Student Name : Sonal Shivaji Sherkar
Student Id : 22020085

Introduction

This Python program generates a simulated online retail database, comprising three main tables: Customers, Orders, and Products. The Customers table stores diverse customer details such as names, contact information, and addresses, providing a foundation for understanding customer demographics. The Orders table tracks order-specific information, including dates, amounts, and shipping details, linked to customers via unique identifiers. Meanwhile, the Products table houses product-related data such as names, categories, prices, and stock quantities. By populating these tables with randomized data, the program offers a platform for analysing customer behaviour, purchase patterns, and product performance, empowering retailers to improve their operations and enhance the online shopping experience.

Data Set Generation

```
import sqlite3
import random
from faker import Faker
from datetime import datetime, timedelta
```

This Python script utilizes several modules to generate fake data and interact with a SQLite database. The sqlite3 module enables communication with SQLite databases, facilitating the creation and manipulation of tables and data. Through the random module, the script introduces randomness into the generated data, adding variability to simulate real-world scenarios. The Faker library plays a crucial role in generating realistic-looking fake data, including names, addresses, emails, and other fields. By utilizing Faker's capabilities, the script ensures that the generated data closely resembles authentic information without compromising privacy or security. Additionally, the datetime module aids in generating random dates within specified ranges, contributing to the diversity of the dataset. Overall, this script serves to populate a SQLite database with synthetic yet believable data, suitable for various purposes such as testing, development, or demonstration, where authentic but non-sensitive data is required. Through the synergy of these modules, the script streamlines the process of creating diverse datasets, offering a valuable tool for database management and application development endeavours.

```
# Initialize Faker library
fake = Faker()

# Connect to SQLite database
conn = sqlite3.connect('online_retail.db')
c = conn.cursor()
```

The code initializes the Faker library for generating fake data. It establishes a connection to a SQLite database named 'online_retail.db'. The database connection is handled through the sqlite3 module. This setup enables the script to interact with the SQLite database, allowing for the creation, insertion, and data manipulation. By leveraging Faker's capabilities and SQLite's database management features, the script can populate the database with realistic but synthetic data, crucial for various applications like testing and development.

```
# Create Customers table
c.execute('''CREATE TABLE IF NOT EXISTS Customers (
                customer_id INTEGER PRIMARY KEY,
                first_name TEXT,
                last_name TEXT,
                email TEXT,
                phone_number TEXT,
                address TEXT,
                city TEXT,
                country TEXT
            )''')

# Create Orders table
c.execute('''CREATE TABLE IF NOT EXISTS Orders (
                order_id INTEGER PRIMARY KEY,
                customer_id INTEGER,
                order_date TEXT,
                total_amount REAL,
                shipping_address TEXT,
                shipping_city TEXT,
                shipping_country TEXT,
                FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
            )''')

# Create Products table
c.execute('''CREATE TABLE IF NOT EXISTS Products (
                product_id INTEGER PRIMARY KEY,
                product_name TEXT,
                category TEXT,
                price REAL,
                description TEXT,
                stock_quantity INTEGER,
                supplier TEXT
            )''')
```

In above, we are defining the schema for three tables in a SQLite database: Customers, Orders, and Products. For the Customers table, we specify columns such as customer_id, first_name, last_name, email, phone_number, address, city, and country to store customer information. The Orders table includes columns like order_id, customer_id, order_date, total_amount, shipping_address, shipping_city, and shipping_country to represent order details. Finally, the Products table consists of columns such as product_id, product_name, category, price, description, stock_quantity, and supplier to manage product information.

By executing these SQL statements, we create a structured database schema capable of storing data pertaining to customers, their orders, and product details. This schema allows for efficient data organization and retrieval, facilitating tasks like customer management, order processing, and inventory tracking. With this setup, the program provides a foundation for building an online retail system, enabling functionalities such as customer registration, order placement, and product management.

```
# Generate and insert random data for Customers
for _ in range(1000):
    first_name = fake.first_name()
    last_name = fake.last_name()
    email = fake.email()
    phone_number = fake.phone_number()
    address = fake.street_address()
    city = fake.city()
    country = fake.country()

    c.execute("INSERT INTO Customers (first_name, last_name, email, phone_number, address, city, country) VALUES (?, ?, ?, ?, ?, ?, ?)",
              (first_name, last_name, email, phone_number, address, city, country))

# Generate and insert random data for Products
for _ in range(100):
    product_name = fake.word()
    category = fake.word()
    price = round(random.uniform(10, 1000), 2)
    description = fake.sentence()
    stock_quantity = random.randint(1, 100)
    supplier = fake.company()

    c.execute("INSERT INTO Products (product_name, category, price, description, stock_quantity, supplier) VALUES (?, ?, ?, ?, ?, ?)",
              (product_name, category, price, description, stock_quantity, supplier))
```

```
# Generate and insert random data for Orders
for _ in range(1000):
    customer_id = random.randint(1, 1000)
    order_date = fake.date_time_between(start_date='-1y', end_date='now')
    total_amount = round(random.uniform(10, 1000), 2)
    shipping_address = fake.street_address()
    shipping_city = fake.city()
    shipping_country = fake.country()

    c.execute("INSERT INTO Orders (customer_id, order_date, total_amount, shipping_address, shipping_city, shipping_country) VALUES (?, ?, ?, ?, ?, ?)",
              (customer_id, order_date.strftime('%Y-%m-%d'), total_amount, shipping_address, shipping_city, shipping_country))
```

In this, we're simulating the creation of a database for an online retail system. Firstly, we generate and insert data for 1000 customers, each with a randomly generated first name, last name, email address, phone number, street address, city, and country. Next, we proceed to populate the Products table with 100 random products, specifying details such as product name, category, price, description, stock quantity, and supplier. Then, we simulate the creation of 1000 orders by associating each order with a randomly selected customer ID, generating a random order date within the past year up to the current date, along with total amount, shipping address, city, and country.

This data generation process emulates the real-world scenario of an online retail platform where customers, products, and orders need to be managed. The generated data can be used for various purposes such as testing the functionality of the retail system, analysing user behaviour, optimizing inventory management, and enhancing customer experience. By populating the database with synthetic yet realistic data, developers can assess the performance and reliability of the system under different scenarios, ensuring its robustness and effectiveness in serving customers' needs in the UK market.

```
# Commit changes and close connection
conn.commit()
conn.close()
```

In these lines of code, the changes to the SQLite database are locked in by committing them, guaranteeing they are solidly stored. After that, the database connection is wrapped up, freeing up system resources. This completes the data insertion process and keeps the database in tip-top condition.

Sample database queries involving JOINs and selections

This query uses the SELECT command in SQL to fetch the first five rows from the table. It employs the LIMIT clause to control the number of rows returned, offering a succinct snapshot of the dataset. This approach proves useful for accessing and showcasing a brief overview of the data.

```
# Execute the SQL query
c.execute("SELECT * FROM Customers LIMIT 5")

# Fetch and print the results
rows = c.fetchall()
for row in rows:
    print(row)
```

```
In [31]: runfile('C:/Users/Dell/Desktop/untitled2.py', wdir='C:/Users/Dell/Desktop')
(1, 'Jennifer', 'Khan', 'belinda64@example.net', '001-403-693-1390', '889 Rachel Walks', 'Roseside', 'Saint Martin')
(2, 'Greg', 'Johnson', 'thomas26@example.com', '001-217-362-1988x76581', '91269 Emily Roads Suite 762', 'South
Michaelfort', 'Monaco')
(3, 'Lisa', 'Curry', 'tony76@example.net', '405-475-0776x342', '120 Lori Tunnel Suite 657', 'Brentview', 'Nauru')
(4, 'William', 'Warner', 'kmorris@example.net', '+1-381-984-9871x685', '48635 Virginia Stravenue Apt. 145', 'Reedton',
'Austria')
(5, 'Luis', 'Vargas', 'michaeldixon@example.com', '959.685.0173x886', '54160 Heather Orchard Apt. 635', 'West Sarahmouth',
'Mexico')
```

Utilizing an INNER JOIN, this query merges two tables, which acts as the foreign key. The aim is to retrieve comprehensive details by uniting information from both tables, providing a holistic view of the associated data.

Database Queries:

1. Selection Query:

```
1
2   SELECT * FROM Customers;
```

	customer_id	first_name	last_name	email	phone_number	address	city
1	1	Jennifer	Khan	belinda64@example.net	001-403-693-1390	889 Rachel Walks	Roseside
2	2	Greg	Johnson	thomas26@example.com	001-217-362-1988x76581	91269 Emily Roads Suite 762	South Michaelfort
3	3	Lisa	Curry	tony76@example.net	405-475-0776x342	120 Lori Tunnel Suite 657	Brentview
4	4	William	Warner	kmorris@example.net	+1-381-984-9871x685	48635 Virginia Stravenue Apt. 145	Reedton
5	5	Luis	Vargas	michaeldixon@example.com	959.685.0173x886	54160 Heather Orchard Apt. 635	West Sarahmouth
6	6	Teresa	Rogers	blake96@example.org	971-473-6958x232	5269 Nathaniel Locks Suite 200	Port Adam
7	7	Erin	Lawson	sdelgado@example.com	7135753820	03836 Bryan Plains Suite 169	New Jennifer
8	8	Patrick	Ellis	shermandaniel@example.org	001-643-768-0824x89115	74847 Paul Island	West Maria
9	9	Benjamin	Hunter	anita84@example.net	(735)338-8463x3410	4136 Rocha Centers Apt. 160	Lake Vickichester
10	10	Michelle	Adams	allison19@example.org	(914)555-7437	05913 Michael Trail Suite 728	Williamsonberg
11	11	Calvin	Torres	mary87@example.net	(249)481-3176	7546 Weiss Fields	Lake Matthewton
12	12	Anthony	Griffith	richard10@example.org	+1-683-452-5889	7871 Jonathan Rest Apt. 260	North Robert
13	13	Logan	Hill	erinperry@example.net	+1-499-405-2752x0894	304 Lee Lodge	North Michael
14	14	Tyler	Hall	barronangela@example.org	3842469397	1497 Matthew Prairie	Lake Markfurt
15	15	Mary	Kelley	robert30@example.org	(587)401-2640x75666	72860 Byrd Drive Suite 458	Andresstad
16	16	Kathleen	Anderson	ibrown@example.org	001-817-261-1621x1450	42467 Smith Valleys	Port Patrick
17	17	Steven	Graham	kdavis@example.com	828-859-4214	2596 Nunez Rapid	Allentown
18	18	Heidi	Nichols	teresa02@example.net	(902)827-0818x972	816 Austin Unions Apt. 791	Thomaschester
19	19	Abigail	Harrell	christopher24@example.net	001-610-665-1985x70814	095 Moody Lodge	New Kimberly
20	20	Madison	Patel	smithjerry@example.com	001-569-228-0826x74525	49436 Lewis Meadows Apt. 001	Lake Dillonchester

2. Joins :

```
1
2   SELECT * FROM Products ORDER BY product_name ASC LIMIT 4;
```

	product_id	product_name	category	price	description	stock_quantity	supplier
1	399	Congress	color	465.04	Behavior let Democrat themselves or...	23	Davis, Pittman and Freeman
2	23	Democrat	foot	629.41	Occur health first always.	13	Perkins Group
3	495	Democrat	friend	32.34	Perhaps actually head state price.	33	Huerta and Sons
4	694	Democrat	sea	161.04	Finally science summer able military ...	56	Brown, Henry and Rojas

3. Aggregate Function :

```
1
2   SELECT AVG (price) FROM Products;
```

	AVG (price)
1	489.87905

```
1   SELECT category, AVG (price)
2   FROM Products
3   GROUP BY category;
```

	category	AVG (price)
1	American	534.61
2	Beauty	644.235714285714
3	Books	450.7775
4	Clothing	534.605333333333
5	Democrat	541.97
6	Electronics	511.046
7	Home & Kitchen	386.893636363636

```sql
SELECT Customers.*, Orders.order_id, Orders.order_date, Orders.total_amount,
          Orders.shipping_address, Orders.shipping_city, Orders.shipping_country
FROM Customers
INNER JOIN Orders ON Customers.customer_id = Orders.customer_id
LIMIT 10
```

	customer_id	first_name	last_name	email	phone_number	address	city	country	c
1	636	Julie	Franklin	jameswright@example.com	896-351-1733x612	12771 Weeks Ports	West Michelle	Nicaragua	
2	855	Micheal	Graham	lisaosborn@example.com	(624)513-8901x3604	492 Higgins Greens Apt. 236	Kimberlyburgh	Georgia	
3	793	Michelle	Crosby	gardnermaria@example.org	(929)851-5174x9304	231 Barker Club Apt. 618	North Angela	Czech Republic	
4	612	David	Cummings	jonestracey@example.org	2634270104	78380 Matthew Mountain Suite 701	Matthewtown	Jordan	
5	601	Valerie	Cooper	hharrison@example.org	(665)312-9359	582 Kerr Mountain Apt. 950	Rodriguezfurt	Peru	
6	325	Jason	Simmons	lparrish@example.org	001-900-614-7914x339	6312 White Harbor	Lopezton	Madagascar	
7	651	William	Wall	fbrown@example.com	+1-393-792-9079	29557 Bass Freeway Suite 602	Hugheston	Rwanda	
8	385	Stephanie	Thomas	riverasophia@example.org	+1-739-269-1804x164	21210 Scott Summit	North Brianhaven	Maldives	
9	764	Courtney	Lamb	ptaylor@example.com	945-582-7054x96410	4363 Lori Ville Suite 731	Kellyport	Bahamas	
10	892	Shane	Lewis	scottmerritt@example.com	+1-335-383-2123x274	700 Collins Locks Apt. 035	Harmonchester	Mali	