SQL ASSIGNMENT

Name : Vegu Mahesh Babu
Studentid: 22034351

Table of Contents

Online E-commerce Database

Database Generation:

For database generation, Python was employed to generate the data, considering the need for a substantial dataset of at least 1000 rows. Various Python libraries were utilized to create realistic data for different columns.

A database is created with 3 tables in it.

- **Table-1: Users Table**
  - This table comprises 9 columns with 1001 rows, and I have used Faker library in Python to create realistic-looking Usernames, countries, and other information.
  - Random integers, uniform distributions, and choices from predefined lists were employed to simulate diverse data types such as product prices, ratings, user IDs, ages, and wallet balances.

- **Table -2: Products Table**
  - This table consists of 5 columns with 48 rows. To enhance realism, product names and categories were manually added, and the Random library in Python was employed for generating other columns' data.

- **Table - 3: Purchases Table**
  - This table consists of 3 columns and 1001 rows, the three columns are (UserId, ProductId, PurchaseDate) which are combined to form a composite key, which ensures that each purchase record is unique.

And in the above tables, Ensured to maintain at least one nominal data, ordinal data, ratio data, interval data.

Database Schema:

The database schema includes two tables: "Users", "Products" and “Purchases”. These tables are connected through the use of primary keys, foreign keys and composite keys, providing a relational structure. Here is a summary of the schema:

Table: Products

Columns:

- **ProductId (TEXT, Primary Key)**
- **ProductName (TEXT)**
- **ProductPrices (INTEGER)**
- **Category (TEXT)**
- **Rating (REAL)**

Table: Customers
Columns:

- **UserId (INTEGER, Primary Key)**
- **UserName (TEXT)**
- **ProductId (TEXT)**
- **UsersAges (TEXT)**
- **Region (TEXT)**
- **Country (TEXT)**
- **Postal (TEXT)**
- **WalletBalance (REAL)**
- **OrderDate Date**

Table: Purchases
Columns:

- **UserId** (INTEGER, Foreign Key, References Users(UserId))
- **ProductId** (TEXT, Foreign Key, References Products(ProductId))
- **PurchaseDate** (DATE)

Composite Key:

- (**UserId**, **ProductId**, **PurchaseDate**)

Foreign Key and Composite Key Utilization:

- The **UserId** and **ProductId** in the **Purchases** table serves as a foreign key that references the **UserId** in the **Users** table and **ProductId** in the **Products** table.
- The composite key (**UserId**, **ProductId**, **PurchaseDate**) in the **Purchases** table ensures that each purchase is uniquely identified by the combination of the user, the product, and the purchase date.

Justification for Separate Tables:

Instead of consolidating all information into a single database, I have chosen to create two distinct tables: **Users** and **Products**. The **Users** table stores comprehensive information related to users, On the other hand, the **Products** table is dedicated to storing information specifically related to products,

The database was structured into separate tables instead of consolidating them into a single table for the following reasons

Normalization:
The adoption of distinct tables adheres to the fundamental principles of database normalization. Each table is dedicated to specific entities, minimizing redundancy, and optimizing the storage and retrieval of data. For example, the 'User' table houses user

details, the 'Product' table encapsulates product information, and the 'Purchase' table documents transaction data. This methodology streamlines data management, enhancing the overall efficiency of the database.

Data Integrity:
The employment of separate tables, coupled with established relationships, contributes significantly to preserving data integrity. For instance, the 'Purchase' table utilizes foreign keys like 'UserId,' referencing the 'User' table, ensuring the association of each purchase with an existing user. The incorporation of a composite key in the 'Purchase' table serves to prevent redundant entries, elevating the precision of the data.

Scalability:
A meticulously structured database with isolated tables provides scalability. As the e-commerce platform expands, the addition of new records to pertinent tables seamlessly occurs without disrupting the overarching database structure. This scalability proves crucial for accommodating a burgeoning customer base, diverse product catalog, and increased transaction volumes.

Data Retention and Security:
Ethical database management involves well-defined protocols for data retention, deletion, or anonymization, especially for sensitive customer information. Prioritizing data security includes timely identification and resolution of vulnerabilities, demonstrating a commitment to ethical standards and safeguarding customer data from potential threats.

Ethical Discussions:

Taking ethical measures involves clear guidelines for handling data, especially sensitive customer information, including how and when to retain, delete, or anonymize it. Prioritizing data security means promptly addressing vulnerabilities, showcasing a commitment to ethical standards. This approach ensures customer data is protected, reflecting a commitment to ethical data practices throughout the database's life. And also few of the points are

Data Privacy and Security:

- **Prudent Handling:** Synthetic data safeguards user privacy and ensures security measures are in place for customer and food-related information.

Consent and Transparency:

- **Transparent Practices:** The code emphasizes transparent communication and simplifies consent procedures, even with synthetic data.

Regulatory Compliance:

- **Compliance Assurance:** The database structure aligns with regulations, showcasing ethical data handling practices and protecting sensitive information.

Example Queries

Basic Selections of Tables:

- Select all products with their names and prices:

```sql
SELECT ProductName, ProductPrices FROM products;
```

	ProductName	ProductPrices
1	Double cot Bed	29
2	15" Tire	24
3	Boult Earbuds	10
4	D&G Perfume	30

```
Execution finished without errors.
Result: 48 rows returned in 7ms
At line 1:
SELECT ProductName, ProductPrices FROM products;
```

Joining Tables:

- Select usernames and the products they purchased, along with product names & prices:

```sql
SELECT u.UserName, p.ProductName, p.ProductPrices
FROM Users u
JOIN products p ON u.ProductId = p.ProductId;
```

	UserName	ProductName	ProductPrices
1	Michael Wilson	Boult Earbuds	10
2	Martin Reed	Blue Sofa	26
3	Gary Reed	Philips Hairdryer	26
4	Charles Wade	5KG Dumbbells	7
5	Wendy Lambert	FitBit Sports Water Bottle	19
6	Edward Jimenez	Dashboard Camera	6
7	James Scott	FitBit Fitness Tracker	24

```
Execution finished without errors.
Result: 1001 rows returned in 5ms
At line 1:
SELECT u.UserName, p.ProductName, p.ProductPrices
FROM Users u
JOIN products p ON u.ProductId = p.ProductId;
```

Filtering with WHERE Clause:

- Select products with prices greater than 100:

```sql
SELECT ProductName, ProductPrices
FROM products
WHERE ProductPrices > 20;
```

	ProductName	ProductPrices
1	Yellow Sneakers	24
2	JBL headset	24
3	Philips Hairdryer	26
4	Car rearview mirror	24
5	Car Seat Covers	23
6	GPS Navigation System	28

```
Execution finished without errors.
Result: 17 rows returned in 8ms
At line 1:
SELECT ProductName, ProductPrices
FROM products
WHERE ProductPrices > 20;
```

Aggregation with GROUP BY:

- Find the average rating for each category of products:

SQL 1 | SQL 3 | SQL 4

```sql
SELECT Category, AVG(Rating) AS AvgRating
FROM products
GROUP BY Category
```

	Category	AvgRating
1	Automotive	2.50125
2	Beauty	2.575
3	Clothing	2.61375
4	Electronics	2.8725
5	Home	1.64625
6	Sports	3.31875

```
Execution finished without errors.
Result: 6 rows returned in 7ms
At line 1:
SELECT Category, AVG(Rating) AS AvgRating
FROM products
GROUP BY Category
```

Sorting with ORDER BY:

- Select customer names and their ages, sorted by age in descending order:

SQL 1 SQL 3 SQL 4 SC

```sql
SELECT UserName, UserAges
FROM Users
ORDER BY UserAges ASC;
```

	UserName	UserAges
1	Allen Carson	15-20
2	Erin Green	15-20
3	Amber Dixon	15-20
4	David Harvey	15-20
5	Justin Berry	15-20
6	Paul James	15-20

```
Execution finished without errors.
Result: 1001 rows returned in 62ms
At line 1:
SELECT UserName, UserAges
FROM Users
ORDER BY UserAges ASC;
```

Combining Joins and Aggregation:

- Find the total number of products purchased by each customer:

SQL 1 SQL 3 SQL 4 SQL 5

```sql
SELECT u.UserName, COUNT(p.ProductId) AS TotalProductsPurchased
FROM Users u
JOIN products p ON u.ProductId = p.ProductId
GROUP BY u.UserName;
```

	UserName	TotalProductsPurchased
1	Aaron Knight	1
2	Aaron Lawrence	1
3	Aaron Mays	1
4	Aaron Price	1

```
Execution finished without errors.
Result: 994 rows returned in 63ms
At line 1:
SELECT u.UserName, COUNT(p.ProductId) AS TotalProductsPurchased
FROM Users u
JOIN products p ON u.ProductId = p.ProductId
GROUP BY u.UserName;
```

Screenshots of the code:

```
!pip install faker
```

```
import numpy as np
import pandas as pd
import random
from faker import Faker
import sqlite3
import csv
from datetime import datetime, timedelta
```

```
[5] r=1001
fake = Faker()
#product_table
categories = ["Electronics", "Clothing", "Home", "Beauty", "Sports", "Automotive"]

products_dict = {
    "Electronics": ["Samsung Z fold", "Macbook pro", "DSLR camera", "Ipad 5", "JBL headset", "Galaxy Smartwatch", "PS4 Console", "Boult Earbuds"],
    "Clothing": ["WHite T-Shirt", "Black Dress", "Blue Jeans", "Denim Jacket", "Brown Shoes", "Pink Blouse", "Green Hoodie", "Yellow Sneakers"],
    "Home": ["Blue Sofa", "Double cot Bed", "Wooden Table", "4 Chairs", "Bed Lamp", "14' Curtains", "Woolen Rug", "Bookshelf"],
    "Beauty": ["Plp Shampoo", "D&G Perfume", "Makeup Kit", "Philips Hairdryer", "Electric Toothbrush", "E45 Cream", "Maybelline Nail Polish", "Maybelline Lipstick"],
    "Sports": ["WHite Running Shoes", "Hero Bicycle", "FitBit Fitness Tracker", "Yonex Tennis Racket", "Rayban Sunglasses", "Yoga Mat", "5KG Dumbbells", "FitBit Sports Water Bottle'
    "Automotive": ["Car rearview mirror", "Yamaha Motorcycle", " 15'' Tire", "Windshield Wipers", "Car Wax", "GPS Navigation System", "Car Seat Covers", "Dashboard Camera"]
}
```

```
categories_data = []
productname_data = []
productId_data = []
selected_products = set()

while len(selected_products) < 48:
    category = random.choice(categories)
    products_in_category = products_dict[category]
    product = random.choice(products_in_category)
    product_id = f'{category[:4].upper()}{str(random.randint(1, 999)).zfill(3)}'

    if product not in selected_products:
        selected_products.add(product)
        categories_data.append(category)
        productname_data.append(product)
        productId_data.append(product_id)

#Nominal data
productId_data1 = np.random.choice(productId_data, r)

#product price ranging from 2 to 30
product_prices = [random.randint(2, 30) for i in range(48)]

#Ratio Data
# Product ratings (on a scale from 0 to 5)
product_ratings = [round(random.uniform(0, 5), 2) for i in range(48)]
```

Each product in the database is linked to a specific category.
The product ID is created based on the category to which the product belongs.
For example, for a product like "IPAD 5" categorized under "Electronics," the product ID comprises the first four letters of the category and a random alphanumeric value.
This structure makes it straightforward to discern the product's category by examining its product ID.

```
###Customer Table
#User id is 4 digit numerical number
Userid_data = [random.randint(10**4, 10**5 - 1) for i in range(r)]
random.shuffle(Userid_data)

#Nominal data
#Postal codes with alphanumeric values
Postal = [f'HP{str(i).zfill(2)}' for i in range(r)]
postal_data = np.random.choice(Postal, r)

#Direction of the Regions
Region =["East","West","North","South"]
Region_data = np.random.choice(Region, r)

#Having unique names for user ids
unique_names_list = [fake.name() for i in range(r)]
```

```
    #generating the countries of the user id
    Country_data = [fake.country() for i in range(r)]

    #Ordinal Data
    #Age ranges of the user
    ages = ['15-20', '20-30', '30-40', '40-50', '50-60', '60+']
    CustomersAges = [random.choice(ages) for i in range(r)]

    #Ratio Data
    #Balance in the user's wallet in the account
    AmountBalance = [round(random.uniform(0, 100), 2) for i in range(r)]

    #interval data
    start_date = datetime(2020, 1, 1)  #from January 1, 2020
    end_date = datetime(2023, 1, 1)    #To January 1, 2023
    orderdate = [start_date + timedelta(days=random.randint(0, (end_date - start_date).days)) for i in range(r)]
    order_date = [date.strftime('%Y-%m-%d') for date in orderdate]
```

```
Productdf = pd.DataFrame({
    'ProductName': productname_data,
    'ProductId': productId_data,
    'ProductPrices': product_prices,
    'Category': categories_data,
    'Rating': product_ratings
})

Userdf = pd.DataFrame({
    'UserId': Userid_data,
    'UserName': unique_names_list,
    'ProductId': productId_data1,
    'UserAges': CustomersAges,
    'Region': Region_data,
    'Country': Country_data,
    'Postal': postal_data,
    'WalletBalance': AmountBalance,
    'OrderDate': order_date
})

Purchasedf = pd.DataFrame({
    'UserId': Userid_data,
    'ProductId': productId_data1,
    'PurchaseDate': order_date
})
```

```
conn = sqlite3.connect('E-commerce.db')
cursor = conn.cursor()

cursor.executescript('''
    CREATE TABLE Products (
        ProductId TEXT,
        ProductName TEXT,
        ProductPrices INTEGER,
        Category TEXT,
        Rating REAL,
        PRIMARY KEY (ProductId, Category)
    );

    CREATE TABLE Users (
        UserId INTEGER PRIMARY KEY,
        UserName TEXT,
        ProductId TEXT,
        CustomersAges TEXT,
        Region TEXT,
        Country TEXT,
        Postal TEXT,
        WalletBalance INTEGER,
        OrderDate DATE,
        FOREIGN KEY (ProductId) REFERENCES Products(ProductId)
    );

     CREATE TABLE Purchases (
         UserId INTEGER,
         ProductId TEXT,
         PurchaseDate DATE,

         PRIMARY KEY (UserId, ProductId, PurchaseDate),
         FOREIGN KEY (UserId) REFERENCES Users(UserId),
         FOREIGN KEY (ProductId) REFERENCES Products(ProductId)
     )
''')

conn.commit()
conn.close()
```

```
conn = sqlite3.connect('E-commerce.db')

# Write the DataFrame to an SQLite table named 'customers'
Userdf.to_sql('Users', conn, if_exists='replace', index=False)
Productdf.to_sql('Products', conn, if_exists='replace', index=False)
Purchasedf.to_sql('Purchases', conn, if_exists='replace', index=False)

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

Screenshots of Database:

User's Table:

Table: Users

	UserId	UserName	ProductId	UserAges	Region	Country	Postal	WalletBalance	OrderDate
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	77395	Monica Harmon	HOME277	40-50	East	Papua New Guinea	HP32	76.77	2020-09-19
2	23983	Lori Gibson	HOME751	40-50	East	Myanmar	HP247	62.13	2021-10-16
3	15982	Erika Valentine	CLOT531	60+	West	Germany	HP848	74.2	2022-07-18
4	51902	Russell Rogers	CLOT399	20-30	North	Saint Vincent and the Grenadines	HP412	43.63	2020-02-06
5	48563	Jerry Holland	BEAU971	15-20	West	Zimbabwe	HP416	91.72	2022-07-18
6	74080	Thomas Jefferson	SPOR740	15-20	East	Myanmar	HP307	56.94	2022-12-23
7	32003	Melissa Clark	AUTO721	20-30	North	Mongolia	HP376	60.85	2021-03-15
8	77852	Michelle Thomas	AUTO108	15-20	West	Grenada	HP706	69.15	2021-02-22
9	86706	Cody Gardner	SPOR171	60+	South	Thailand	HP11	59.62	2021-12-25
10	94355	Ronald Wilson	SPOR592	50-60	South	Lesotho	HP171	53.66	2022-11-04
11	10700	Richard Morrison	AUTO452	60+	East	Germany	HP777	87.55	2020-12-23
12	33921	Taylor Brown	SPOR435	20-30	North	Turks and Caicos Islands	HP662	36.59	2020-05-27
13	30622	Dennis Gardner	HOME798	20-30	East	Albania	HP720	19.71	2021-10-31
14	72343	Antonio West	CLOT536	15-20	North	Puerto Rico	HP712	53.55	2022-06-16
15	84733	Leslie Taylor	SPOR740	15-20	East	Cocos (Keeling) Islands	HP71	91.84	2021-01-12
16	23646	Jeremiah Terry	ELEC882	15-20	East	Moldova	HP575	12.87	2021-10-15
17	72222	Christian Mack	BEAU690	30-40	West	Puerto Rico	HP524	8.3	2020-02-11
18	32583	Lori Smith	ELEC327	30-40	East	United Kingdom	HP310	44.46	2020-03-13

1 - 19 of 1001 Go to: 1

Products Table:

Table: Products

	ProductName	ProductId	ProductPrices	Category	Rating
	Filter	Filter	Filter	Filter	Filter
1	Dashboard Camera	AUTO096	29	Automotive	2.56
2	Rayban Sunglasses	SPOR671	18	Sports	3.23
3	Black Dress	CLOT641	19	Clothing	2.9
4	Macbook pro	ELEC382	25	Electronics	3.16
5	D&G Perfume	BEAU796	28	Beauty	3.33
6	Denim Jacket	CLOT219	15	Clothing	3.35
7	Yellow Sneakers	CLOT191	2	Clothing	2.0
8	15" Tire	AUTO188	14	Automotive	4.49
9	5KG Dumbbells	SPOR496	14	Sports	1.3
10	Boult Earbuds	ELEC844	13	Electronics	1.56
11	Ipad 5	ELEC772	19	Electronics	2.35
12	Brown Shoes	CLOT209	28	Clothing	0.62
13	Car Seat Covers	AUTO108	7	Automotive	3.02
14	Yamaha Motorcycle	AUTO505	18	Automotive	3.34
15	Yonex Tennis Racket	SPOR171	3	Sports	1.26
16	Blue Sofa	HOME098	15	Home	3.11
17	GPS Navigation System	AUTO452	9	Automotive	4.09
18	Yoga Mat	SPOR435	10	Sports	1.59

1 - 19 of 48

Purchases Table:

Table: Purchases

	UserId	ProductId	PurchaseDate
	Filter	Filter	Filter
1	77395	HOME277	2020-09-19
2	23983	HOME751	2021-10-16
3	15982	CLOT531	2022-07-18
4	51902	CLOT399	2020-02-06
5	48563	BEAU971	2022-07-18
6	74080	SPOR740	2022-12-23
7	32003	AUTO721	2021-03-15
8	77852	AUTO108	2021-02-22
9	86706	SPOR171	2021-12-25
10	94355	SPOR592	2022-11-04
11	10700	AUTO452	2020-12-23
12	33921	SPOR435	2020-05-27
13	30622	HOME798	2021-10-31
14	72343	CLOT536	2022-06-16
15	84733	SPOR740	2021-01-12
16	23646	ELEC882	2021-10-15
17	72222	BEAU690	2020-02-11
18	32583	ELEC327	2020-03-13

1 - 19 of 1001