In [ ]:
```
import pandas as pd
import random
from datetime import datetime, timedelta
import sqlite3
```

In [ ]:
```
# Connect to SQLite database (or create it if it doesn't exist)
conn = sqlite3.connect('ecommerce.db')
cursor = conn.cursor()

# Execute the SQL script to create tables
sql_script = """
CREATE TABLE IF NOT EXISTS Customer (
    CustomerID INTEGER PRIMARY KEY,
    Name TEXT NOT NULL,
    ContactNumber TEXT NOT NULL,
    Email TEXT NOT NULL,
    ShippingAddress TEXT NOT NULL,
    DateCreated DATE NOT NULL,
    LastUpdated DATE NOT NULL
);

CREATE TABLE IF NOT EXISTS Product (
    ProductID INTEGER PRIMARY KEY,
    ProductName TEXT NOT NULL,
    Description TEXT,
    Category TEXT,
    DateCreated DATE NOT NULL,
    LastUpdated DATE NOT NULL
);

CREATE TABLE IF NOT EXISTS ProductVariant (
    VariantID INTEGER PRIMARY KEY,
    ProductID INTEGER NOT NULL,
    VariantName TEXT NOT NULL,
    Attributes TEXT,
    DateCreated DATE NOT NULL,
    LastUpdated DATE NOT NULL,
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
```

```
CREATE TABLE IF NOT EXISTS ProductPriceHistory (
    PriceID INTEGER PRIMARY KEY,
    VariantID INTEGER NOT NULL,
    Price REAL NOT NULL,
    StartDate DATE NOT NULL,
    EndDate DATE NOT NULL,
    FOREIGN KEY (VariantID) REFERENCES ProductVariant(VariantID)
);

CREATE TABLE IF NOT EXISTS `Order` (
    OrderID INTEGER PRIMARY KEY,
    CustomerID INTEGER NOT NULL,
    OrderDate DATE NOT NULL,
    TotalAmount REAL NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);

CREATE TABLE IF NOT EXISTS OrderItem (
    OrderItemID INTEGER PRIMARY KEY,
    OrderID INTEGER NOT NULL,
    VariantID INTEGER NOT NULL,
    Quantity INTEGER NOT NULL,
    Price REAL NOT NULL,
    Discount REAL NOT NULL,
    TotalPrice REAL NOT NULL,
    FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID),
    FOREIGN KEY (VariantID) REFERENCES ProductVariant(VariantID)
);
"""

# Execute the script
cursor.executescript(sql_script)
conn.commit()

print("Database and tables created successfully.")
```

```
Database and tables created successfully.
```

Step-by-Step Guide to Setting Up SQLite Database for E-commerce

1. Importing Required Libraries

```
import sqlite3
```

2. Connecting to SQLite Database

```
# Connect to SQLite database (or create it if it doesn't exist)
conn = sqlite3.connect('ecommerce.db')
cursor = conn.cursor()
```

Explanation:

- We import the `sqlite3` module to interact with SQLite databases.
- We establish a connection to the SQLite database named 'ecommerce.db' using `sqlite3.connect()`.
- A cursor object (`cursor`) is created to execute SQL commands within the database.

3. Creating Tables

```
# Execute the SQL script to create tables
sql_script = """
CREATE TABLE IF NOT EXISTS Customer (
    CustomerID INTEGER PRIMARY KEY,
    Name TEXT NOT NULL,
    ContactNumber TEXT NOT NULL,
    Email TEXT NOT NULL,
    ShippingAddress TEXT NOT NULL,
    DateCreated DATE NOT NULL,
    LastUpdated DATE NOT NULL
);

CREATE TABLE IF NOT EXISTS Product (
    ProductID INTEGER PRIMARY KEY,
    ProductName TEXT NOT NULL,
    Description TEXT,
    Category TEXT,
    DateCreated DATE NOT NULL,
    LastUpdated DATE NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS ProductVariant (
    VariantID INTEGER PRIMARY KEY,
    ProductID INTEGER NOT NULL,
    VariantName TEXT NOT NULL,
    Attributes TEXT,
    DateCreated DATE NOT NULL,
    LastUpdated DATE NOT NULL,
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);

CREATE TABLE IF NOT EXISTS ProductPriceHistory (
    PriceID INTEGER PRIMARY KEY,
    VariantID INTEGER NOT NULL,
    Price REAL NOT NULL,
    StartDate DATE NOT NULL,
    EndDate DATE NOT NULL,
    FOREIGN KEY (VariantID) REFERENCES ProductVariant(VariantID)
);

CREATE TABLE IF NOT EXISTS `Order` (
    OrderID INTEGER PRIMARY KEY,
    CustomerID INTEGER NOT NULL,
    OrderDate DATE NOT NULL,
    TotalAmount REAL NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);

CREATE TABLE IF NOT EXISTS OrderItem (
    OrderItemID INTEGER PRIMARY KEY,
    OrderID INTEGER NOT NULL,
    VariantID INTEGER NOT NULL,
    Quantity INTEGER NOT NULL,
    Price REAL NOT NULL,
    Discount REAL NOT NULL,
    TotalPrice REAL NOT NULL,
    FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID),
    FOREIGN KEY (VariantID) REFERENCES ProductVariant(VariantID)
);
```

```
"""

# Execute the script
cursor.executescript(sql_script)
conn.commit()

print("Database and tables created successfully.")
```

Explanation:

- The `sql_script` variable contains a multi-line SQL script that defines the structure of several tables (`Customer` , `Product` , `ProductVariant` , `ProductPriceHistory` , `Order` , `OrderItem`) for an e-commerce database.
- Each `CREATE TABLE IF NOT EXISTS` statement ensures that the table is created only if it does not already exist, with specified columns and constraints (e.g., primary keys, foreign keys).
- `cursor.executescript(sql_script)` executes the entire SQL script defined in `sql_script` using the cursor object (`cursor`).
- `conn.commit()` commits the changes made (i.e., table creation) to the SQLite database.
- Finally, a confirmation message "Database and tables created successfully." is printed.

Conclusion

This code snippet sets up an SQLite database (`ecommerce.db`) and creates necessary tables for an e-commerce application. Each table's schema is defined with appropriate columns and constraints to store customer information, products, orders, and order items.

In [ ]:
```
import pandas as pd
import random
from datetime import datetime, timedelta

def random_date(start, end):
    return start + timedelta(days=random.randint(0, (end - start).days))

def generate_customers(n):
    customers = []
    for i in range(n):
        customer = {
            'CustomerID': i + 1,
            'Name': f'Customer {i + 1}',
            'ContactNumber': f'{random.randint(1000000000, 9999999999)}',
            'Email': f'customer{i + 1}@example.com',
```

```
            'ShippingAddress': f'Address {i + 1}',
            'DateCreated': random_date(datetime(2020, 1, 1), datetime(2022, 1, 1)),
            'LastUpdated': datetime.now()
        }
        customers.append(customer)
    return pd.DataFrame(customers)
def generate_products(n):
    products = []
    for i in range(n):
        product = {
            'ProductID': i + 1,
            'ProductName': f'Product {i + 1}',
            'Description': f'Description for Product {i + 1}',
            'Category': random.choice(['Electronics', 'Books', 'Clothing']),
            'DateCreated': random_date(datetime(2020, 1, 1), datetime(2022, 1, 1)),
            'LastUpdated': datetime.now()
        }
        products.append(product)
    return pd.DataFrame(products)

def generate_variants(products, n_variants=2):
    variants = []
    for product_id in products['ProductID']:
        for j in range(n_variants):
            variant = {
                'VariantID': len(variants) + 1,
                'ProductID': product_id,
                'VariantName': f'Variant {j + 1} for Product {product_id}',
                'Attributes': f'Color: {random.choice(["Red", "Blue", "Green"])}; Size: {random.choice(["S", "M", "L"])}',
                'DateCreated': random_date(datetime(2020, 1, 1), datetime(2022, 1, 1)),
                'LastUpdated': datetime.now()
            }
            variants.append(variant)
    return pd.DataFrame(variants)

def generate_price_history(variants):
    price_history = []
    for variant_id in variants['VariantID']:
        start_date = random_date(datetime(2020, 1, 1), datetime(2022, 1, 1))
        for _ in range(3):  # Three price changes per variant
            end_date = start_date + timedelta(days=random.randint(30, 365))
```

```
            price = {
                'PriceID': len(price_history) + 1,
                'VariantID': variant_id,
                'Price': round(random.uniform(10, 1000), 2),
                'StartDate': start_date,
                'EndDate': end_date
            }
            price_history.append(price)
            start_date = end_date + timedelta(days=1)
    return pd.DataFrame(price_history)

def generate_orders(customers, variants):
    orders = []
    order_items = []
    for customer_id in customers['CustomerID']:
        order_date = random_date(datetime(2022, 1, 1), datetime(2024, 1, 1))
        order = {
            'OrderID': len(orders) + 1,
            'CustomerID': customer_id,
            'OrderDate': order_date,
            'TotalAmount': 0  # Will be calculated later
        }
        orders.append(order)
        total_amount = 0
        for _ in range(random.randint(1, 5)):  # 1 to 5 items per order
            variant = random.choice(variants['VariantID'])
            price = random.uniform(10, 1000)
            quantity = random.randint(1, 5)
            discount = random.uniform(0, 0.3) * price
            total_price = (price - discount) * quantity
            order_item = {
                'OrderItemID': len(order_items) + 1,
                'OrderID': order['OrderID'],
                'VariantID': variant,
                'Quantity': quantity,
                'Price': price,
                'Discount': discount,
                'TotalPrice': total_price
            }
            order_items.append(order_item)
            total_amount += total_price
```

```
        order['TotalAmount'] = total_amount
    return pd.DataFrame(orders), pd.DataFrame(order_items)

# Generate sample data
customers_df = generate_customers(100)
products_df = generate_products(50)
variants_df = generate_variants(products_df, 2)
price_history_df = generate_price_history(variants_df)
orders_df, order_items_df = generate_orders(customers_df, variants_df)

# Insert the data into the SQLite database
conn = sqlite3.connect('ecommerce.db')
cursor = conn.cursor()

# Insert data
customers_df.to_sql('Customer', conn, if_exists='append', index=False)
products_df.to_sql('Product', conn, if_exists='append', index=False)
variants_df.to_sql('ProductVariant', conn, if_exists='append', index=False)
price_history_df.to_sql('ProductPriceHistory', conn, if_exists='append', index=False)
orders_df.to_sql('Order', conn, if_exists='append', index=False)
order_items_df.to_sql('OrderItem', conn, if_exists='append', index=False)

print("Data inserted successfully.")
```

```
Data inserted successfully.
```

To convert the data generation and insertion process into a structured Jupyter Notebook format with explanations, follow these steps:

Generating and Inserting Sample Data into SQLite Database for E-commerce

1. Importing Required Libraries

```
import pandas as pd
import random
from datetime import datetime, timedelta
```

2. Defining Utility Functions

Explanation:

- `random_date(start, end)` : Generates a random date between `start` and `end`.
- `generate_customers(n)` : Generates `n` customer records with randomized data.
- `generate_products(n)` : Generates `n` product records with randomized data.
- `generate_variants(products, n_variants=2)` : Generates product variants (defaulting to 2 variants per product) with randomized data.
- `generate_price_history(variants)` : Generates price history records for product variants with randomized data.
- `generate_orders(customers, variants)` : Generates order records and associated order items with randomized data, linking to customers and variants.

3. Generating Sample Data

Explanation:

- Calls the defined functions to generate sample dataframes (`customers_df` , `products_df` , `variants_df` , `price_history_df` , `orders_df` , `order_items_df`) filled with randomized records.

4. Connecting to SQLite Database and Inserting Data

Explanation:

- Establishes a connection to the SQLite database (`ecommerce.db`) and creates a cursor (`cursor`) to execute SQL commands.
- Uses `to_sql()` method from Pandas to insert each dataframe (`customers_df` , `products_df` , `variants_df` , `price_history_df` , `orders_df` , `order_items_df`) into their respective tables (`Customer` , `Product` , `ProductVariant` , `ProductPriceHistory` , `Order` , `OrderItem`) within the database.
- Prints "Data inserted successfully." upon successful completion of data insertion.

In [ ]:
```
# Connect to the SQLite database
conn = sqlite3.connect('ecommerce.db')
cursor = conn.cursor()

# Define the number of customers and orders per customer for both years
num_customers = 10
orders_per_customer = 5

# Get the current year and the previous year
```

```
current_year = datetime.now().year
previous_year = current_year - 1

# Insert orders for the current year for each customer
for customer_id in range(1, num_customers + 1):
    for _ in range(orders_per_customer):
        order_date = f"{current_year}-06-15"  # Example order date
        total_amount = round(random.uniform(100, 1000), 2)  # Random total amount
        cursor.execute("INSERT INTO `Order` (CustomerID, OrderDate, TotalAmount) VALUES (?, ?, ?)",
                       (customer_id, order_date, total_amount))

# Insert orders for the previous year for each customer
for customer_id in range(1, num_customers + 1):
    for _ in range(orders_per_customer):
        order_date = f"{previous_year}-07-10"  # Example order date
        total_amount = round(random.uniform(100, 1000), 2)  # Random total amount
        cursor.execute("INSERT INTO `Order` (CustomerID, OrderDate, TotalAmount) VALUES (?, ?, ?)",
                       (customer_id, order_date, total_amount))

# Commit the changes to the database
conn.commit()

# Close the database connection
conn.close()

print("Sample data with random order amounts inserted successfully.")
```

```
Sample data with random order amounts inserted successfully.
```

Explanation of Code for Inserting Sample Order Data into SQLite Database

1. Connecting to the SQLite Database

- Establishes a connection to the SQLite database named `ecommerce.db` using `sqlite3.connect('ecommerce.db')`.
- Creates a cursor (`cursor`) to execute SQL commands within the database.

2. Defining Parameters

- **`num_customers`**: Specifies the number of customers.

- `orders_per_customer`: Determines the number of orders per customer for both the current and previous years.

3. Getting Current and Previous Years

- `current_year`: Retrieves the current year using `datetime.now().year`.
- `previous_year`: Calculates the previous year as `current_year - 1`.

4. Inserting Orders for the Current Year

- Loops through each customer (`customer_id`) from 1 to `num_customers`.
- Inserts `orders_per_customer` orders for the current year (`current_year`) using a specified example order date (`order_date = f"{current_year}-06-15"`) and a randomly generated total amount (`total_amount`).

5. Inserting Orders for the Previous Year

- Similar to the current year loop, but inserts orders for the previous year (`previous_year`) using a different example order date (`order_date = f"{previous_year}-07-10"`) and randomly generated total amount (`total_amount`).

6. Committing Changes

- Calls `conn.commit()` to save all changes (i.e., insertion of orders) made to the SQLite database.

7. Closing the Database Connection

- Closes the connection to the SQLite database using `conn.close()` to release resources.

8. Print Confirmation Message

- Outputs "Sample data with random order amounts inserted successfully." to indicate the successful execution of the script.

In [ ]:
```
# Connect to the SQLite database
conn = sqlite3.connect('ecommerce.db')
cursor = conn.cursor()

# Function to fetch and display data from a table
def fetch_data(table_name):
    query = f"SELECT * FROM \"{table_name}\""  # Wrap the table name in double quotes
```

```
    df = pd.read_sql_query(query, conn)
    print(f"\nData from {table_name}:\n")
    print(df)

# List of tables to fetch data from
tables = ['Customer', 'Product', 'ProductVariant', 'ProductPriceHistory', 'Order', 'OrderItem']

# Fetch and display data from each table
for table in tables:
    fetch_data(table)

# Close the database connection
conn.close()
```

```
Data from Customer:

    CustomerID          Name ContactNumber                  Email  \
0            1    Customer 1    1192836915    customer1@example.com
1            2    Customer 2    8149930154    customer2@example.com
2            3    Customer 3    7542983580    customer3@example.com
3            4    Customer 4    4135774297    customer4@example.com
4            5    Customer 5    5000227778    customer5@example.com
..         ...           ...           ...                      ...
95          96   Customer 96    7702687547   customer96@example.com
96          97   Customer 97    3272797464   customer97@example.com
97          98   Customer 98    6565777886   customer98@example.com
98          99   Customer 99    5994649111   customer99@example.com
99         100  Customer 100    5111905931  customer100@example.com

   ShippingAddress          DateCreated                 LastUpdated
0        Address 1  2020-06-22 00:00:00  2024-06-12 08:07:55.802743
1        Address 2  2020-02-19 00:00:00  2024-06-12 08:07:55.802743
2        Address 3  2021-06-14 00:00:00  2024-06-12 08:07:55.802743
3        Address 4  2020-01-23 00:00:00  2024-06-12 08:07:55.802743
4        Address 5  2021-10-30 00:00:00  2024-06-12 08:07:55.802743
..             ...                  ...                         ...
95      Address 96  2020-02-25 00:00:00  2024-06-12 08:07:55.802743
96      Address 97  2021-11-29 00:00:00  2024-06-12 08:07:55.802743
97      Address 98  2020-04-04 00:00:00  2024-06-12 08:07:55.802743
98      Address 99  2021-12-16 00:00:00  2024-06-12 08:07:55.802743
99     Address 100  2020-05-27 00:00:00  2024-06-12 08:07:55.802743

[100 rows x 7 columns]

Data from Product:

    ProductID ProductName                Description     Category  \
0           1   Product 1   Description for Product 1        Books
1           2   Product 2   Description for Product 2  Electronics
2           3   Product 3   Description for Product 3     Clothing
3           4   Product 4   Description for Product 4        Books
4           5   Product 5   Description for Product 5     Clothing
5           6   Product 6   Description for Product 6     Clothing
6           7   Product 7   Description for Product 7     Clothing
7           8   Product 8   Description for Product 8  Electronics
```

8	9	Product 9	Description for Product 9	Clothing
9	10	Product 10	Description for Product 10	Electronics
10	11	Product 11	Description for Product 11	Clothing
11	12	Product 12	Description for Product 12	Clothing
12	13	Product 13	Description for Product 13	Books
13	14	Product 14	Description for Product 14	Books
14	15	Product 15	Description for Product 15	Clothing
15	16	Product 16	Description for Product 16	Electronics
16	17	Product 17	Description for Product 17	Books
17	18	Product 18	Description for Product 18	Clothing
18	19	Product 19	Description for Product 19	Clothing
19	20	Product 20	Description for Product 20	Clothing
20	21	Product 21	Description for Product 21	Clothing
21	22	Product 22	Description for Product 22	Books
22	23	Product 23	Description for Product 23	Electronics
23	24	Product 24	Description for Product 24	Books
24	25	Product 25	Description for Product 25	Clothing
25	26	Product 26	Description for Product 26	Books
26	27	Product 27	Description for Product 27	Books
27	28	Product 28	Description for Product 28	Books
28	29	Product 29	Description for Product 29	Books
29	30	Product 30	Description for Product 30	Clothing
30	31	Product 31	Description for Product 31	Books
31	32	Product 32	Description for Product 32	Clothing
32	33	Product 33	Description for Product 33	Electronics
33	34	Product 34	Description for Product 34	Books
34	35	Product 35	Description for Product 35	Clothing
35	36	Product 36	Description for Product 36	Books
36	37	Product 37	Description for Product 37	Clothing
37	38	Product 38	Description for Product 38	Electronics
38	39	Product 39	Description for Product 39	Clothing
39	40	Product 40	Description for Product 40	Electronics
40	41	Product 41	Description for Product 41	Clothing
41	42	Product 42	Description for Product 42	Clothing
42	43	Product 43	Description for Product 43	Electronics
43	44	Product 44	Description for Product 44	Electronics
44	45	Product 45	Description for Product 45	Clothing
45	46	Product 46	Description for Product 46	Books
46	47	Product 47	Description for Product 47	Electronics
47	48	Product 48	Description for Product 48	Clothing
48	49	Product 49	Description for Product 49	Clothing

```
49         50  Product 50  Description for Product 50        Books

            DateCreated                 LastUpdated
0   2021-07-16 00:00:00  2024-06-12 08:07:55.809744
1   2021-05-10 00:00:00  2024-06-12 08:07:55.809744
2   2021-09-14 00:00:00  2024-06-12 08:07:55.809744
3   2020-10-11 00:00:00  2024-06-12 08:07:55.809744
4   2020-11-04 00:00:00  2024-06-12 08:07:55.809744
5   2021-08-29 00:00:00  2024-06-12 08:07:55.809744
6   2021-04-19 00:00:00  2024-06-12 08:07:55.809744
7   2021-04-23 00:00:00  2024-06-12 08:07:55.809744
8   2021-05-01 00:00:00  2024-06-12 08:07:55.809744
9   2021-11-07 00:00:00  2024-06-12 08:07:55.809744
10  2021-12-24 00:00:00  2024-06-12 08:07:55.809744
11  2020-02-09 00:00:00  2024-06-12 08:07:55.809744
12  2020-07-12 00:00:00  2024-06-12 08:07:55.809744
13  2020-04-29 00:00:00  2024-06-12 08:07:55.809744
14  2020-11-20 00:00:00  2024-06-12 08:07:55.809744
15  2021-12-05 00:00:00  2024-06-12 08:07:55.809744
16  2020-04-25 00:00:00  2024-06-12 08:07:55.809744
17  2020-09-25 00:00:00  2024-06-12 08:07:55.809744
18  2020-05-24 00:00:00  2024-06-12 08:07:55.809744
19  2020-06-29 00:00:00  2024-06-12 08:07:55.809744
20  2021-03-29 00:00:00  2024-06-12 08:07:55.809744
21  2021-07-08 00:00:00  2024-06-12 08:07:55.809744
22  2021-11-07 00:00:00  2024-06-12 08:07:55.809744
23  2020-04-12 00:00:00  2024-06-12 08:07:55.809744
24  2021-02-27 00:00:00  2024-06-12 08:07:55.809744
25  2021-09-16 00:00:00  2024-06-12 08:07:55.809744
26  2020-09-02 00:00:00  2024-06-12 08:07:55.809744
27  2021-07-04 00:00:00  2024-06-12 08:07:55.809744
28  2020-03-15 00:00:00  2024-06-12 08:07:55.809744
29  2021-03-23 00:00:00  2024-06-12 08:07:55.809744
30  2021-07-25 00:00:00  2024-06-12 08:07:55.809744
31  2020-07-21 00:00:00  2024-06-12 08:07:55.809744
32  2021-06-17 00:00:00  2024-06-12 08:07:55.809744
33  2021-10-11 00:00:00  2024-06-12 08:07:55.809744
34  2021-01-08 00:00:00  2024-06-12 08:07:55.809744
35  2020-09-27 00:00:00  2024-06-12 08:07:55.809744
36  2020-05-04 00:00:00  2024-06-12 08:07:55.809744
37  2021-12-11 00:00:00  2024-06-12 08:07:55.809744
```

```
38  2020-09-29 00:00:00  2024-06-12 08:07:55.809744
39  2020-05-31 00:00:00  2024-06-12 08:07:55.809744
40  2020-08-11 00:00:00  2024-06-12 08:07:55.809744
41  2020-07-25 00:00:00  2024-06-12 08:07:55.809744
42  2021-12-18 00:00:00  2024-06-12 08:07:55.809744
43  2021-12-04 00:00:00  2024-06-12 08:07:55.809744
44  2020-04-03 00:00:00  2024-06-12 08:07:55.809744
45  2021-03-20 00:00:00  2024-06-12 08:07:55.809744
46  2021-07-07 00:00:00  2024-06-12 08:07:55.809744
47  2020-07-30 00:00:00  2024-06-12 08:07:55.809744
48  2021-12-31 00:00:00  2024-06-12 08:07:55.809744
49  2020-04-01 00:00:00  2024-06-12 08:07:55.809744

Data from ProductVariant:

    VariantID  ProductID              VariantName             Attributes  \
0           1          1   Variant 1 for Product 1   Color: Blue; Size: L
1           2          1   Variant 2 for Product 1  Color: Green; Size: L
2           3          2   Variant 1 for Product 2  Color: Green; Size: L
3           4          2   Variant 2 for Product 2   Color: Blue; Size: L
4           5          3   Variant 1 for Product 3   Color: Blue; Size: L
..        ...        ...                       ...                    ...
95         96         48  Variant 2 for Product 48   Color: Blue; Size: L
96         97         49  Variant 1 for Product 49  Color: Green; Size: L
97         98         49  Variant 2 for Product 49    Color: Red; Size: S
98         99         50  Variant 1 for Product 50  Color: Green; Size: L
99        100         50  Variant 2 for Product 50    Color: Red; Size: L

            DateCreated               LastUpdated
0   2021-05-27 00:00:00  2024-06-12 08:07:55.813685
1   2021-08-27 00:00:00  2024-06-12 08:07:55.813685
2   2021-03-15 00:00:00  2024-06-12 08:07:55.813685
3   2020-12-11 00:00:00  2024-06-12 08:07:55.813685
4   2021-09-25 00:00:00  2024-06-12 08:07:55.813685
..                  ...                       ...
95  2020-04-17 00:00:00  2024-06-12 08:07:55.814709
96  2021-01-04 00:00:00  2024-06-12 08:07:55.814709
97  2021-11-25 00:00:00  2024-06-12 08:07:55.814709
98  2020-05-21 00:00:00  2024-06-12 08:07:55.814709
99  2020-03-18 00:00:00  2024-06-12 08:07:55.814709
```

```
[100 rows x 6 columns]

Data from ProductPriceHistory:

     PriceID  VariantID   Price            StartDate              EndDate
0          1          1  427.11  2020-12-10 00:00:00  2021-08-20 00:00:00
1          2          1  106.02  2021-08-21 00:00:00  2022-01-26 00:00:00
2          3          1  645.17  2022-01-27 00:00:00  2023-01-08 00:00:00
3          4          2   22.71  2021-02-20 00:00:00  2021-04-28 00:00:00
4          5          2  415.68  2021-04-29 00:00:00  2021-10-29 00:00:00
..       ...        ...     ...                  ...                  ...
295      296         99  476.88  2020-08-31 00:00:00  2020-12-07 00:00:00
296      297         99  255.38  2020-12-08 00:00:00  2021-10-18 00:00:00
297      298        100  529.81  2020-12-01 00:00:00  2021-04-25 00:00:00
298      299        100  406.39  2021-04-26 00:00:00  2022-03-18 00:00:00
299      300        100  884.90  2022-03-19 00:00:00  2022-08-23 00:00:00

[300 rows x 5 columns]

Data from Order:

     OrderID  CustomerID            OrderDate  TotalAmount
0          1           1  2023-12-15 00:00:00  4032.901672
1          2           2  2023-08-21 00:00:00  5913.462733
2          3           3  2022-09-12 00:00:00  7232.220148
3          4           4  2022-10-05 00:00:00    47.092696
4          5           5  2023-09-27 00:00:00  4594.339536
..       ...         ...                  ...          ...
319      320          10           2023-07-10   334.650000
320      321          10           2023-07-10   945.190000
321      322          10           2023-07-10   326.770000
322      323          10           2023-07-10   409.580000
323      324          10           2023-07-10   533.490000

[324 rows x 4 columns]

Data from OrderItem:

     OrderItemID  OrderID  VariantID  Quantity       Price   Discount  \
0              1        1         70         3  928.483071  58.005362
1              2        1         71         3  116.875380  24.098524
```

```
2             3        1         35         1  520.027229  87.797977
3             4        1         49         4  221.959264  57.811318
4             5        1         46         1   60.062738   5.745796
..          ...      ...        ...       ...         ...        ...
280         281      100         79         5  138.677435  34.314471
281         282      100         23         5  129.016864   7.876539
282         283      100         61         1  301.665191  22.041414
283         284      100         89         3  427.083276  43.392971
284         285      100         34         4  473.724762  25.826193

      TotalPrice
0    2611.433125
1     278.330567
2     432.229252
3     656.591786
4      54.316942
..           ...
280   521.814823
281   605.701622
282   279.623776
283  1151.070916
284  1791.594275

[285 rows x 7 columns]
```

Explanation of Code for Fetching and Displaying Data from SQLite Database Tables

1. Connecting to the SQLite Database

- **Connection**:
  - Establishes a connection to the SQLite database named `ecommerce.db` using `sqlite3.connect('ecommerce.db')`.
  - Initializes a cursor (`cursor`) to execute SQL commands within the database.

2. Defining `fetch_data` Function

- **Function Purpose**:

- `fetch_data(table_name)`: Defines a function that retrieves and displays data from a specified table (`table_name`) within the connected SQLite database.

- **Query Construction**:
  - Constructs a SQL query string (`query`) to select all columns (`SELECT *`) from the specified table (`table_name`).
  - Uses double quotes (`\"`) around the table name to handle cases where the table name contains special characters or spaces.
- **Data Retrieval and Display**:
  - Executes the SQL query using `pd.read_sql_query(query, conn)` to fetch data from the database into a Pandas DataFrame (`df`).
  - Prints a header indicating the table name (`print(f"\nData from {table_name}:\n")`) for clarity.
  - Displays the DataFrame (`print(df)`) showing the fetched data in a tabular format.

3. Iterating Over Tables and Fetching Data

- **Table List (`tables`)**:
  - Defines a list (`tables`) containing names of tables (`['Customer', 'Product', 'ProductVariant', 'ProductPriceHistory', 'Order', 'OrderItem']`) from which data will be fetched.
- **Fetch Loop**:
  - Iterates through each table name (`table`) in the `tables` list.
  - Calls the `fetch_data(table)` function for each table name to fetch and display its data.

4. Closing the Database Connection

- **Connection Closure**:
  - Closes the connection to the SQLite database using `conn.close()` to release resources after fetching and displaying data.

5. Output

- **Output Confirmation**:
  - After fetching and displaying data from all tables, the script completes and implicitly confirms successful execution.

In [ ]:
```
# Connect to the SQLite database
conn = sqlite3.connect('ecommerce.db')
```

```
# 1. Retrieve the top 5 customers who have made the highest average order amounts in the last 6 months
top_customers_df = pd.read_sql_query("""
    SELECT
        c.CustomerID,
        c.Name,
        AVG(o.TotalAmount) AS AvgOrderAmount
    FROM
        Customer c
    JOIN
        `Order` o ON c.CustomerID = o.CustomerID
    WHERE
        o.OrderDate >= DATE('now', '-6 months')
    GROUP BY
        c.CustomerID,
        c.Name
    ORDER BY
        AvgOrderAmount DESC
    LIMIT
        5;
""", conn)

# 2. Retrieve the list of customers whose order value is lower this year as compared to the previous year
lower_order_customers_df = pd.read_sql_query("""
    WITH OrderTotals AS (
        SELECT
            CustomerID,
            SUM(TotalAmount) AS TotalOrderAmount,
            STRFTIME('%Y', OrderDate) AS OrderYear
        FROM
            `Order`
        GROUP BY
            CustomerID,
            OrderYear
    )
    SELECT
        o1.CustomerID,
        c.Name
    FROM
        OrderTotals o1
    JOIN
```

```
        OrderTotals o2 ON o1.CustomerID = o2.CustomerID
    JOIN
        Customer c ON o1.CustomerID = c.CustomerID
    WHERE
        o1.OrderYear = STRFTIME('%Y', DATE('now')) AND
        o2.OrderYear = STRFTIME('%Y', DATE('now', '-1 year')) AND
        o1.TotalOrderAmount < o2.TotalOrderAmount;
""", conn)

# 3. Create a table showing cumulative purchase by a particular customer. Show the breakup of cumulative purchases by product
cumulative_purchase_df = pd.read_sql_query("""
    SELECT
        o.CustomerID,
        c.Name AS CustomerName,
        p.Category,
        SUM(o.TotalAmount) AS CumulativePurchase
    FROM
        `Order` o
    JOIN
        Customer c ON o.CustomerID = c.CustomerID
    JOIN
        OrderItem oi ON o.OrderID = oi.OrderID
    JOIN
        ProductVariant pv ON oi.VariantID = pv.VariantID
    JOIN
        Product p ON pv.ProductID = p.ProductID
    GROUP BY
        o.CustomerID,
        c.Name,
        p.Category
    ORDER BY
        o.CustomerID,
        p.Category;
""", conn)

# 4. Retrieve the list of top 5 selling products. Further, bifurcate the sales by product variants
top_selling_products_df = pd.read_sql_query("""
    WITH ProductSales AS (
        SELECT
            pv.ProductID,
            pv.VariantID,
```

```
            p.ProductName,
            pv.VariantName,
            SUM(oi.Quantity) AS TotalQuantity
        FROM
            OrderItem oi
        JOIN
            ProductVariant pv ON oi.VariantID = pv.VariantID
        JOIN
            Product p ON pv.ProductID = p.ProductID
        GROUP BY
            pv.ProductID,
            pv.VariantID,
            p.ProductName,
            pv.VariantName
    )
    SELECT
        ps.ProductID,
        ps.ProductName,
        ps.VariantID,
        ps.VariantName,
        ps.TotalQuantity
    FROM
        ProductSales ps
    JOIN
        (
            SELECT
                ProductID,
                MAX(TotalQuantity) AS MaxQuantity
            FROM
                ProductSales
            GROUP BY
                ProductID
            ORDER BY
                MaxQuantity DESC
            LIMIT
                5
        ) AS top_products ON ps.ProductID = top_products.ProductID
    ORDER BY
        ps.ProductID,
        ps.TotalQuantity DESC;
""", conn)
```

```
# Close the connection
conn.close()
```

Explanation of SQL Queries and Pandas DataFrames Operations

1. Query to Retrieve Top 5 Customers by Highest Average Order Amounts in Last 6 Months

- **SQL Query**:
  - **Purpose**: Retrieves the top 5 customers who have the highest average order amounts in the last 6 months.
  - **Tables Used**: `Customer` (aliased as `c`) and `Order` (aliased as `o`).
  - **Criteria**:
    - Computes the average order amount (`AVG(o.TotalAmount) AS AvgOrderAmount`) for each customer.
    - Filters orders placed in the last 6 months (`o.OrderDate >= DATE('now', '-6 months')`).
  - **Grouping and Sorting**:
    - Groups results by `CustomerID` and `Name`.
    - Orders results by `AvgOrderAmount` in descending order.
    - Limits the output to 5 rows (`LIMIT 5`).
- **Pandas DataFrame Operation**:
  - `top_customers_df`: Stores the result of the SQL query as a Pandas DataFrame using `pd.read_sql_query()`.

2. Query to Retrieve Customers with Lower Order Value This Year Compared to Previous Year

- **SQL Query**:
  - **Purpose**: Retrieves customers whose total order value this year is lower compared to the previous year.
  - **Tables Used**: `Order` (aliased as `o1` and `o2`), `Customer` (aliased as `c`).
  - **Subquery (`OrderTotals`)**:
    - Calculates total order amount (`SUM(TotalAmount) AS TotalOrderAmount`) for each customer (`CustomerID`) grouped by order year (`OrderYear`).
  - **Criteria**:

    - Compares total order amounts between the current year (`o1.OrderYear = STRFTIME('%Y', DATE('now'))`) and the previous year (`o2.OrderYear = STRFTIME('%Y', DATE('now', '-1 year'))`).
    - Selects customers where the order amount this year (`o1.TotalOrderAmount`) is less than that of the previous year (`o2.TotalOrderAmount`).
- **Pandas DataFrame Operation**:
  - `lower_order_customers_df` : Stores the result of the SQL query as a Pandas DataFrame using `pd.read_sql_query()` .

3. Query to Create a Table Showing Cumulative Purchase by Customer, Broken Down by Product Category

- **SQL Query**:
  - **Purpose**: Creates a table showing cumulative purchases by each customer, categorized by product category.
  - **Tables Used**: `Order` (aliased as `o`), `Customer` (aliased as `c`), `OrderItem` (aliased as `oi`), `ProductVariant` (aliased as `pv`), `Product` (aliased as `p`).
  - **Criteria**:
    - Joins tables to link orders (`o`), customers (`c`), order items (`oi`), product variants (`pv`), and products (`p`).
    - Computes the sum of total order amounts (`SUM(o.TotalAmount)`) grouped by `CustomerID` , `CustomerName` , and `Category` .
    - Orders results by `CustomerID` and `Category` .
- **Pandas DataFrame Operation**:
  - `cumulative_purchase_df` : Stores the result of the SQL query as a Pandas DataFrame using `pd.read_sql_query()` .

4. Query to Retrieve Top 5 Selling Products and Their Variants

- **SQL Query**:
  - **Purpose**: Retrieves the top 5 selling products and their variants based on total quantity sold.
  - **Tables Used**: `OrderItem` (aliased as `oi`), `ProductVariant` (aliased as `pv`), `Product` (aliased as `p`).
  - **Subquery (`ProductSales`)**:
    - Computes total quantity sold (`SUM(oi.Quantity) AS TotalQuantity`) for each product variant (`ProductID` , `VariantID` , `ProductName` , `VariantName`).
  - **Criteria**:

    - Selects products (`ps`) from `ProductSales` where the total quantity sold (`ps.TotalQuantity`) is among the top 5 (`LIMIT 5`).
    - Orders results by `ProductID` and total quantity (`TotalQuantity`) in descending order.
- **Pandas DataFrame Operation**:
    - `top_selling_products_df` : Stores the result of the SQL query as a Pandas DataFrame using `pd.read_sql_query()` .

5. Closing the Database Connection

- **Connection Closure**:
    - Closes the connection to the SQLite database (`conn.close()`) to release resources after all queries have been executed and results fetched into Pandas DataFrames.

Top 5 customers with highest average order amounts in the last 6 months:"

In [ ]:
```
top_customers_df
```

Out[ ]:

	CustomerID	Name	AvgOrderAmount
0	1	Customer 1	829.872111
1	8	Customer 8	672.551000
2	9	Customer 9	615.484000
3	6	Customer 6	584.034000
4	7	Customer 7	573.006000

Customers with order value lower this year compared to previous year

In [ ]:
```
lower_order_customers_df
```

Out[ ]:

	CustomerID	Name
0	1	Customer 1
1	2	Customer 2
2	4	Customer 4
3	5	Customer 5
4	6	Customer 6
5	7	Customer 7
6	8	Customer 8
7	10	Customer 10

Cumulative purchase breakup by product category:

In [ ]:
```
cumulative_purchase_df
```

Out[ ]:

	CustomerID	CustomerName	Category	CumulativePurchase
0	1	Customer 1	Books	4032.901672
1	1	Customer 1	Clothing	12098.705016
2	1	Customer 1	Electronics	4032.901672
3	2	Customer 2	Books	5913.462733
4	2	Customer 2	Clothing	5913.462733
...	...	...	...	...
185	99	Customer 99	Clothing	12999.274458
186	99	Customer 99	Electronics	12999.274458
187	100	Customer 100	Books	8699.610825
188	100	Customer 100	Clothing	8699.610825
189	100	Customer 100	Electronics	4349.805413

190 rows × 4 columns

Top 5 selling products with sales bifurcated by product variants:

In [ ]:
```
top_selling_products_df
```

Out[ ]:

	ProductID	ProductName	VariantID	VariantName	TotalQuantity
0	13	Product 13	26	Variant 2 for Product 13	24
1	13	Product 13	25	Variant 1 for Product 13	5
2	32	Product 32	63	Variant 1 for Product 32	31
3	32	Product 32	64	Variant 2 for Product 32	11
4	33	Product 33	65	Variant 1 for Product 33	25
5	33	Product 33	66	Variant 2 for Product 33	11
6	40	Product 40	79	Variant 1 for Product 40	22
7	40	Product 40	80	Variant 2 for Product 40	5
8	47	Product 47	94	Variant 2 for Product 47	22
9	47	Product 47	93	Variant 1 for Product 47	1