DATA 311 - Fall 2020

Assignment #4 - Due Friday, October 30 by midnight

Load all of the sales data from the sales_data.zip file provided into our Store database.

- Make sure to start with a fresh, empty copy of the database.
- Destory the sales file we were using for testing in class - only use the new data provided
- Make sure to load the data in chronological order, so that we will all end up with the same values for order_id and cust_id
- The data provided is for all of 2019, and the first 9 months of 2020 (21 files total).
- The data was generated in such a way that our total sales every month are usually, but not always, increasing. You can use this fact as a sanity check to make sure the data was loaded correctly.
- I will be providing new sales data eventually, so make sure the loading process is seamless and easy, and make sure to thoroughly test it.
- When loading a file, you might want to have your code move that file into a different directory once it is succesfully loaded, so that you don't accidentally try to load it again later. Let me know if you need help with that!

In [1]:
```
import sqlite3
import Store
import pandas as pd
```

In [2]:
```
conn = sqlite3.connect('Store.db')
curs = conn.cursor()
curs.execute("PRAGMA foreign_keys=ON;")
```

Out[2]:
```
<sqlite3.Cursor at 0x7f3626173ea0>
```

In [3]:
```
Store.Rebuild()
```

Out[3]:
```
1
```

In [4]:
```
def GetCustomerID(first_name,last_name,address,zip_code):
    '''Function will check if a record for customer exists.
        If so, return the customer id
        If multiple records are found, print a warning and return None
        If no record exists, create one and return the customer id.'''

    sql = """SELECT cust_id
                FROM tCust
                WHERE first_name = ?
                AND last_name = ?
                AND address = ?
                AND zip = ?;"""
    # Make sure to convert zip to string
    cust = pd.read_sql(sql, conn, params=(first_name,last_name,address,str(zip_code)))

    # There should only be at most, one result
    if len(cust) > 1:
```

```
        print('Found multiple customers: ' + str(len(cust)))
        return None

    # If the customer did not exist, then create it
    if len(cust) == 0:
        sql_insert = """INSERT INTO tCust (first_name,last_name,address,zip) VALUES (?,
        curs.execute(sql_insert, (first_name,last_name,address,str(zip_code)))
        cust = pd.read_sql(sql, conn, params=(first_name,last_name,address,str(zip_code

    return cust['cust_id'][0]

def GetOrderID(cust_id, day, month, year):

    # Check to see if an order already exists for this customer/day
    sql_check_order = """SELECT order_id
                            FROM tOrder
                            WHERE cust_id = ?
                            AND day = ?
                            AND month = ?
                            AND year = ?;"""
    order_id = pd.read_sql(sql_check_order, conn,
                           params=(cust_id, day, month, year))

    if len(order_id) == 0:
        # Enter the order
        sql_enter_order = """INSERT INTO tOrder (cust_id, day, month, year)
                                VALUES (?,?,?,?);"""
        curs.execute(sql_enter_order, (cust_id, day, month, year))
        order_id = pd.read_sql(sql_check_order, conn,
                               params =(cust_id, day, month, year))
    elif len(order_id)>1:
        # You might want to make this message a bit more informative
        print('WARNING! Multiple orders found...')
        return None
    else:
        print('Order information for customer ' + str(cust_id) +
              ' on ' + str(day) + '/' + str(month) + '/' + str(year)
             + ' already exists')

    return order_id['order_id'][0]
```

In [5]:
```
years = []
for x in range(1, 13):
    if x < 10:
        years.append("20190" + str(x))
    else:
        years.append("2019" + str(x))

for x in range(1, 10):
    years.append("20200" + str(x))
```

In [6]:
```
for y in years:
    filename = './data/Sales_' + y + '.csv'
    data = pd.read_csv(filename, dtype={'zip':str})

    # Append customer ids to the DataFrame

    cust = data[['first','last','addr','city','state','zip']].drop_duplicates()

    cust_id = []
```

```
    for row in cust.values:
        cust_id.append(GetCustomerID(row[0], row[1], row[2], row[5]))

    cust['cust_id'] = cust_id
    data_with_cust = data.merge(cust, on=['first','last','addr','zip'])

    # Append order ids to the DataFrame

    order_id = []
    orders  = data_with_cust[['cust_id', 'date']].drop_duplicates()
    orders[['year','month','day']] = orders['date'].str.split('-',expand=True)

    for row in orders.values:
        order_id.append(GetOrderID(row[0], row[4], row[3], row[2]))

    orders['order_id'] = order_id
    data_with_cust_order = data_with_cust.merge(orders, on=['cust_id', 'date'])

    COL_ORDER_ID = 17
    COL_PROD_ID = 7
    COL_QTY = 10

    sql = "INSERT INTO tOrderDetail VALUES(?,?,?)"
    for row in data_with_cust_order.values:
        curs.execute(sql, (row[COL_ORDER_ID], row[COL_PROD_ID], row[COL_QTY]))
```

In [7]:
```
conn.commit()
```

After doing so, answer the following questions:

---

1) Generate a summary, by month and year of how our store is performing.

Have your query return the following:

- year
- month
- Sales: total sales for the month
- NumOrders: number of orders placed for the month
- NumCust: number of distinct customers who made a purchase (i.e. only count the customer at most once per month)
- OrdersPerCust: average number of orders per customer (i.e. NumOrders/NumCust)
- SalesPerCust: average sales per customer (i.e. Sales/NumCust)
- SalesPerOrder: average sales per order (i.e. Sales/NumOrders)

The results should be grouped and sorted by year and month, in ascending order.

Keep in mind that you have data for all 12 months of 2019, and the first 9 months of 2020, so there should be 21 rows in your results. Also, watch out for integer division!

In [11]:
```
curs.execute("DROP VIEW IF EXISTS view_1;")
curs.execute("""CREATE VIEW view_1 AS
                SELECT year, month, sum(qty*unit_price) as Sales, count(order_id) as Num
                FROM tState
```

```
                    JOIN tZip USING(st)
                    JOIN tCust USING(zip)
                    JOIN tOrder USING(cust_id)
                    JOIN tOrderDetail USING(order_id)
                    JOIN tProd USING(prod_id)
                GROUP BY year, month
                ORDER BY year, month ASC;""")

pd.read_sql("""SELECT year, month, Sales, NumOrders, NumCust, (NumOrders * 1.0) / (NumC
               FROM view_1;""", conn)
```

Out[11]:

	year	month	Sales	NumOrders	NumCust	OrdersPerCust	SalesPerCust	SalesPerOrder
0	2019	1	68464.61	327	85	3.847059	805.466000	209.371896
1	2019	2	55560.32	274	73	3.753425	761.100274	202.774891
2	2019	3	100491.07	424	85	4.988235	1182.247882	237.007241
3	2019	4	110661.05	528	95	5.557895	1164.853158	209.585322
4	2019	5	125623.57	586	97	6.041237	1295.088351	214.374693
5	2019	6	137173.59	665	109	6.100917	1258.473303	206.276075
6	2019	7	158080.03	709	103	6.883495	1534.757573	222.961961
7	2019	8	228577.44	999	126	7.928571	1814.106667	228.806246
8	2019	9	229094.40	1106	126	8.777778	1818.209524	207.137794
9	2019	10	354471.44	1712	151	11.337748	2347.492980	207.051075
10	2019	11	313319.82	1340	135	9.925926	2320.887556	233.820761
11	2019	12	584023.71	2402	170	14.129412	3435.433588	243.140595
12	2020	1	480742.54	2375	166	14.307229	2896.039398	202.417912
13	2020	2	562773.44	2610	172	15.174419	3271.938605	215.622008
14	2020	3	908514.52	4097	209	19.602871	4346.959426	221.751164
15	2020	4	743491.76	3218	177	18.180791	4200.518418	231.041566
16	2020	5	1066398.25	4696	197	23.837563	5413.189086	227.086510
17	2020	6	1403832.02	6355	228	27.872807	6157.157982	220.901970
18	2020	7	1542354.28	7382	243	30.378601	6347.136955	208.934473
19	2020	8	2197253.30	9827	244	40.274590	9005.136475	223.593498
20	2020	9	106506.53	471	88	5.352273	1210.301477	226.128514

---

2) Get our total sales for all states (50 + DC and PR, so 52 records total) for **January 2019 only**.

Have your query return:

- st: The state abbreviation
- state: The name of the state
- Sales: The total sales in that state

Order the results by the state abbreviation, in ascending order.

Make sure that all states are returned even if they had no sales. In that case, have the query return 0 instead of NaN or Null.

In [33]:
```
curs.execute("DROP VIEW IF EXISTS vSales;")
curs.execute("""CREATE VIEW vSales AS
                SELECT st, state, sum(qty*unit_price) as Sales
                FROM tState
                    JOIN tZip USING(st)
                    JOIN tCust USING(zip)
                    JOIN tOrder USING(cust_id)
                    JOIN tOrderDetail USING(order_id)
                    JOIN tProd USING(prod_id)
                WHERE year = '2019' AND month = '1'
                GROUP BY st
                ORDER BY st;""")

curs.execute("DROP VIEW IF EXISTS vStates;")
curs.execute("""CREATE VIEW vStates AS
                SELECT DISTINCT st as st1, state as state1
                FROM tState
                JOIN tZip USING(st);""")

pd.read_sql("""SELECT st1, state1, IFNULL(Sales, 0) as Sales
                FROM vStates as A
                    LEFT JOIN vSales as B
                    ON A.st1 = B.st AND A.state1 = B.state
                ORDER BY st1;""", conn)
```

Out[33]:

	st1	state1	Sales
0	AK	Alaska	0.00
1	AL	Alabama	2476.61
2	AR	Arkansas	597.43
3	AZ	Arizona	1959.23
4	CA	California	0.00
5	CO	Colorado	198.86
6	CT	Connecticut	223.68
7	DC	District of Columbia	1328.35
8	DE	Delaware	1650.19
9	FL	Florida	564.38
10	GA	Georgia	0.00
11	HI	Hawaii	3490.25
12	IA	Iowa	517.85
13	ID	Idaho	1683.08
14	IL	Illinois	0.00
15	IN	Indiana	856.41

	st1	state1	Sales
16	KS	Kansas	6005.12
17	KY	Kentucky	0.00
18	LA	Louisiana	2389.68
19	MA	Massachusetts	902.67
20	MD	Maryland	439.72
21	ME	Maine	523.48
22	MI	Michigan	191.83
23	MN	Minnesota	117.95
24	MO	Missouri	485.23
25	MS	Mississippi	4093.82
26	MT	Montana	275.90
27	NC	North Carolina	742.37
28	ND	North Dakota	1867.18
29	NE	Nebraska	4120.92
30	NH	New Hampshire	10.99
31	NJ	New Jersey	493.85
32	NM	New Mexico	4517.45
33	NV	Nevada	0.00
34	NY	New York	883.46
35	OH	Ohio	3009.38
36	OK	Oklahoma	196.77
37	OR	Oregon	0.00
38	PA	Pennsylvania	4171.87
39	PR	Puerto Rico	4088.15
40	RI	Rhode Island	652.00
41	SC	South Carolina	2372.45
42	SD	South Dakota	298.47
43	TN	Tennessee	0.00
44	TX	Texas	1524.81
45	UT	Utah	3174.09
46	VA	Virginia	71.91
47	VT	Vermont	450.45
48	WA	Washington	0.00

	st1	state1	Sales
49	WI	Wisconsin	2810.57
50	WV	West Virginia	1284.08
51	WY	Wyoming	751.67

---

3) Going back to question 1, you may have noticed that our sales were not very good last month!

Generate a list of all customers who did not place an order last month (September, 2020)

Have your query return:

- cust_id
- NumOrder: a count of the number of orders they placed last month (which should all be zero).

In [50]:
```
curs.execute("DROP VIEW IF EXISTS vPlacedOrder;")
curs.execute("""CREATE VIEW vPlacedOrder AS
                SELECT cust_id, count(order_id) as NumOrder
                FROM tState
                    JOIN tZip USING(st)
                    JOIN tCust USING(zip)
                    JOIN tOrder USING(cust_id)
                    JOIN tOrderDetail USING(order_id)
                    JOIN tProd USING(prod_id)
                WHERE year = 2020 AND month = 9
                GROUP BY cust_id
                ORDER BY cust_id;""")

curs.execute("DROP VIEW IF EXISTS vAllCust;")
curs.execute("""CREATE VIEW vAllCust AS
                SELECT cust_id as cust_id1 FROM tCust;""")

curs.execute("DROP VIEW IF EXISTS vOutput;")
curs.execute("""CREATE VIEW vOutput AS
               SELECT cust_id1, IFNULL(NumOrder, 0) as NumOrder
               FROM vAllCust as A
                   LEFT JOIN vPlacedOrder as B
                   ON A.cust_id1 = B.cust_id
               """)

pd.read_sql("""SELECT cust_id1 AS cust_id, NumOrder
               FROM vOutput
               WHERE NumOrder = 0""", conn)
```

Out[50]:

	cust_id	NumOrder
0	1	0
1	2	0
2	3	0
3	4	0
4	5	0

	cust_id	NumOrder
...	...	...
217	302	0
218	306	0
219	307	0
220	308	0
221	310	0

222 rows × 2 columns

---

4) Using the list of customers from the last question, add two new columns to the result containing 1) each customer's average sales for months 1 through 8 of 2020, and 2) their sales for September of 2019. Maybe we'll give that info to our sales team and see if we can do some marketing to those customers.

In [85]:
```
curs.execute("DROP VIEW IF EXISTS vPrev;")
curs.execute("""CREATE VIEW vPrev AS
                SELECT cust_id1 AS cust_id, NumOrder
                FROM vOutput
                WHERE NumOrder = 0""")

curs.execute("DROP VIEW IF EXISTS v2019Sales;")
curs.execute("""CREATE VIEW v2019Sales AS
                SELECT cust_id as cust19, AVG(qty*unit_price) as [AverageSales 2019]
                FROM tState
                    JOIN tZip USING(st)
                    JOIN tCust USING(zip)
                    JOIN tOrder USING(cust_id)
                    JOIN tOrderDetail USING(order_id)
                    JOIN tProd USING(prod_id)
                WHERE year = 2019 and month = 9
                GROUP BY cust_id
                ORDER BY cust_id;""")

curs.execute("DROP VIEW IF EXISTS v2018Sales;")
curs.execute("""CREATE VIEW v2018Sales AS
                SELECT cust_id as cust18, AVG(qty*unit_price) as [AverageSales 2018]
                FROM tState
                    JOIN tZip USING(st)
                    JOIN tCust USING(zip)
                    JOIN tOrder USING(cust_id)
                    JOIN tOrderDetail USING(order_id)
                    JOIN tProd USING(prod_id)
                WHERE year = 2020 AND (month = 1 OR month = 2 OR month = 3 OR month = 4
                GROUP BY cust_id
                ORDER BY cust_id;""")

pd.read_sql("""SELECT cust_id, IFNULL([AverageSales 2018], 0) as [AverageSales 2018], I
                FROM vPrev as A
                    LEFT JOIN v2018Sales as B
                    ON A.cust_id = B.cust18
```

```
                    LEFT JOIN v2019Sales as C
                    ON A.cust_id = C.cust19;""", conn)
```

Out[85]:

	cust_id	AverageSales 2018	AverageSales 2019
0	1	163.241333	69.473333
1	2	207.797097	89.625000
2	3	245.046607	275.316667
3	4	244.393038	0.000000
4	5	255.700478	0.000000
...	...	...	...
217	302	221.403415	0.000000
218	306	212.294670	0.000000
219	307	233.239200	0.000000
220	308	225.830710	0.000000
221	310	187.973458	0.000000

222 rows × 3 columns

---

5) What is our top selling product (in terms of dollars) so far?

Have your query return:

- prod_id
- prod_name
- total quantity sold, based on all the data we have in the database
- total sales, based on all the current data in the database

In [90]:
```
pd.read_sql("""SELECT prod_id, prod_name, sum(qty) as quantity, sum(qty*unit_price) as
                FROM tState
                    JOIN tZip USING(st)
                    JOIN tCust USING(zip)
                    JOIN tOrder USING(cust_id)
                    JOIN tOrderDetail USING(order_id)
                    JOIN tProd USING(prod_id)
                GROUP BY prod_id
                ORDER BY sales DESC;""",conn)
```

Out[90]:

	prod_id	prod_name	quantity	sales
0	329	Chainsaw	8951	4475410.49
1	328	Workbench	8524	2557200.00
2	327	Ladder	8757	700560.00
3	326	Drill	9011	621759.00
4	325	Toolbox	8569	428450.00

	prod_id	prod_name	quantity	sales
5	324	Wire	8884	328708.00
6	323	Hatchet	8339	300120.61
7	322	Monkeywrench	8895	266405.25
8	321	Axe	8836	247319.64
9	319	Paint	8713	174172.87
10	320	Saw	8294	174091.06
11	318	Hacksaw	8595	171814.05
12	315	Pliers	9094	145413.06
13	317	Level	8533	144975.67
14	316	Anvil	8906	144277.20
15	314	Mallet	8900	106800.00
16	312	Plane	8681	95404.19
17	313	Wrench	8398	92378.00
18	311	Hammer	9101	75265.27
19	310	Scraper	8306	66364.94
20	309	Chisel	9245	46132.55
21	306	Clamp	8884	26563.16
22	307	Sandpaper	8809	26427.00
23	308	Screwdriver	8772	26316.00
24	305	Bradawl	8357	16630.43
25	304	Bolt	8845	8845.00
26	303	Screw	8794	4397.00
27	301	Nail	8699	2174.75
28	302	Nut	8690	2172.50
29	300	Washer	8600	860.00

In [91]:
```
# Don't forget to close your connection when done!
conn.close()
```

In [ ]:
```
```