SQL ASSIGNMENT

Name: Vegu Mahesh Babu

Studentid: 22034351

Table of Contents

Database Generation:	
Database Schema:	
Justification for Separate Table	
Ethical Discussions	
Example Queries	
Screenshots of the Code	
Screenshots of Database:	12

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 (Userld, Productld, 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.

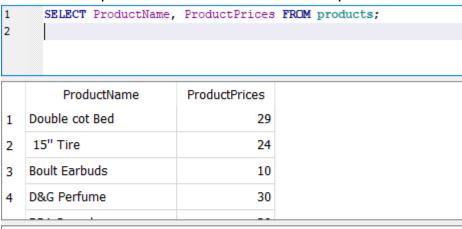
Regulatory Compliance:

• **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:



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:

```
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	lames Scott	FitRit 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:

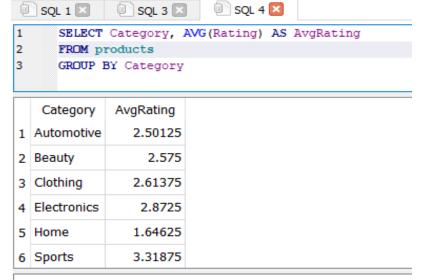
Select products with prices greater than 100:

1	SELECT ProductName, ProductPrices FROM products				
3	WHERE ProductPrice	es > 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:

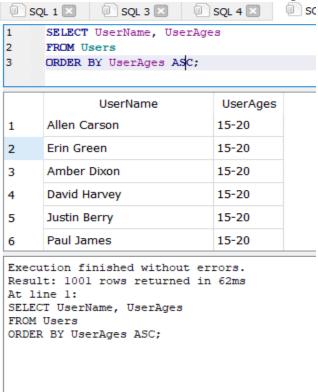
Find the average rating for each category of products:



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

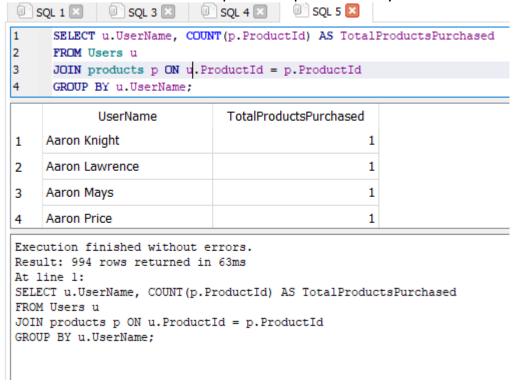
Sorting with ORDER BY:

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



Combining Joins and Aggregation:

Find the total number of products purchased by each customer:



Screenshots of the code:

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

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

product_dict = {
    "Electronics": "Samsung Z fold", "Macbook pro", "DSLR camera", "Ipad 5", "DBL headset", "Galaxy Smartwatch", "PS4 Console", "Boult Earbuds"],
    "Clothing": ["Mitte T-Shirt", "Black Dress", "Blue Jeans", "Denim Jacket", "Brown Shoes", "Pink Blouse", "Green Hoodie", "Yellow Sneakers"],
    "home": ["Blue Sofa", "Jouble cot Bed", "Mooden Table", "4 Chairs", "Bed Lamp', "14" Curtains", "Moolen Rug", "Bookshelf"],
    "Boauty": ["Plo Shampoot, "DSG Perfume," "Makepu kit", "Philips Hairdyre", "Electric Toothbrush", "E45 Cream, "Naybelline Nail Polish", "Maybelline Lipstick"],
    "Sports": ["Mitte Running Shoes", "Hero Bicycle", "FitBit Fitness Tracker", "Yonex Tennis Racket", "Bayban Sunglasses", "Yog Mat", "SKG Dumbbells", "FitBit Sports Water Bottle"
    "Automotive": ["Car rearview mirror", "Vamaha 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:</pre>
   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)]
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)]

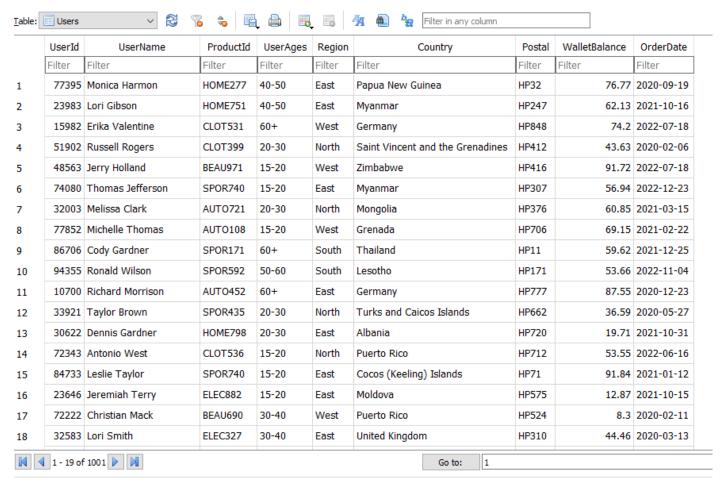
#interval data
start_date = datetime(2020, 1, 1) #from January 1, 2020
end_date = datetime(2023, 1, 1) #To January 1, 2023
orderdate = [start_date + timedelta(days=random.randint(0, (end_date - start_date).days)) for i in range(r)]
order_date = [date.strftime('%Y-%m-%d') for date in orderdate]
```

```
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:



Products Table:



Purchases Table:

