Assignment4_JY

October 29, 2020

0.1 DATA 311 - Fall 2020

0.2 ### 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!

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!

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

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

[2]:
```
<sqlite3.Cursor at 0x7ff9a5819ea0>
```

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

[3]:
```
1
```

[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]
```

[5]:
```
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]
```

[6]:
```
file = []
for i in range (1,13):
    if i < 10:
        file.append("20190" + str(i))
    else:
        file.append("2019" + str(i))
for i in range (1, 10):
    if i < 10:
        file.append("20200" + str(i))
```

[8]:
```
for x in file:
    filename = './data/Sales_' + x + '.csv'
    data=pd.read_csv(filename, dtype={'zip':str})

    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'])

    #GetOrderID
    order_id = []
    #split date
```

```
    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]))
```

```
Order information for customer 1 on 01/01/2019 already exists
Order information for customer 2 on 02/01/2019 already exists
Order information for customer 3 on 02/01/2019 already exists
Order information for customer 4 on 02/01/2019 already exists
Order information for customer 5 on 02/01/2019 already exists
Order information for customer 6 on 03/01/2019 already exists
Order information for customer 7 on 03/01/2019 already exists
Order information for customer 8 on 03/01/2019 already exists
Order information for customer 9 on 03/01/2019 already exists
Order information for customer 10 on 03/01/2019 already exists
Order information for customer 11 on 04/01/2019 already exists
Order information for customer 12 on 05/01/2019 already exists
Order information for customer 13 on 06/01/2019 already exists
Order information for customer 13 on 11/01/2019 already exists
Order information for customer 13 on 29/01/2019 already exists
Order information for customer 14 on 06/01/2019 already exists
Order information for customer 15 on 07/01/2019 already exists
Order information for customer 16 on 07/01/2019 already exists
Order information for customer 16 on 24/01/2019 already exists
Order information for customer 17 on 08/01/2019 already exists
Order information for customer 18 on 08/01/2019 already exists
Order information for customer 19 on 08/01/2019 already exists
Order information for customer 20 on 08/01/2019 already exists
Order information for customer 21 on 09/01/2019 already exists
Order information for customer 22 on 09/01/2019 already exists
Order information for customer 23 on 09/01/2019 already exists
Order information for customer 24 on 10/01/2019 already exists
Order information for customer 25 on 10/01/2019 already exists
Order information for customer 26 on 10/01/2019 already exists
```

```
Order information for customer 27 on 11/01/2019 already exists
Order information for customer 28 on 12/01/2019 already exists
Order information for customer 29 on 12/01/2019 already exists
Order information for customer 30 on 12/01/2019 already exists
Order information for customer 30 on 20/01/2019 already exists
Order information for customer 31 on 13/01/2019 already exists
Order information for customer 32 on 13/01/2019 already exists
Order information for customer 33 on 13/01/2019 already exists
Order information for customer 34 on 13/01/2019 already exists
Order information for customer 34 on 28/01/2019 already exists
Order information for customer 35 on 14/01/2019 already exists
Order information for customer 36 on 14/01/2019 already exists
Order information for customer 37 on 14/01/2019 already exists
Order information for customer 38 on 14/01/2019 already exists
Order information for customer 39 on 15/01/2019 already exists
Order information for customer 40 on 15/01/2019 already exists
Order information for customer 41 on 15/01/2019 already exists
Order information for customer 42 on 15/01/2019 already exists
Order information for customer 43 on 15/01/2019 already exists
Order information for customer 44 on 16/01/2019 already exists
Order information for customer 45 on 16/01/2019 already exists
Order information for customer 46 on 17/01/2019 already exists
Order information for customer 47 on 17/01/2019 already exists
Order information for customer 48 on 17/01/2019 already exists
Order information for customer 48 on 21/01/2019 already exists
Order information for customer 49 on 17/01/2019 already exists
Order information for customer 50 on 17/01/2019 already exists
Order information for customer 51 on 18/01/2019 already exists
Order information for customer 52 on 18/01/2019 already exists
Order information for customer 53 on 18/01/2019 already exists
Order information for customer 54 on 18/01/2019 already exists
Order information for customer 55 on 18/01/2019 already exists
Order information for customer 56 on 19/01/2019 already exists
Order information for customer 57 on 20/01/2019 already exists
Order information for customer 58 on 20/01/2019 already exists
Order information for customer 59 on 20/01/2019 already exists
Order information for customer 60 on 21/01/2019 already exists
Order information for customer 61 on 21/01/2019 already exists
Order information for customer 62 on 21/01/2019 already exists
Order information for customer 63 on 22/01/2019 already exists
Order information for customer 64 on 22/01/2019 already exists
Order information for customer 65 on 23/01/2019 already exists
Order information for customer 66 on 25/01/2019 already exists
Order information for customer 67 on 25/01/2019 already exists
Order information for customer 68 on 25/01/2019 already exists
Order information for customer 69 on 25/01/2019 already exists
Order information for customer 70 on 26/01/2019 already exists
Order information for customer 71 on 27/01/2019 already exists
```

```
Order information for customer 72 on 27/01/2019 already exists
Order information for customer 73 on 27/01/2019 already exists
Order information for customer 74 on 28/01/2019 already exists
Order information for customer 75 on 28/01/2019 already exists
Order information for customer 76 on 28/01/2019 already exists
Order information for customer 77 on 29/01/2019 already exists
Order information for customer 78 on 29/01/2019 already exists
Order information for customer 79 on 29/01/2019 already exists
Order information for customer 80 on 30/01/2019 already exists
Order information for customer 81 on 30/01/2019 already exists
Order information for customer 82 on 30/01/2019 already exists
Order information for customer 83 on 30/01/2019 already exists
Order information for customer 84 on 31/01/2019 already exists
Order information for customer 85 on 31/01/2019 already exists
```

```
---------------------------------------------------------------------------
OperationalError                          Traceback (most recent call last)
<ipython-input-8-29240ef9f092> in <module>
     29     sql = "INSERT INTO tOrderDetail() VALUES(?,?,?)"
     30     for row in data_with_cust_order.values:
---> 31         curs.execute(sql, (row[COL_ORDER_ID], row[COL_PROD_ID],␣
 ↪row[COL_QTY]))

OperationalError: near ")": syntax error
```

[9]:
```
pd.read_sql("SELECT * FROM tCust;", conn)
```

[9]:

	cust_id	first_name	last_name	address	zip
0	1	Bib Fortuna	Walker	6829 2nd Street	10177
1	2	Unkar Plutt	Jennings	5295 4th Street South	35130
2	3	Dodonna	Garza	3639 Briarwood Court	79783
3	4	Rabe	Woodward	2517 Lake Avenue	18505
4	5	Plo Koon	Ferguson	3332 Prospect Street	14433
..	…	…	…	…	…
80	81	Sun Rit	Walker	3961 Beechwood Drive	85354
81	82	Jabba	Jennings	2985 Washington Street	43210
82	83	Jira	Adams	7763 Main Street	46538
83	84	Bala-Tik	Elliott	9088 Valley Road	55003
84	85	Rieekan	Chen	2668 College Street	39631

[85 rows x 5 columns]

[10]:
```
conn.close()
```

---

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.

---

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).

[ ]:

---

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.

[ ]:

---

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

[ ]:

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