 'High Salary' as category) a
```

```
left join
(
Select
case when income<20000 then 'Low Salary'
when income between 20000 and 50000 then 'Average Salary'
when income>50000 then 'High Salary' end as category,
count(account_id) as accounts_count
from Accounts
group by
case when income<20000 then 'Low Salary'
when income between 20000 and 50000 then 'Average Salary'
when income>50000 then 'High Salary' end
) b
on a.category=b.category
```


37. **1978. Employees Whose Manager Left the Company**

Table: Employees

```
+-------------+----------+
| Column Name | Type     |
+-------------+----------+
| employee_id | int      |
| name        | varchar  |
| manager_id  | int      |
```

| salary | int |

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

In SQL, employee_id is the primary key for this table.

This table contains information about the employees, their salary, and the ID of their manager. Some employees do not have a manager (manager_id is null).

Find the IDs of the employees whose salary is strictly less than $30000 and whose manager left the company. When a manager leaves the company, their information is deleted from the Employees table, but the reports still have their manager_id set to the manager that left.

Return the result table ordered by employee_id.

Answer:

```
Select a.employee_id
from employees a
left join employees b
on a.manager_id=b.employee_id
where a.salary<30000
and a.manager_id is not null
and b.employee_id is null
order by a.employee_id asc
```


38. **626. Exchange Seats**

Table: Seat

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

| Column Name | Type |

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

| id | int |

| student | varchar |

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

id is the primary key (unique value) column for this table.

Each row of this table indicates the name and the ID of a student.

The ID sequence always starts from 1 and increments continuously.

Write a solution to swap the seat id of every two consecutive students. If the number of students is odd, the id of the last student is not swapped.

Return the result table ordered by id **in ascending order**.

Answer:

```
Select case when id%2=1
and
id=(select max(id) from seat) then id
when id%2=1 then id+1
else id-1 end as id,
student
from
Seat
order by id asc
```


39. **1341. Movie Rating**

Table: Movies

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

| Column Name | Type |

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

| movie_id | int |

| title | varchar |

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

movie_id is the primary key (column with unique values) 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 (column with unique values) for this table.

The column 'name' has unique values.

Table: MovieRating

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

| Column Name | Type |

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

| movie_id | int |

| user_id | int |

| rating | int |

| created_at | date |

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

(movie_id, user_id) is the primary key (column with unique values) 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 a solution to:

- Find the name of the user who has rated the greatest number of movies. In case of a tie, return the lexicographically smaller user name.
- Find the movie name with the **highest average** rating in February 2020. In case of a tie, return the lexicographically smaller movie name.

Answer:

```
Select results from
(
   Select top 1 name as results
   from
   (
   Select a.user_id,
   name,
   count(movie_id) as countmovies
   from movierating a
   join
   users b
   on
   a.user_id=b.user_id
   group by a.user_id,name
   ) users1
   order by countmovies desc,name asc
)first_part

union all
Select * from
(
   Select top 1 title as results
from
```

```
(
Select a.movie_id,
title,
round(Sum(rating)*1.0/count(user_id),2) as rating
from movierating a
join
movies b
on
a.movie_id=b.movie_id
where left(created_at,7)='2020-02'
group by a.movie_id,title
)movies1
order by rating desc, title asc
)second_part
```


40. **1321. Restaurant Growth**

Table: Customer

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| customer_id   | int     |
| name          | varchar |
| visited_on    | date    |
| amount        | int     |
+---------------+---------+
```

In SQL,(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) has 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).

Compute the moving average of how much the customer paid in a seven days window (i.e., current day + 6 days before). average_amount should be **rounded to two decimal places**.

Return the result table ordered by visited_on **in ascending order**.

Answer:

```
With cte as
(
   Select visited_on,
   sum(amount) as amount,
   row_number() over(order by visited_on asc) as rn
   from customer
   group by visited_on
),
 cte2 as(
Select visited_on,rn,
sum(amount)
over( order by visited_on asc rows between 6 preceding and current row ) as amount,
round(avg(amount) over( order by visited_on asc rows between 6 preceding and current row )*1.0,2) as average_amount
from cte)
Select
visited_on,
amount,
average_amount
```

```
from cte2
where rn>=7
```

41. **602. Friend Requests II: Who Has the Most Friends**

Table: RequestAccepted

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| requester_id   | int     |
| accepter_id    | int     |
| accept_date    | date    |
+----------------+---------+
```

(requester_id, accepter_id) is the primary key (combination of columns with unique values) for this table.

This table contains the ID of the user who sent the request, the ID of the user who received the request, and the date when the request was accepted.

Write a solution to find the people who have the most friends and the most friends number.

The test cases are generated so that only one person has the most friends.

Answer:

```
Select
id,
sum(num) as num
from (
Select requester_id as id,
```

```
count(accepter_id) as num
from requestaccepted
group by requester_id
union all
Select accepter_id as id,
count(requester_id) as num
from requestaccepted
group by accepter_id
)base
group by id
order by num desc
limit 1
```


42. **585. Investments in 2016**

Table: Insurance

```
+-------------+-------+
| Column Name | Type  |
+-------------+-------+
| pid         | int   |
| tiv_2015    | float |
| tiv_2016    | float |
| lat         | float |
| lon         | float |
+-------------+-------+
```

pid is the primary key (column with unique values) for this table.

Each row of this table contains information about one policy where:

pid is the policyholder's policy ID.

tiv_2015 is the total investment value in 2015 and tiv_2016 is the total investment value in 2016.

lat is the latitude of the policy holder's city. It's guaranteed that lat is not NULL.

lon is the longitude of the policy holder's city. It's guaranteed that lon is not NULL.

Write a solution to report the sum of all total investment values in 2016 tiv_2016, for all policyholders who:

- have the same tiv_2015 value as one or more other policyholders, and
- are not located in the same city as any other policyholder (i.e., the (lat, lon) attribute pairs must be unique).

Round tiv_2016 to **two decimal places**.

Answer:


```
with cte as(
Select tiv_2016,
count(*) over(partition by tiv_2015) as count_tiv,
count(*) over(partition by lat,lon) as count_loc
from insurance
)
Select round(sum(tiv_2016),2) as tiv_2016
from cte
where count_tiv>1
and count_loc=1
```

43. **185. Department Top Three Salaries**

Table: Employee

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

| Column Name | Type |

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

| id | int |

| name | varchar |

| salary | int |

| departmentId | int |

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

id is the primary key (column with unique values) for this table.

departmentId is a foreign key (reference column) of the ID from the Department table.

Each row of this table indicates the ID, name, and salary of an employee. It also contains the ID of their department.

Table: Department

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

| Column Name | Type |

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

| id | int |

| name | varchar |

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

id is the primary key (column with unique values) for this table.

Each row of this table indicates the ID of a department and its name.

A company's executives are interested in seeing who earns the most money in each of the company's departments. A **high earner** in a department is an employee who has a salary in the **top three unique** salaries for that department.

Write a solution to find the employees who are **high earners** in each of the departments.

Return the result table **in any order**.

Answer:


```
With cte as(

   Select a.name,
   salary,
   departmentid,
   b.name as dept_name,
   dense_rank() over(partition by departmentid order by salary desc) as rn
   from employee a
   join department b
   on a.departmentid=b.id
)
Select dept_name as department,
name as employee,
salary
from cte
where rn<=3
```

44. **1667. Fix Names in a Table**

Table: Users

```
+----------------+---------+
| Column Name    | Type    |
+----------------+---------+
| user_id        | int     |
| name           | varchar |
+----------------+---------+
```

user_id is the primary key (column with unique values) for this table.

This table contains the ID and the name of the user. The name consists of only lowercase and uppercase characters.

Write a solution to fix the names so that only the first character is uppercase and the rest are lowercase.

Return the result table ordered by user_id.

Answer:

```
Select user_id,
concat(
   upper(left(name,1)),
   lower(right(name,length(name)-1)))
   as name
from users
order by user_id
```


45. **1527. Patients With a Condition**

Table: Patients

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| patient_id   | int     |
| patient_name | varchar |
| conditions   | varchar |
+--------------+---------+
```

patient_id is the primary key (column with unique values) for this table.

'conditions' contains 0 or more code separated by spaces.

This table contains information of the patients in the hospital.

Write a solution to find the patient_id, patient_name, and conditions of the patients who have Type I Diabetes. Type I Diabetes always starts with DIAB1 prefix.

Return the result table in **any order**.

Answer:

```
Select *
from Patients
where conditions like '% DIAB1%'
or conditions like 'DIAB1%'
```


46. **196. Delete Duplicate Emails**

Table: Person

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| email       | varchar |
+-------------+---------+
```

id is the primary key (column with unique values) for this table.

Each row of this table contains an email. The emails will not contain uppercase letters.

Write a solution to **delete** all duplicate emails, keeping only one unique email with the smallest id.

For SQL users, please note that you are supposed to write a DELETE statement and not a SELECT one.

For Pandas users, please note that you are supposed to modify Person in place.

After running your script, the answer shown is the Person table. The driver will first compile and run your piece of code and then show the Person table. The final order of the Person table **does not matter**.

Answer:

```
Delete
From person
  where id
  not in
  (select min(id)
  From person
  group by email)
```


47. **176. Second Highest Salary**

Table: Employee

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| id          | int  |
| salary      | int  |
+-------------+------+
```

id is the primary key (column with unique values) for this table.

Each row of this table contains information about the salary of an employee.

Write a solution to find the second highest **distinct** salary from the Employee table. If there is no second highest salary, return null (return None in Pandas).

Answer:

```
Select max(salary) as SecondHighestSalary
from employee
where salary<(Select max(salary) from employee)
```


48. **1484. Group Sold Products By The Date**

Table Activities:

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| sell_date   | date    |
| product     | varchar |
```

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

There is no primary key (column with unique values) for this table. It may contain duplicates.

Each row of this table contains the product name and the date it was sold in a market.

Write a solution to find for each date the number of different products sold and their names.

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

Return the result table ordered by sell_date.

Answer:

```
Select Sell_date,
count(distinct product) as num_sold,
string_Agg( product , ',') within group (order by product) as products
from
(Select distinct *
from activities
 )a
group by sell_date
order by sell_Date,products
```


49. **1327. List the Products Ordered in a Period**

Table: Products

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

| Column Name | Type |

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

| product_id | int |

| product_name | varchar |

| product_category | varchar |

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

product_id is the primary key (column with unique values) for this table.

This table contains data about the company's products.

Table: Orders

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

| Column Name | Type |

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

| product_id | int |

| order_date | date |

| unit | int |

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

This table may have duplicate rows.

product_id is a foreign key (reference column) to the Products table.

unit is the number of products ordered in order_date.

Write a solution to get the names of products that have at least 100 units ordered in **February 2020** and their amount.

Return the result table in **any order**.

Answer:

```
With cte as
(
   Select a.product_id,
```

```
    unit,
    b.product_name
    from orders a
    join products b
    on a.product_id=b.product_id
    where left(order_date,7)='2020-02'
)
Select product_name,
sum(unit) as unit
from cte
group by product_name
having sum(unit)>=100
```


50. **1517. Find Users With Valid E-Mails**

Table: Users

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

| Column Name | Type |

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

| user_id | int |

| name | varchar |

| mail | varchar |

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

user_id is the primary key (column with unique values) for this table.

This table contains information of the users signed up in a website. Some e-mails are invalid.

Write a solution to find the users who have **valid emails**.

A valid e-mail has a prefix name and a domain where:

- **The prefix name** is a string that may contain letters (upper or lower case), digits, underscore '_', period '.', and/or dash '-'. The prefix name **must** start with a letter.
- **The domain** is '@leetcode.com'.

Return the result table in **any order**.

Answer:

```
Select user_id,
name,
mail
from users
where mail regexp '^[a-zA-Z]+[a-zA-Z0-9_.-]*@leetcode\?\\.com$'
```