Problem 1

SQL Schema Pandas Schema Table: World

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

A country is big if:

it has an area of at least three million (i.e., 3000000 km2), or it has a population of at least twenty-five million (i.e., 25000000). Write a solution to find the name, population, and area of the big countries.

Return the result table in any order.

The result format is in the following example.

Example 1:

Input: World table: +-------------+-----------+---------+------------+--------------+ | name | continent | area | population | gdp | +-------------+-----------+---------+------------+--------------+ | Afghanistan | Asia | 652230 | 25500100 | 20343000000 | | Albania | Europe | 28748 | 2831741 | 12960000000 | | Algeria | Africa | 2381741 | 37100000 | 188681000000 | | Andorra | Europe | 468 | 78115 | 3712000000 | | Angola | Africa | 1246700 | 20609294 | 100990000000 | +-------------+-----------+---------+------------+--------------+ Output: +-------------+------------+---------+ | name | population | area | +-------------+------------+---------+ | Afghanistan | 25500100 | 652230 | | Algeria | 37100000 | 2381741 | +-------------+------------+---------+

In [1]:

```
%%writefile word.csv

| name        | continent | area    | population | gdp          |

| Afghanistan | Asia      | 652230  | 25500100   | 20343000000  |
| Albania     | Europe    | 28748   | 2831741    | 12960000000  |
| Algeria     | Africa    | 2381741 | 37100000   | 188681000000 |
| Andorra     | Europe    | 468     | 78115      | 3712000000   |
| Angola      | Africa    | 1246700 | 20609294   | 100990000000 |
```

```
Overwriting word.csv
```

In [2]:

```
import pandas as pd

world = pd.read_table('word.csv', sep=r' *\| *', engine='python')

world.drop(columns=['Unnamed: 0','Unnamed: 6'], axis=1, inplace=True)
world
```

Out[2]:

	name	continent	area	population	gdp
0	Afghanistan	Asia	652230	25500100	20343000000
1	Albania	Europe	28748	2831741	12960000000
2	Algeria	Africa	2381741	37100000	188681000000
3	Andorra	Europe	468	78115	3712000000
4	Angola	Africa	1246700	20609294	100990000000

In [3]:

```
import pandas as pd

def big_countries(world: pd.DataFrame) -> pd.DataFrame:
    op = pd.DataFrame(columns=['name', 'population', 'area'])
    for index, row in world.iterrows():
        if row['area'] >= 3000000 or row['population'] >= 25000000:
            op = op.append({'name': row['name'], 'population': row['population'], 'area':
    return op
```


In [4]:

```
big_countries(world)
```

```
C:\Users\shaha\AppData\Local\Temp\ipykernel_25544\3103732240.py:7: FutureW
arning: The frame.append method is deprecated and will be removed from pan
das in a future version. Use pandas.concat instead.
  op = op.append({'name': row['name'], 'population': row['population'], 'a
rea': row['area']},ignore_index=True)
C:\Users\shaha\AppData\Local\Temp\ipykernel_25544\3103732240.py:7: FutureW
arning: The frame.append method is deprecated and will be removed from pan
das in a future version. Use pandas.concat instead.
  op = op.append({'name': row['name'], 'population': row['population'], 'a
rea': row['area']},ignore_index=True)
```

Out[4]:

	name	population	area
0	Afghanistan	25500100	652230
1	Algeria	37100000	2381741

Problem 2

Write a solution to find the ids of products that are both low fat and recyclable.

Return the result table in any order.

In [5]:

```
%%writefile Products.csv


| product_id  | low_fats | recyclable |

| 0           | Y        | N          |
| 1           | Y        | Y          |
| 2           | N        | Y          |
| 3           | Y        | Y          |
| 4           | N        | N          |
```

```
Overwriting Products.csv
```

In [6]:

```
Products=pd.read_table('Products.csv',sep=r' *\| *',engine='python')
Products.drop(columns=['Unnamed: 0','Unnamed: 4'],axis=1, inplace=True)
Products
```

Out[6]:

	product_id	low_fats	recyclable
0	0	Y	N
1	1	Y	Y
2	2	N	Y
3	3	Y	Y
4	4	N	N

In [7]:

```
import pandas as pd

def find_products(Products: pd.DataFrame) -> pd.DataFrame:
    op=[]
    for index, row in Products.iterrows():

        if row['low_fats'] == 'Y' and row['recyclable'] == 'Y':
            op.append({'product_id': row['product_id']})
    return pd.DataFrame(op)
```

In [8]:

```
find_products(Products)
```

Out[8]:

	product_id
0	1
1	3

Problem 3

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

Return the result table in any order.

In [9]:

```
%%writefile customers.csv

| id | name  |

| 1  | Joe   |
| 2  | Henry |
| 3  | Sam   |
| 4  | Max   |
```

```
Overwriting customers.csv
```

In [10]:

```
customers=pd.read_table('customers.csv',sep=r' *\| *', engine='python')
customers.drop(columns=['Unnamed: 0','Unnamed: 3'],axis=1, inplace=True)
customers
```

Out[10]:

	id	name
0	1	Joe
1	2	Henry
2	3	Sam
3	4	Max

In [11]:

```
%%writefile orders.csv

| id | customerId |

| 1  | 3          |
| 2  | 1          |
```

```
Overwriting orders.csv
```

In [12]:

```
orders=pd.read_table('orders.csv',sep=r' *\| *', engine='python')
orders.drop(columns=['Unnamed: 0','Unnamed: 3'],axis=1, inplace=True)
orders
```

Out[12]:

	id	customerId
0	1	3
1	2	1

In [13]:

```
cust_order = customers.join(orders.set_index('customerId'), on='id', how='outer', rsuffix
cust_order
```

Out[13]:

	id	name	id_order
0	1	Joe	2.0
1	2	Henry	NaN
2	3	Sam	1.0
3	4	Max	NaN

In [14]:

```
import pandas as pd

def find_customers(customers: pd.DataFrame, orders: pd.DataFrame) -> pd.DataFrame:
    cust_order = customers.join(orders.set_index('customerId'), on='id', how='outer', rsu
    selected_customers = cust_order[cust_order['id_order'].isnull()]

    return selected_customers[['name']].rename(columns={'name': 'Customers'})
```

In [15]:

```
find_customers(customers,orders)
```

Out[15]:

	Customers
1	Henry
3	Max

Problem 4

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

The result format is in the following example.

In [16]:

```
%%writefile views.csv

| article_id | author_id | viewer_id | view_date  |

| 126        | 17        | 62        | 2019-07-02 |
| 149        | 42        | 22        | 2019-06-23 |
| 138        | 39        | 33        | 2019-07-26 |
| 165        | 42        | 74        | 2019-07-05 |
| 48         | 95        | 91        | 2019-06-28 |
| 151        | 30        | 19        | 2019-06-22 |
| 195        | 76        | 65        | 2019-07-24 |
| 67         | 4         | 91        | 2019-06-01 |
| 91         | 55        | 13        | 2019-07-11 |
| 126        | 17        | 77        | 2019-06-14 |
| 142        | 38        | 100       | 2019-07-07 |
| 148        | 63        | 48        | 2019-07-21 |
| 156        | 24        | 8         | 2019-07-29 |
| 63         | 73        | 30        | 2019-06-16 |
| 32         | 93        | 66        | 2019-07-23 |
| 143        | 10        | 38        | 2019-07-31 |
| 171        | 82        | 2         | 2019-07-25 |
| 88         | 61        | 67        | 2019-07-02 |
| 40         | 92        | 67        | 2019-06-03 |
| 23         | 18        | 39        | 2019-07-20 |
| 51         | 41        | 10        | 2019-07-17 |
| 76         | 10        | 65        | 2019-07-13 |
| 123        | 50        | 23        | 2019-07-06 |
| 77         | 76        | 1         | 2019-07-02 |
| 176        | 25        | 39        | 2019-06-01 |
| 4          | 34        | 100       | 2019-06-26 |
| 175        | 82        | 48        | 2019-06-04 |
| 109        | 47        | 65        | 2019-07-19 |
| 168        | 67        | 46        | 2019-07-12 |
| 129        | 49        | 74        | 2019-06-09 |
| 162        | 60        | 89        | 2019-06-18 |
| 40         | 92        | 24        | 2019-07-21 |
| 108        | 19        | 17        | 2019-07-27 |
| 68         | 83        | 19        | 2019-06-21 |
| 22         | 62        | 26        | 2019-07-29 |
| 190        | 9         | 13        | 2019-06-22 |
| 50         | 57        | 95        | 2019-06-21 |
| 47         | 49        | 29        | 2019-06-05 |
| 155        | 29        | 87        | 2019-06-25 |
| 158        | 25        | 57        | 2019-07-14 |
| 101        | 24        | 7         | 2019-07-24 |
| 189        | 43        | 40        | 2019-07-21 |
| 148        | 63        | 49        | 2019-06-18 |
| 150        | 75        | 73        | 2019-06-27 |
| 11         | 53        | 56        | 2019-07-11 |
| 131        | 65        | 70        | 2019-06-10 |
| 124        | 21        | 25        | 2019-07-12 |
| 80         | 8         | 33        | 2019-06-05 |
| 129        | 49        | 81        | 2019-08-01 |
| 172        | 2         | 33        | 2019-06-23 |
| 21         | 3         | 63        | 2019-07-31 |
| 104        | 97        | 92        | 2019-07-25 |
| 46         | 81        | 8         | 2019-06-13 |
| 169        | 30        | 50        | 2019-06-12 |
| 70         | 93        | 59        | 2019-06-01 |
```

7	15	73	2019-06-25
146	53	9	2019-07-07
114	83	67	2019-06-16
32	93	78	2019-07-20
18	7	47	2019-07-17
78	87	92	2019-07-13
39	93	62	2019-07-14
180	43	8	2019-06-27
105	81	45	2019-07-10
184	7	56	2019-07-16
58	93	58	2019-07-27
176	25	40	2019-07-23
48	95	71	2019-07-17
186	28	70	2019-06-14
85	74	46	2019-07-03
132	4	49	2019-06-24
103	11	84	2019-06-15
56	11	63	2019-06-07
111	71	51	2019-07-07
2	47	27	2019-07-22
133	40	85	2019-06-02
23	18	3	2019-07-07
130	2	19	2019-07-06
142	38	90	2019-07-06
133	40	1	2019-07-20
131	65	64	2019-07-26
15	37	21	2019-07-17
111	71	68	2019-06-16
84	48	70	2019-07-08
133	40	14	2019-06-13
31	85	80	2019-06-01
183	13	87	2019-07-16
114	83	58	2019-06-08
40	92	87	2019-07-08
122	36	91	2019-07-02
135	96	47	2019-07-26
96	92	73	2019-06-22
97	23	14	2019-06-11
83	74	85	2019-07-20
143	10	2	2019-06-08
193	85	91	2019-06-22
56	11	55	2019-06-25
20	19	75	2019-07-19
70	93	36	2019-07-01
168	67	22	2019-06-06
71	27	45	2019-06-14
176	25	92	2019-06-05
198	40	96	2019-07-08
168	67	51	2019-07-28
78	87	5	2019-07-18
154	3	12	2019-06-19
168	67	18	2019-06-26
67	4	37	2019-06-10
173	73	18	2019-06-22
162	60	83	2019-07-24
37	83	36	2019-07-26
127	19	88	2019-07-26
199	71	67	2019-07-06
12	34	2	2019-06-22
45	16	63	2019-06-19
139	29	37	2019-07-30

179	64	65	2019-07-02
9	55	2	2019-06-20
16	94	81	2019-07-08
66	32	35	2019-07-19
168	67	90	2019-07-08
147	99	60	2019-06-17
42	7	63	2019-07-18
18	7	79	2019-07-06
113	61	3	2019-07-24
129	49	80	2019-06-07
51	41	44	2019-07-18
102	48	96	2019-07-24
122	36	80	2019-07-16
193	85	87	2019-07-24
16	94	32	2019-07-01
20	19	45	2019-07-09
172	2	84	2019-06-15
87	60	60	2019-07-16
24	43	69	2019-07-30
57	57	11	2019-06-29
72	43	5	2019-06-06
173	73	45	2019-07-31
156	24	22	2019-06-21
130	2	43	2019-06-24
111	71	71	2019-06-23
12	34	13	2019-06-22
115	89	78	2019-06-28
12	34	31	2019-06-08
1	67	74	2019-07-23
93	98	41	2019-07-17
94	50	65	2019-07-03
50	57	25	2019-07-09
165	42	79	2019-07-01
37	83	51	2019-07-28
148	63	7	2019-07-09
119	72	5	2019-07-22
97	23	24	2019-06-13
133	40	95	2019-07-07
37	83	60	2019-07-13
110	48	97	2019-07-28
23	18	66	2019-06-22
183	13	25	2019-07-27
47	49	10	2019-06-03
83	74	57	2019-06-07
58	93	66	2019-06-30
193	85	93	2019-06-09
144	66	7	2019-06-07
192	10	100	2019-06-23
158	25	48	2019-07-04
125	76	72	2019-06-21
4	34	7	2019-07-02
19	78	5	2019-07-18
164	51	69	2019-07-04
9	55	35	2019-07-26
127	19	34	2019-06-04
133	40	65	2019-07-07
107	90	30	2019-06-28
173	73	96	2019-06-10
118	43	20	2019-06-27
141	15	69	2019-06-30
75	100	35	2019-08-01

141	15	35	2019-06-01
183	13	27	2019-06-21
131	65	33	2019-06-07
75	100	89	2019-06-02
104	97	61	2019-07-21
126	17	33	2019-06-22
50	57	75	2019-06-17
48	95	27	2019-06-05
147	99	13	2019-07-07
50	57	72	2019-06-30
102	48	2	2019-07-01
2	47	36	2019-06-22
76	10	2	2019-07-03
88	61	57	2019-07-12
94	50	15	2019-07-01
150	75	6	2019-07-01
173	73	92	2019-06-10
165	42	76	2019-07-20
101	24	59	2019-06-15
83	74	64	2019-07-30
160	7	51	2019-07-10
88	61	48	2019-07-11
23	18	16	2019-07-09
153	94	97	2019-07-27
161	26	6	2019-07-07
29	8	73	2019-06-02
13	93	78	2019-06-15
118	43	82	2019-06-01
123	50	9	2019-07-11
155	29	56	2019-07-20
45	16	67	2019-06-21
65	69	21	2019-07-01
185	30	98	2019-06-22
64	77	56	2019-06-21
147	99	31	2019-07-03
4	34	80	2019-07-20
86	47	61	2019-06-15
93	98	21	2019-07-22
80	8	81	2019-07-14
99	17	4	2019-07-09
140	18	37	2019-06-28
4	34	61	2019-06-12
107	90	22	2019-08-01
73	84	4	2019-07-23
126	17	17	2019-07-27
35	50	100	2019-06-29
101	24	88	2019-06-07
191	5	43	2019-07-21
63	73	63	2019-07-21
91	55	84	2019-07-27
185	30	80	2019-06-29
137	90	82	2019-07-30
113	61	14	2019-07-24
166	85	58	2019-06-04
132	4	99	2019-06-07
151	30	90	2019-07-04
72	43	46	2019-07-13
97	23	29	2019-08-01
182	28	65	2019-07-07
89	62	61	2019-07-15
130	2	48	2019-06-13

```
| 121          | 5           | 37          | 2019-07-06 |
| 109          | 47          | 2           | 2019-07-23 |
| 165          | 42          | 39          | 2019-06-25 |
| 66           | 32          | 30          | 2019-07-18 |
| 130          | 2           | 24          | 2019-07-09 |
| 134          | 20          | 73          | 2019-07-07 |
| 91           | 55          | 64          | 2019-07-11 |
| 123          | 50          | 28          | 2019-07-12 |
| 145          | 88          | 49          | 2019-06-23 |
| 13           | 93          | 33          | 2019-07-24 |
| 29           | 8           | 30          | 2019-07-27 |
| 11           | 53          | 1           | 2019-07-23 |
| 33           | 98          | 10          | 2019-07-27 |
| 23           | 18          | 39          | 2019-07-13 |
| 23           | 18          | 27          | 2019-07-23 |
| 144          | 66          | 71          | 2019-07-11 |
| 108          | 19          | 11          | 2019-06-09 |
| 41           | 23          | 66          | 2019-06-05 |
```

```
Overwriting views.csv
```

In [17]:

```
views=pd.read_table('views.csv', sep=' *\|* ', engine='python')
views.drop(columns=['Unnamed: 0','|'], axis=1, inplace=True)
views
```

Out[17]:

	article_id	author_id	viewer_id	view_date
0	126	17	62	2019-07-02
1	149	42	22	2019-06-23
2	138	39	3	2019-07-26
3	165	42	74	2019-07-05
4	48	95	91	2019-06-28
...	...	...	...	...
495	23	18	39	2019-07-13
496	23	18	27	2019-07-23
497	144	66	71	2019-07-11
498	108	19	11	2019-06-09
499	41	23	66	2019-06-05

500 rows × 4 columns

In [18]:

```
import pandas as pd

def article_views(views: pd.DataFrame) -> pd.DataFrame:
    views.drop_duplicates(inplace=True)
    selected_rows = views[views['author_id'] == views['viewer_id']]
    selected_rows_df = selected_rows[['author_id']].rename(columns={'author_id': 'id'})

    # Check if selected_rows_df is empty before sorting
    if not selected_rows_df.empty:
        selected_rows_df.sort_values(by='id', inplace=True)
        selected_rows_df.drop_duplicates(subset='id', keep='first', inplace=True)
    # Reset the index before returning
    selected_rows_df.reset_index(drop=True, inplace=True)
    return selected_rows_df
```

In [19]:

```
result=article_views(views)
result_ids = views['article_id'].iloc[result.index]
result
```

Out[19]:

	id
0	5
1	17
2	28
3	40
4	60
5	71

In [20]:

```
result_ids
```

Out[20]:

```
0    126
1    149
2    138
3    165
4     48
5    151
Name: article_id, dtype: int64
```

Problem 5

+---------------+---------+ | Column Name | Type | +---------------+---------+ | tweet_id | int | | content | varchar | +---------------+---------+ tweet_id is the primary key (column with unique values) for this table. 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.

In [21]:

```
%%writefile tweets.csv

| tweet_id | content                          |

| 1        | Vote for Biden                   |
| 2        | Let us make America great again! |
```

```
Writing tweets.csv
```

In [29]:

```
tweets=pd.read_table('tweets.csv', sep=r'\s*\|\s*', engine='python',quoting=3)
tweets.drop(columns=['Unnamed: 0','Unnamed: 3'],axis=1, inplace=True)
tweets
```

Out[29]:

	tweet_id	content
0	1	Vote for Biden
1	2	Let us make America great again!

In this code, the quoting parameter is set to 3, which is csv.QUOTE_NONE. This setting treats quotes as regular characters and should prevent the issue of quotes being ignored when a multi-character delimiter is used.

In [30]:

```
import pandas as pd

def invalid_tweets(tweets: pd.DataFrame) -> pd.DataFrame:
    invalid_tweets_df = []
    for index, row in tweets.iterrows():
        if len(row['content']) > 15:
            invalid_tweets_df.append(row['tweet_id'])

    return pd.DataFrame({'tweet_id': invalid_tweets_df})
```

In [31]:

```
invalid_tweets(tweets)
```

Out[31]:

	tweet_id
0	2

Problem 6

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| employee_id | int     |
| name        | varchar |
| salary      | int     |
+-------------+---------+
employee_id is the primary key (column with unique values) for this table.
Each row of this table indicates the employee ID, employee name, and salary.


Write a solution to calculate the bonus of each employee. The bonus of an employee is
100% of their salary if the ID of the employee is an odd number and the employee's name
does not start with the character 'M'. The bonus of an employee is 0 otherwise.

Return the result table ordered by employee_id
```

In [32]:

```
%%writefile employees.csv

| employee_id | name    | salary |

| 2           | Meir    | 3000   |
| 3           | Michael | 3800   |
| 7           | Addilyn | 7400   |
| 8           | Juan    | 6100   |
| 9           | Kannon  | 7700   |
```

```
Writing employees.csv
```


In [39]:

```
employees=pd.read_table('employees.csv', sep=' *\|* ', engine='python')
employees.drop(columns=['Unnamed: 0','|'],axis=1, inplace=True)
employees
```

Out[39]:

	employee_id	name	salary
0	2	Meir	3000
1	3	Michael	3800
2	7	Addilyn	7400
3	8	Juan	6100
4	9	Kannon	7700

In [72]:

```
import pandas as pd

def calculate_special_bonus(employees: pd.DataFrame) -> pd.DataFrame:

    for index, row in employees.iterrows():
        if row['employee_id'] %2 ==0 or row['name'].startswith('M'):
            employees.at[index, 'bonus'] = 0
        else:
            employees.at[index, 'bonus'] = row['salary']
    employees_result = employees[['employee_id', 'bonus']]
    return employees_result.sort_values('employee_id',ascending=True)
```

In [73]:

```
calculate_special_bonus(employees)
```

Out[73]:

	employee_id	bonus
0	2	0.0
1	3	0.0
2	7	7400.0
3	8	0.0
4	9	7700.0

problem 7 Fix Names in a Table

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

In [74]:

```
%%writefile users.csv
| user_id | name  |

| 1       | aLice |
| 2       | bOB   |
```

```
Writing users.csv
```

In [77]:

```
users=pd.read_table('users.csv', sep=' *\|* ', engine='python')
users.drop(columns=['Unnamed: 0','|'], axis=1, inplace=True)
users
```

Out[77]:

	user_id	name
0	1	aLice
1	2	bOB

In [94]:

```
import pandas as pd

def fix_names(users: pd.DataFrame) -> pd.DataFrame:
    users['name']=users['name'].str.title()
    return users.sort_values('user_id', ascending=True)
```

In [95]:

```
fix_names(users)
```

Out[95]:

	user_id	name
0	1	Alice
1	2	Bob

Problem 8 Find Users With Valid E-Mails

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

In [139]:

```
%%writefile users.csv

| user_id | name      | mail                    |

| 1       | Winston   | winston@leetcode.com    |
| 2       | Jonathan  | jonathanisgreat         |
| 3       | Annabelle | bella-@leetcode.com     |
| 4       | Sally     | sally.come@leetcode.com |
| 5       | Marwan    | quarz#2020@leetcode.com
| 6       | David     | david69@gmail.com       |
| 7       | Shapiro   | .shapo@leetcode.com     |
```

```
Overwriting users.csv
```

In [140]:

```
users=pd.read_table('users.csv', sep=' *\|* ', engine='python')
users.drop(columns=['|','Unnamed: 0'], axis=1, inplace=True)
users
```

Out[140]:

	user_id	name	mail
0	1	Winston	winston@leetcode.com
1	2	Jonathan	jonathanisgreat
2	3	Annabelle	bella-@leetcode.com
3	4	Sally	sally.come@leetcode.com
4	5	Marwan	quarz#2020@leetcode.com
5	6	David	david69@gmail.com
6	7	Shapiro	.shapo@leetcode.com


```
import pandas as pd
import regex as re

def valid_emails(users: pd.DataFrame) -> pd.DataFrame:
    valid_users=users.copy()
    for index, row in valid_users.iterrows():
        email=row['mail']
        if not re.match(r'^[A-Za-z][A-Za-z0-9_.-]*@leetcode\.com$', email):
            valid_users.drop(index, axis=0, inplace=True)
    return valid_users
```

In [143]:

```
import pandas as pd
import regex as re

def valid_emails(users: pd.DataFrame) -> pd.DataFrame:
    valid_mail=users[users['mail'].str.match(r'^[A-Za-z][A-Za-z0-9_.-]*@leetcode\.com$')]
    return valid_mail
```

In [144]:

```
valid_emails(users)
```

Out[144]:

	user_id	name	mail
0	1	Winston	winston@leetcode.com
2	3	Annabelle	bella-@leetcode.com
3	4	Sally	sally.come@leetcode.com

In [ ]:

In [ ]: