Python Pandas equivalent SQL Queries

In [21]:
```
#Import necessary libraries and load the data
import pandas as pd

customers = pd.read_csv('Mall_Customers.csv')
```

SELECT

In [22]:
```
#SQL Statement (Simple SELECT)
'''SELECT * FROM customers'''

#Pandas Equivalant
customers
```

Out[22]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

LIMIT

In [23]:
```
#SQL Statement (SELECT with limited number of records)
'''SELECT * FROM customers LIMIT 7'''

#Pandas Equivalant
customers.head(7)
```

Out[23]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
0	1	Male	19	15	39
1	2	Male	21	15	81

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6

SELECT...WHERE

In [24]:
```
#SQL Statement (SELECT and WHERE on single condition)
'''SELECT CustomerID FROM customers WHERE Annual_Income_K_Dollars = 16'''

#Pandas Equivalant
customers[customers.Annual_Income_K_Dollars == 16].CustomerID
```

Out[24]:
```
2    3
3    4
Name: CustomerID, dtype: int64
```

SELECT...WHERE (Multiple conditions, All columns)

In [25]:
```
#SQL Statement (SELECT and WHERE on multiple conditions such that all the columns will
'''SELECT * FROM customers WHERE Annual_Income_K_Dollars = 16 AND Spending_Score_1_to_1

#Pandas Equivalant
customers[(customers.Annual_Income_K_Dollars == 16) & (customers.Spending_Score_1_to_10
```

Out[25]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
2	3	Female	20	16	6

SELECT...WHERE(Multiple consitions, Subset of columns)

In [26]:
```
#SQL Statement (SELECT and WHERE on multiple conditions such that subset of columns wil
'''SELECT Genre, Age, Spending_Score_1_to_100 FROM customers WHERE Annual_Income_K_Doll
Spending_Score_1_to_100 = 6'''

#Pandas Equivalant
customers[(customers.Annual_Income_K_Dollars == 16) & (customers.Spending_Score_1_to_10
```

Out[26]:

	Genre	Age	Spending_Score_1_to_100
2	Female	20	6

AGGREGATE

In [27]:
```
#SQL Statement (Operation using AGGREGATE functions like MEAN, MIN and MAX)
'''SELECT mean(Age), max(Age), min(Age) FROM customers'''
```

```
#Pandas Equivalant
customers.agg({'Age': ['mean', 'max', 'min']})
```

Out[27]:

	Age
mean	38.85
max	70.00
min	18.00

DISTINCT

In [28]:

```
#SQL Statement (Finding distinct values of a column)
'''SELECT DISTINCT Annual_Income_K_Dollars FROM customers'''

#Pandas Equivalant
customers.Annual_Income_K_Dollars.unique()
```

Out[28]:

```
array([ 15,  16,  17,  18,  19,  20,  21,  23,  24,  25,  28,  29,  30,
        33,  34,  37,  38,  39,  40,  42,  43,  44,  46,  47,  48,  49,
        50,  54,  57,  58,  59,  60,  61,  62,  63,  64,  65,  67,  69,
        70,  71,  72,  73,  74,  75,  76,  77,  78,  79,  81,  85,  86,
        87,  88,  93,  97,  98,  99, 101, 103, 113, 120, 126, 137],
      dtype=int64)
```

ORDERBY (Ascending Order)

In [29]:

```
#SQL Statement (Filtering on Genre called Female & Ordering By Spending_Score_1_to_100)
'''SELECT * FROM customers WHERE Genre = "Female" ORDER BY Spending_Score_1_to_100'''

#Pandas Equivalant
customers[customers.Genre == 'Female'].sort_values('Spending_Score_1_to_100')
```

Out[29]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
140	141	Female	57	75	5
22	23	Female	46	25	5
2	3	Female	20	16	6
6	7	Female	35	18	6
136	137	Female	44	73	7
...	...	...	...	...	...
163	164	Female	31	81	93
7	8	Female	23	18	94
167	168	Female	33	86	95
19	20	Female	35	23	98
11	12	Female	35	19	99

112 rows × 5 columns

ORDERBY (Descending Order)

In [30]:
```
#SQL Statement (Filtering on Genre called Female & Ordering By Spending_Score_1_to_100
'''SELECT * FROM customers WHERE Genre = "Female" ORDER BY Spending_Score_1_to_100 DESC

#Pandas Equivalant
customers[customers.Genre == 'Female'].sort_values('Spending_Score_1_to_100', ascending
```

Out[30]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
11	12	Female	35	19	99
19	20	Female	35	23	98
167	168	Female	33	86	95
7	8	Female	23	18	94
163	164	Female	31	81	93
...	...	...	...	...	...
136	137	Female	44	73	7
6	7	Female	35	18	6
2	3	Female	20	16	6
22	23	Female	46	25	5
140	141	Female	57	75	5

112 rows × 5 columns

GROUPBY (Count)

In [31]:
```
#SQL Statement (Grouping by Genre & Age and counting each occurence of it)
'''SELECT Genre, Age, count(*) FROM customers GROUPBY Genre, Age'''

#Pandas Equivalant
customers.groupby(['Genre', 'Age']).size().to_frame('Count').reset_index()
```

Out[31]:

	Genre	Age	Count
0	Female	18	1
1	Female	19	2
2	Female	20	2
3	Female	21	4
4	Female	22	2
...	...	...	...
82	Male	66	1

	Genre	Age	Count
83	Male	67	3
84	Male	68	1
85	Male	69	1
86	Male	70	2

87 rows × 3 columns

GROUPBY (Count and Descending order on a column)

In [32]:
```
# SQL Statement (Grouping by Genre & Age and counting each occurence of it such that Ag
'''SELECT Genre, Age, count(*) FROM customers GROUPBY Genre, Age ORDER BY Age DESC'''

#Pandas Equivalant
customers.groupby(['Genre', 'Age']).size().to_frame('Count').reset_index().sort_values(
```

Out[32]:

	Genre	Age	Count
86	Male	70	2
85	Male	69	1
42	Female	68	2
84	Male	68	1
83	Male	67	3
...	...	...	...
2	Female	20	2
1	Female	19	2
44	Male	19	6
0	Female	18	1
43	Male	18	3

87 rows × 3 columns

HAVING

In [33]:
```
#SQL Statement (Additional filter on Grouped Data by making use of HAVING)
'''SELECT Age, count(*) FROM customers GROUPBY Age HAVING count(*) < 3'''

#Pandas Equivalant
customers.groupby('Age').filter(lambda x: len(x) < 3).groupby('Age').size()
```

Out[33]:
```
Age
26    2
41    2
42    2
44    2
```

```
51    2
52    2
53    2
55    1
56    1
57    2
58    2
63    2
64    1
65    2
66    2
69    1
70    2
dtype: int64
```

IN

In [34]:

```
#SQL Statement (Filter records based on values which are available in the given list )
'''SELECT * FROM customers WHERE Age IN (20, 30, 40)'''

#Pandas Equivalant
customers[customers.Age.isin([20,30,40])]
```

Out[34]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
2	3	Female	20	16	6
9	10	Female	30	19	72
17	18	Male	20	21	66
28	29	Female	40	29	31
37	38	Female	30	34	73
39	40	Female	20	37	75
77	78	Male	40	54	48
93	94	Female	40	60	40
99	100	Male	20	61	49
122	123	Female	40	69	58
127	128	Male	40	71	95
134	135	Male	20	73	5
157	158	Female	30	78	78
159	160	Female	30	78	73
170	171	Male	40	87	13
175	176	Female	30	88	86
185	186	Male	30	99	97
199	200	Male	30	137	83

NOT IN

In [35]:
```
#SQL Statement (Filter records based on values which are NOT available in the given lis
'''SELECT * FROM customers WHERE Age NOT IN (20, 30, 40)'''

#Pandas Equivalant
customers[~customers.Age.isin([20,30,40])]
```

Out[35]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
0	1	Male	19	15	39
1	2	Male	21	15	81
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
...	...	...	...	...	...
194	195	Female	47	120	16
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18

182 rows × 5 columns

Top N Observations

In [36]:
```
#SQL Statement (Identify Top 10 records)
'''SELECT * FROM customers ORDER BY Spending_Score_1_to_100 DESC LIMIT 10'''

#Pandas Equivalant
customers.nlargest(10, columns='Spending_Score_1_to_100')
```

Out[36]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
11	12	Female	35	19	99
19	20	Female	35	23	98
145	146	Male	28	77	97
185	186	Male	30	99	97
127	128	Male	40	71	95
167	168	Female	33	86	95
7	8	Female	23	18	94
141	142	Male	32	75	93
163	164	Female	31	81	93
33	34	Male	18	33	92

Top N Observations with Offset

In [37]:
```
#SQL Statement (Identify Next Top 10 records)
'''SELECT * FROM customers ORDER BY Spending_Score_1_to_100 DESC LIMIT 10 OFFSET 10'''

#Pandas Equivalant
customers.nlargest(20, columns='Spending_Score_1_to_100').tail(10)
```

Out[37]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
41	42	Male	24	38	92
173	174	Male	36	87	92
123	124	Male	39	69	91
193	194	Female	38	113	91
149	150	Male	34	78	90
179	180	Male	35	93	90
155	156	Female	27	78	89
135	136	Female	29	73	88
151	152	Male	39	78	88
183	184	Female	29	98	88

UNION ALL AND UNION

In [38]:
```
shop_customers = pd.read_csv('Shop_Customers.csv')

#SQL Statement (Union two Tables)
'''SELECT * FROM customers WHERE Annual_Income_K_Dollars > 50 UNION ALL SELECT * FROM s

#Pandas Equivalant
pd.concat([customers[customers.Annual_Income_K_Dollars > 50], shop_customers[shop_custo

#If wants to mimic UNION operation then just (append) chain the entire operation with d
```

Out[38]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
74	75	Male	59	54	47
75	76	Male	26	54	54
76	77	Female	45	54	53
77	78	Male	40	54	48
78	79	Female	23	54	52
...	...	...	...	...	...
15	216	Male	64	26	79
16	217	Female	67	26	35

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
17	218	Male	67	27	66
18	219	Male	20	28	29
19	220	Female	23	24	98

146 rows × 5 columns

JOIN

In [39]:
```
transactions = pd.read_csv('Mall_Customers_Transactions.csv')

#SQL Statement (Join two Tables)
'''SELECT * FROM customers c JOIN transactions t ON c.CustomerID = t.CustID WHERE c.Gen

#Pandas Equivalant
customers.merge(transactions[customers.Genre == 'Female'], left_on='CustomerID', right_
```

Out[39]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100	CustID	Transaction_
0	3	Female	20	16	6	3	
1	4	Female	23	16	77	4	
2	5	Female	31	17	40	5	
3	6	Female	22	17	76	6	
4	7	Female	35	18	6	7	
...	...	...	...	...	...	...	
107	192	Female	32	103	69	192	
108	194	Female	38	113	91	194	
109	195	Female	47	120	16	195	
110	196	Female	35	120	79	196	
111	197	Female	45	126	28	197	

112 rows × 8 columns

INSERT

In [40]:
```
#SQL Statement (Insert a new record in the table)
'''INSERT INTO customers VALUES(401, 'Male', 50, 30, 20) '''

#Pandas Equivalant
customers = customers.append({'CustomerID':401, 'Genre':'Male', 'Age':50, 'Annual_Incom
customers
```

Out[40]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83
200	401	Male	50	30	20

201 rows × 5 columns

UPDATE

In [41]:

```
#SQL Statement (Update an existing record in the table)
'''UPDATE customers SET Spending_Score_1_to_100 = 7 WHERE Spending_Score_1_to_100 = 6''

#Pandas Equivalant
customers.loc[customers['Spending_Score_1_to_100'] == 6, 'Spending_Score_1_to_100'] = 7
```

In [42]:

```
#customers = pd.read_csv('Mall_Customers.csv')

#Currently two records with Spending_Score_1_to_100 == 6
customers[(customers.Spending_Score_1_to_100 == 7)]
```

Out[42]:

	CustomerID	Genre	Age	Annual_Income_K_Dollars	Spending_Score_1_to_100
2	3	Female	20	16	7
6	7	Female	35	18	7
136	137	Female	44	73	7

DELETE

In [43]:

```
#SQL Statement (Delete an existing record in the table)
'''DELETE FROM customers WHERE Spending_Score_1_to_100 = 7'''

#Pandas Equivalant
customers = customers.drop(customers[customers.Spending_Score_1_to_100 == 7].index)
```