,Given a table of candidates and their skills, you're tasked with finding the candidates best suited for an open Data Science job. You want to find candidates who are proficient in Python, Tableau, and PostgreSQL.

Write a query to list the candidates who possess all of the required skills for the job. Sort the output by candidate ID in ascending order.

**Assumption:**

* There are no duplicates in the candidates table.

**candidates Table:**

|  |  |
| --- | --- |
| **Column Name** | **Type** |
| candidate\_id | integer |
| skill | varchar |

**candidates Example Input:**

|  |  |
| --- | --- |
| **candidate\_id** | **skill** |
| 123 | Python |
| 123 | Tableau |
| 123 | PostgreSQL |
| 234 | R |
| 234 | PowerBI |
| 234 | SQL Server |
| 345 | Python |
| 345 | Tableau |

**Example Output:**

|  |
| --- |
| **candidate\_id** |
| 123 |

**SELECT candidate\_id**

**FROM candidates**

**WHERE skill in ('Python','Tableau','PostgreSQL')**

**GROUP BY candidate\_id HAVING COUNT(DISTINCT skill)=3**

**ORDER BY candidate\_id;**

Table: Person

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

| Column Name | Type |

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

| personId | int |

| lastName | varchar |

| firstName | varchar |

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

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

This table contains information about the ID of some persons and their first and last names.

Table: Address

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

| Column Name | Type |

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

| addressId | int |

| personId | int |

| city | varchar |

| state | varchar |

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

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

Each row of this table contains information about the city and state of one person with ID = PersonId.

Write a solution to report the first name, last name, city, and state of each person in the Person table. If the address of a personId is not present in the Addresstable, report null instead.

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

The result format is in the following example.

**SELECT**

**p.firstname , p.lastname , a.city , a.state**

**FROM**

**person p LEFT JOIN Address a**

**ON**

**p.personid = a.personid**

**WHERE**

**a.addressid is not null;**

**SELECT**

**e.name as Employee**

**FROM**

**Employee2 e**

**JOIN Employee2 m**

**ON**

**e.managerid = m.id**

**WHERE e.salary>m.salary;**

Assume you're given two tables containing data about Facebook Pages and their respective likes (as in "Like a Facebook Page").

Write a query to return the IDs of the Facebook pages that have zero likes. The output should be sorted in ascending order based on the page IDs.

**pages Example Input:**

|  |  |
| --- | --- |
| **page\_id** | **page\_name** |
| 20001 | SQL Solutions |
| 20045 | Brain Exercises |
| 20701 | Tips for Data Analysts |

**page\_likes Example Input:**

|  |  |  |
| --- | --- | --- |
| **user\_id** | **page\_id** | **liked\_date** |
| 111 | 20001 | 04/08/2022 00:00:00 |
| 121 | 20045 | 03/12/2022 00:00:00 |
| 156 | 20001 | 07/25/2022 00:00:00 |

**Example Output:**

|  |
| --- |
| **page\_id** |
| 20701 |

The dataset you are querying against may have different input & output - **this is just an example**!

**SELECT p.page\_id**

**FROM pages p**

**LEFT JOIN page\_likes pl ON p.page\_id = pl.page\_id**

**WHERE pl.page\_id IS NULL**

**ORDER BY p.page\_id ASC;**

Tesla is investigating production bottlenecks and they need your help to extract the relevant data. Write a query to determine which parts have begun the assembly process but are not yet fi nished.

Assumptions:

* parts\_assembly table contains all parts currently in production, each at varying stages of the assembly process.
* An unfinished part is one that lacks a finish\_date.

This question is straightforward, so let's approach it with simplicity in both thinking and solution.

*Effective April 11th 2023, the problem statement and assumptions were updated to enhance clarity.*

### **parts\_assembly Table**

|  |  |
| --- | --- |
| **Column Name** | **Type** |
| part | string |
| finish\_date | datetime |
| assembly\_step | integer |

### **parts\_assembly Example Input**

|  |  |  |
| --- | --- | --- |
| **part** | **finish\_date** | **assembly\_step** |
| battery | 01/22/2022 00:00:00 | 1 |
| battery | 02/22/2022 00:00:00 | 2 |
| battery | 03/22/2022 00:00:00 | 3 |
| bumper | 01/22/2022 00:00:00 | 1 |
| bumper | 02/22/2022 00:00:00 | 2 |
| bumper |  | 3 |
| bumper |  | 4 |

### **Example Output**

|  |  |
| --- | --- |
| **part** | **assembly\_step** |
| bumper | 3 |
| bumper | 4 |

**SELECT part,assembly\_step**

**FROM parts\_assembly**

**WHERE finish\_date is null;**

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 report all the duplicate emails. Note that it's guaranteed that the email field is not NULL.

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

The result format is in the following example.

**Example 1:**

**Input:**

Person table:

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

| id | email |

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

| 1 | a@b.com |

| 2 | c@d.com |

| 3 | a@b.com |

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

**Output:**

+---------+

| Email |

+---------+

| a@b.com |

+---------+

**SELECT email from person**

**group by email having count(\*)>1;**

Assume you're given the table on user viewership categorised by device type where the three types are laptop, tablet, and phone.

Write a query that calculates the total viewership for laptops and mobile devices where mobile is defined as the sum of tablet and phone viewership. Output the total viewership for laptops as laptop\_reviews and the total viewership for mobile devices as mobile\_views.

*Effective 15 April 2023, the solution has been updated with a more concise and easy-to-understand approach.*

### **viewership Table**

|  |  |
| --- | --- |
| **Column Name** | **Type** |
| user\_id | integer |
| device\_type | string ('laptop', 'tablet', 'phone') |
| view\_time | timestamp |

### **viewership Example Input**

|  |  |  |
| --- | --- | --- |
| **user\_id** | **device\_type** | **view\_time** |
| 123 | tablet | 01/02/2022 00:00:00 |
| 125 | laptop | 01/07/2022 00:00:00 |
| 128 | laptop | 02/09/2022 00:00:00 |
| 129 | phone | 02/09/2022 00:00:00 |
| 145 | tablet | 02/24/2022 00:00:00 |

### **Example Output**

|  |  |
| --- | --- |
| **laptop\_views** | **mobile\_views** |
| 2 | 3 |

**SELECT**

**SUM(CASE WHEN device\_type = 'laptop' THEN 1 ELSE 0 END) AS laptop\_views,**

**SUM(CASE WHEN device\_type IN ('tablet', 'phone') THEN 1 ELSE 0 END) AS mobile\_views**

**FROM viewership;**

Write a query to identify the top 2 Power Users who sent the highest number of messages on Microsoft Teams in August 2022. Display the IDs of these 2 users along with the total number of messages they sent. Output the results in descending order based on the count of the messages.

Assumption:

* No two users have sent the same number of messages in August 2022.

**messages Table:**

|  |  |
| --- | --- |
| **Column Name** | **Type** |
| message\_id | integer |
| sender\_id | integer |
| receiver\_id | integer |
| content | varchar |
| sent\_date | datetime |

**messages Example Input:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **message\_id** | **sender\_id** | **receiver\_id** | **content** | **sent\_date** |
| 901 | 3601 | 4500 | You up? | 08/03/2022 00:00:00 |
| 902 | 4500 | 3601 | Only if you're buying | 08/03/2022 00:00:00 |
| 743 | 3601 | 8752 | Let's take this offline | 06/14/2022 00:00:00 |
| 922 | 3601 | 4500 | Get on the call | 08/10/2022 00:00:00 |

**Example Output:**

|  |  |
| --- | --- |
| **sender\_id** | **message\_count** |
| 3601 | 2 |
| 4500 | 1 |

The dataset you are querying against may have different input & output - **this is just an example**!

**SELECT sender\_id , count(\*) as message\_count**

**FROM messages**

**WHERE sent\_date > '2022-08-01' and sent\_date < '2022-09-01'**

**GROUP BY sender\_id**

**ORDER BY message\_count DESC LIMIT 2;**

Table: Customers

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

| 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 and name of a customer.

Table: Orders

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

| Column Name | Type |

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

| id | int |

| customerId | int |

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

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

customerId is a foreign key (reference columns) of the ID from the Customers table.

Each row of this table indicates the ID of an order and the ID of the customer who ordered it.

Write a solution to find all customers who never order anything.

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

The result format is in the following example.

**Example 1:**

**Input:**

Customers table:

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

| id | name |

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

| 1 | Joe |

| 2 | Henry |

| 3 | Sam |

| 4 | Max |

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

Orders table:

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

| id | customerId |

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

| 1 | 3 |

| 2 | 1 |

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

**Output:**

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

| Customers |

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

| Henry |

| Max |

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

**SELECT name AS Customers**

**FROM Customers**

**WHERE id NOT IN (**

**SELECT customerId FROM Orders**

**);**