SQL ASSIGNMENT

Name: Divya Muraleedharan

Student ID: 22030331

Online Shopping Cart (Electronics)

This report presents an online shopping cart designed specifically for electronic products, containing four tables: Customer, Product, Purchase, OrderDetail. I utilized the Faker library to generate fictitious names.

My database comprises Customer table, which contains a unique CustomerID for each individual. Nominal data includes Name and PhoneNumber. Date of Birth (DOB) is categorized as an interval data. ContactPreference and Address are both classified as Nominal data and represent preferred method of reaching customers and reaching unique location identifier. Finally, CustomerRating is an Ordinal variable that ranks customers by their ranking levels: Low, Medium, High.

```
!pip install Faker
import numpy as np
import pandas as pd
from faker import Faker
# Number of samples
n    = 1000
# Use Faker library to generate random names
Fake = Faker()
```

```
Requirement already satisfied: Faker in /usr/local/lib/python3.10/dist-packages (20.0.0)
Requirement already satisfied: python-dateutil>=2.4 in /usr/local/lib/python3.10/dist-packages (from Faker) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.4->Faker) (1.16.0)
```

```
# Nominal data: Customer ID
customer_id = np.random.choice(np.arange(1000, 10000), size=n,replace=False)
# Nominal data : Name
full_names  = [Fake.name() for _ in range(n)]
# Ratio data: Birth dates
birth_year  = np.random.randint(1970, 2000, n)
birth_month = np.random.randint(1, 13, n)
birth_day   = np.random.randint(1, 29, n)
birth_date  = [f'{birth_year[i]}-{str(birth_month[i]).zfill(2)}-{str(birth_day[i]).zfill(2)}' for i in range(n)]
# Nominal data : Phone Number
country_codes = np.array([f"+{Fake.random_int(1, 999)}" for _ in range(1000)])
phone_number  = np.array([f"{code} {Fake.random_number(10)}" for code in country_codes])
# Nominal data: Contact Preferences
contact_preference = np.random.choice(['Phone','SMS'], size=1000)
# Nominal data: Postcode
characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
postcodes = [f'{np.random.choice(list(characters))}\
{np.random.choice(list(characters))}{np.random.randint(1, 100)}\
{np.random.choice(list(characters))}{np.random.choice(list(characters))}' for _ in range(n)]
# Randomly select 50 indices to set to NaN
n_points = 50
random_indices = np.random.choice(n, n_points, replace=False)
postcodes = np.array(postcodes)  # Convert postcodes to a NumPy array
postcodes[random_indices] = 'NaN'
# Ordinal data: Customer Rating
customer_rating = np.random.choice(['Low', 'Medium', 'High'], n , p=[0.2, 0.4, 0.4])
```

This code creates a customer table using Python's Pandas library. Faker library is used to generate realistic names and random values are generated for birthdays, phone numbers, contact information and address. In addition, 50 zipcodes are randomly set to NaN to simulate missing data, Finally, a dataframe named customer_df is created to organize this data and the resulting table is displayed below.

```
# Create the DataFrame for the Customer table
customer_df = pd.DataFrame({
    'CustomerID': customer_id,
    'Name': full_names,
    'DOB' : birth_date,
    'PhoneNumber': phone_number,
    'ContactPreference' : contact_preference,
    'Address' : postcodes,
    'Customer Rating' : customer_rating
    })
# Display the customer_df DataFrame
customer_df
```

The output of the dataframe, customer_df is given below:

	CustomerID	Name	DOB	PhoneNumber	ContactPreference	Address	Customer Rating
0	5978	Sarah Howard	1983-02-05	+209 4224555638	Phone	SO58CT	Low
1	2918	Dennis Hernandez	1979-06-23	+803 4479216011	SMS	CM18JS	High
2	5537	Brent Hall	1996-07-12	+904 3003203869	SMS	ZN14PQ	Medium
3	7431	Kristen Wagner	1979-07-20	+796 6435145407	SMS	BA49MJ	Low
4	4198	Christian Hayes	1988-01-04	+413 9929761857	SMS	JX69PT	High
...	...	...	...	...	...	...	...
995	9975	Jessica Bird	1978-05-28	+721 2493634913	Phone	XM26TX	Low
996	9062	Jonathan Bradford	1992-09-16	+415 9718416383	Phone	SZ51MZ	High
997	1230	Brandon Kennedy	1980-10-18	+695 2330399903	Phone	JR40FX	High
998	8487	Linda Acosta	1983-11-25	+414 1287665850	SMS	XT51UB	Medium
999	1134	Nathan Porter	1978-10-13	+585 3863000166	SMS	YG46US	High

1000 rows × 7 columns

The code below creates a Product table in which each product is assigned a unique ID, creating a nominal data type to uniquely identify items. The Product names are randomly chosen from a list of electronic items. Prices are generated based on predefined ranges for specific electronic products, embracing the ratio data type as it quantitatively measures the cost of each product. The resulting table, named product_df is created to organize this data and the resulting table is displayed below.

```python
# Creating Products Table
# List of electronic item names
electronic_items = ['Smartphone', 'Laptop', 'Washing Machine', 'Smart TV', 'Air Conditioner', 'Heater',\
                    'Microwave Oven', 'Air Fryer', 'Smartwatch', 'Hair Dryer']

# Nominal data: Product ID
product_id = np.random.choice(np.arange(100, 10000), size=n,replace=False)

# Nominal data: Product Names
product_names = np.random.choice(electronic_items, size=1000)

# Ratio data: Price
price_ranges = {
    'Smartphone': (300, 1200),
    'Laptop': (500, 2000),
    'Washing Machine': (300, 1200),
    'Smart TV': (600, 3000),
    'Air Conditioner': (400, 2500),
    'Heater': (80, 400),
    'Microwave Oven': (100, 800),
    'Air Fryer': (80, 300),
    'Smartwatch': (100, 500),
    'Hair Dryer': (20, 150),
}
prices = [int(np.round(np.random.uniform(low, high))) for name in product_names \
          for product, (low, high) in price_ranges.items() if name == product]
prices = np.round(np.array(prices),2)
```

```python
# Create the DataFrame for the Product table
product_df = pd.DataFrame({
    'ProductID': product_id,
    'ProductName': product_names,
    'Price': prices,
    })
# Display the product_df DataFrame
product_df
```

	ProductID	ProductName	Price
0	2921	Microwave Oven	256
1	8100	Smartwatch	398
2	5454	Microwave Oven	509
3	8205	Smart TV	1958
4	2922	Smart TV	1470
...	...	...	...
995	6749	Smart TV	945
996	3072	Air Conditioner	1503
997	8731	Air Conditioner	2240
998	6507	Microwave Oven	433
999	9894	Heater	133

1000 rows × 3 columns

The code below creates Purchase table which contains OrderID, unique identifier that distinguishes them. Purchase dates are randomly generated and TotalAmount shows the cost of each order. All the information is neatly organized in the order_df table, which provides an overview of our purchase history. Here a compound key, TransactionKey is created by combining the order ID and customer ID, which gives each order its own unique identity.

```
# Creating Order Table
# Nominal data: Order ID
order_id = np.random.choice(np.arange(1000, 10000), size=n,replace=False)

# Interval data: Purchase date
order_dates = np.array([Fake.date_this_decade() for _ in range(1000)])

# Ratio data: Total Amount
total_amounts = np.random.uniform(50, 500, size=1000)
total_amounts = np.round(total_amounts, 2)

# Create the DataFrame for the Order table
order_df = pd.DataFrame({
    'OrderID': order_id,
    'PurchaseDate': order_dates,
    'CustomerID': customer_id,
    'TotalAmount' : total_amounts
    })
# Setting compound key (combination of OrderID and CustomerID)
order_df['TransactionKey'] = order_df['OrderID'].astype(str) + '_' + order_df['CustomerID'].astype(str)
# Display the order_df DataFrame
order_df
```

	OrderID	PurchaseDate	CustomerID	TotalAmount	TransactionKey
0	1479	2022-12-09	5978	359.80	1479_5978
1	2475	2022-07-29	2918	264.09	2475_2918
2	6833	2020-02-19	5537	206.69	6833_5537
3	7424	2023-09-24	7431	331.97	7424_7431
4	1847	2020-01-12	4198	380.13	1847_4198
...	...	...	...	...	...
995	2991	2022-06-19	9975	143.00	2991_9975
996	9794	2020-06-22	9062	258.90	9794_9062
997	8766	2023-05-09	1230	207.38	8766_1230
998	6303	2022-12-04	8487	382.03	6303_8487
999	4593	2023-05-28	1134	355.52	4593_1134

1000 rows × 5 columns

```
[73] # Creating Sales Table
     # Nominal data: Sales ID
     sales_id = np.random.choice(np.arange(10000, 100000), size=n,replace=False)

     # Ratio data: Quantity
     quantity_sold = np.random.randint(1, 11, size=1000)

     # Ratio data: Subtotal
     subtotal = np.round(np.random.uniform(10, 100, size=1000),2)

     # Ordinal data: Shipping Method
     shipping_method = np.random.choice(['Standard', 'Express', 'Next-Day'], size=1000)

     # Create the DataFrame for the Sales table
     sales_df = pd.DataFrame({
         'SaleID': sales_id,
         'ProductID' : product_id,
         'Quantity': quantity_sold,
         'Subtotal': subtotal,
         'ShippingMethod': shipping_method
         })
     # Display the sales_df DataFrame
     sales_df
```

The code above creates OrderDetail table which contains unique SaleID. Quantity sold shows how many items were sold in each sample sale and the subtotal shows how much money each sale made. Shipping method means how the goods are delivered, whether regular, fast, or super-fast. All these are packed into sales_df table and the output is shown below:

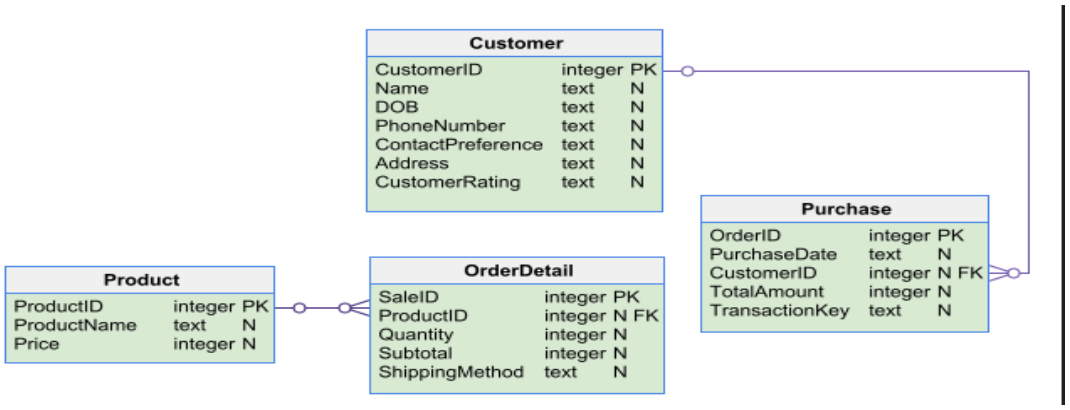
	SaleID	ProductID	Quantity	Subtotal	ShippingMethod
0	55461	128	1	31.13	Standard
1	97205	5036	4	11.83	Next-Day
2	58647	7143	8	37.00	Next-Day
3	70629	6180	10	99.24	Standard
4	30252	3854	8	66.60	Express
...	...	...	...	...	...
995	12145	6224	3	37.17	Standard
996	46184	8691	4	25.52	Express
997	41312	731	2	76.44	Next-Day
998	13868	9774	3	69.57	Next-Day
999	98378	9997	8	67.79	Express

1000 rows × 5 columns

The code below organizes the table and save them as CSV files. The customer_df, product_df, order_df and sales_df tables are each set up with a special index for easy reference. Each table is saved as a CSV file – Customer.csv, Product.csv, Purchase,csv and OrderDetails.csv

```
# Set the index and save to CSV
customer_df.set_index('CustomerID').to_csv('Customer.csv')
product_df.set_index('ProductID').to_csv('Product.csv')
order_df.set_index('OrderID').to_csv('Purchase.csv')
sales_df.set_index('SaleID').to_csv('OrderDetails.csv')
```

Then I imported the produced CSV file into SQLite to construct a database. The tables are modified by setting the primary key and foreign key.



My database schema is illustrated above, and it has four tables: Customer, Product, Purchase and OrderDetail. The Customer table stores customer information, the Product table stores product catalogue information, the Purchase table stores transactions, and The OrderDetail table stores sales information. These tables create a relational structure for effectively managing customer interactions, product data, and transaction records by connecting with each other via foreign keys.

The query below retrieves information about the top five spending customers and their total purchase amount.

```
1  SELECT c.CustomerID,
2         c.Name,
3         SUM(p.TotalAmount) AS TotalPurchaseAmount
4  FROM "Customer" c
5  JOIN "Purchase" p ON c.CustomerID = p.CustomerID
6  GROUP BY c.CustomerID, c.Name
7  ORDER BY TotalPurchaseAmount DESC
8  LIMIT 5;
9
```

	CustomerID	Name	TotalPurchaseAmount
1	7557	Victor Whitaker	499.17
2	9201	Gregory Camacho	498.97
3	8878	Sydney Sims	498.8
4	6318	Judy Johnson	498.49
5	1734	Hannah Luna	498.43

The query below retrieves six records from the "Purchase" table where the "PurchaseDate" falls between '2022-03-21' and '2022-06-19'.

SQL 1

```
SELECT * FROM "Purchase" WHERE PurchaseDate BETWEEN '2022-03-21' AND '2022-06-19' LIMIT 6;
```

OrderID	PurchaseDate	CustomerID	TotalAmount	TransactionKey
1028	2022-06-02	7799	359.47	1028_7799
1055	2022-03-21	5302	350.21	1055_5302
1056	2022-06-19	7089	223.86	1056_7089
1253	2022-06-15	6741	316.14	1253_6741
1272	2022-05-08	4997	181.28	1272_4997
1339	2022-05-28	7608	175.47	1339_7608

The query below searches for the five best-selling products, summarizes their sales quantities, and displays the ProductID,ProductName, and TotalQuantitySold.

SQL 1

```
SELECT
    Product.ProductID,
    Product.ProductName,
    SUM(OrderDetail.Quantity) AS TotalQuantitySold
FROM
    Product
JOIN
    OrderDetail ON Product.ProductID = OrderDetail.ProductID
GROUP BY
    Product.ProductID, Product.ProductName
ORDER BY
    TotalQuantitySold DESC
    LIMIT 5
```

	ProductID	ProductName	TotalQuantitySold
1	2097	Smartphone	10
2	2613	Smart TV	10
3	4951	Smartwatch	10
4	6042	Microwave Oven	10
5	6618	Heater	10

Th query below retrieves the last purchase data for the transaction key 1015_9024 from my database, sorting the results in descending order by purchase date and limiting to only one entry.

```
SELECT *
FROM Purchase
WHERE TransactionKey = "1015_9024";
```

OrderID	PurchaseDate	CustomerID	TotalAmount	TransactionKey