DATA MINING AND DISCOVERY

SQL ASSIGNMENT – REPORT

RESTAURANT MANAGEMENT SYSTEM -DATABASE

SUBMITTED BY: SHAHANA SHIRIN KATTIL

STUDENT ID: 22016279

Data generation:

A synthetic Restaurant Management database was generated using the Faker package and NumPy libraries. Faker generates realistic-looking data for various types, including names, addresses, phone numbers, email addresses, etc., while NumPy libraries for numerical data. The dataset includes information about restaurants, customers, orders, and dishes. During the data generation process, fictitious information is produced for various database entities, each distinguished by specific attributes. For the restaurants dataset, attributes like 'RestaurantName', 'Address', 'PhoneNumber', and 'Website' are randomly generated, while 'Cuisines' is randomly selected from a predefined list, and 'Prices' is uniformly distributed between 5 and 100. For the Customer dataset, this includes 'FirstName', 'LastName', 'EmailAddress', 'Address', 'PhoneNumber', 'RestaurantID', and 'ReviewRating', with review ratings assigned randomly based on different probabilities. The orders dataset, encompassing 'Quantity', 'MenuItem', and 'Menucategory', 'CustomerID', 'RestaurantID', 'CustomerRating', 'OrderTimestamp', and 'OrderAmount', with menu items selected from predefined categories and, quantities, ratings, and timestamps, is generated randomly, and the OrderAmount is generated as a product of price and quantity. In the dishes dataset, attributes like 'DishID', 'DishName',' Cuisine', and 'Price' are created, with cuisines randomly selected and prices uniformly distributed between 5.0 and 100.0. This approach generates a nuanced and realistic dataset for versatile analytical queries.

Database schema:

The database schema consists of four tables:

1. Restaurants Table:
    - 'RestaurantID' (Primary key)
    - 'RestaurantName'
    - 'Cuisine'
    - 'PriceRange'
    - 'Address'
    - 'PhoneNumber'
    - 'Website'

2. Customers Table:
    - CustomerID (Primary Key)
    - FirstName
    - LastName
    - EmailAddress
    - Address
    - PhoneNumber
    - RestaurantID (Foreign Key referencing Restaurants.RestaurantID)
    - ReviewRating
3. Orders Table:
    - OrderID (Primary Key)
    - Quantity
    - MenuItemID
    - MenuCategory
    - MenuItem
    - CustomerID (Foreign Key referencing Customers.CustomerID)

- RestaurantID (Foreign Key referencing Restaurants.RestaurantID)
- CustomerRating
- OrderTimestamp
- OrderAmount

4. Dishes Table:
    - DishID (Primary Key)
    - DishName
    - Cuisine
    - Price

Primary keys in this database are used to uniquely identify each record in a table, facilitating effective data retrieval and preserving integrity, while foreign keys are used to create important linkages between tables.

Justification for separate tables:

The database used, separate tables to store information about the restaurants, customers, dishes, and orders because this is a more normalized and efficient way to store the data. This approach allows for easy querying and updating of the data; it also helps to prevent data redundancy and improve data integrity.

The details about the restaurants are centrally stored in the Restaurants table, and costomert information is captured in the Customers table, which is connected by foreign keys to ensure integrity. To facilitate the investigation of ordering patterns, the Orders table organises transactional data by creating links between consumers, menu items, and restaurants. The Dishes table holds information about dish items, promoting menu consistency.

The use of separate tables for different types of data provides several benefits:

- Data Normalization: The database is organised into separate tables to achieve normalization, reduce data redundancy, and improving data integrity. For example, customer information is stored in the Customers table rather than being duplicated in the Orders table.
- Relationships and Data Integrity: Foreign keys are used to establish relationships between tables. This ensures referential integrity and allows for the retrieval of related information through joins.
- Data Efficiency: Separate tables enhance data efficiency by allowing more focused and targeted data queries. This structure enables the retrieval of specific information without unnecessary data retrieval. Additionally, it supports efficient retrieval and manipulation of specific data subsets, eliminating the need to access the entire database.

Ethical Discussion and Data Privacy:

It is important to think about ethical concerns when creating synthetic data. The use of Faker ensures that the generated data is entirely fictional and doesn't represent real individuals or businesses. Making datasets that replicate real-world situations requires caution, though, as biases or trends may happen accidentally. To prevent any potential misuse or misunderstanding, it is crucial to have transparent documentation and to be upfront about the synthetic nature of the data. By separate tables, which enhances scalability and maintains relational integrity, aligning with ethical principles and ensuring a secure, respectful approach to database development and analysis.

Example Queries:

1. Selection and Aggregation: **Query to find the total amount for each restaurant.**

```
SELECT Restaurants.RestaurantName, SUM(Orders.OrderAmount) AS TotalOrderAmount
FROM Restaurants
JOIN Orders ON Restaurants.RestaurantID = Orders.RestaurantID
GROUP BY Restaurants.RestaurantName;
```

2. Joining table: **Query to retrieve customer information along with the restaurant name for orders with a customer rating of 'Excellent'.**

```
SELECT Customers.CustomerID, Customers.FirstName, Customers.LastName,
Restaurants.RestaurantName
FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID
JOIN Restaurants ON Orders.RestaurantID = Restaurants.RestaurantID
WHERE Customers.ReviewRating = 'Excellent';
```

3. Filtering by Cuisine: **Query to retrieve dish names and prices for Italian cuisine.**

```
SELECT DishName, Price
FROM Dishes
WHERE Cuisine = 'Italian';
```

Python code (pasted code):

Here is the code for generating Restaurant Management System database in SQLite database:

```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov  5 20:37:13 2023

@author: SHAHANA SHIRIN
"""
import numpy as np
import pandas as pd
from faker import Faker
from datetime import datetime, timedelta
import sqlite3
#Initialise faker as fake for creatig fake data
fake = Faker()

# Create SQLite connection and cursor
```

```
conn = sqlite3.connect('restaurant_database.db')
# Enable Foreign Keys
conn.execute("PRAGMA foreign_keys = on;")
cursor = conn.cursor()

# Create Restaurants table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS Restaurants (
        RestaurantID INTEGER PRIMARY KEY,
        RestaurantName TEXT NOT NULL,
        Cuisine TEXT NOT NULL,
        PriceRange TEXT NOT NULL,
        Address TEXT NOT  NULL,
        PhoneNumber TEXT NOT NULL,
        Website TEXT NOT NULL
    )
''')

# Create Customers table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS Customers (
        CustomerID INTEGER PRIMARY KEY,
        FirstName TEXT NOT NULL,
        LastName TEXT NOT NULL,
        EmailAddress TEXT NOT NULL,
        Address TEXT NOT NULL,
        PhoneNumber TEXT NOT NULL,
        RestaurantID INTEGER NOT NULL,
        ReviewRating TEXT NOT NULL,
        FOREIGN KEY (RestaurantID) REFERENCES Restaurants (RestaurantID)
    )
''')

# Create Orders table
cursor.execute('''
```

```
    CREATE TABLE IF NOT EXISTS Orders (
        OrderID INTEGER PRIMARY KEY,
        CustomerID INTEGER NOT NULL,
        MenuItemID INTEGER NOT NULL,
        MenuCategory TEXT NOT NULL,
        MenuItem TEXT NOT NULL,
        Quantity INTEGER NOT NULL,
        OrderTimestamp DATETIME NOT NULL,
        OrderAmount REAL NOT NULL,
        CustomerRating TEXT NOT NULL,
        RestaurantID INTEGER NOT NULL,
        FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID),
        FOREIGN KEY (MenuItemID) REFERENCES Dishes (DishID),
        FOREIGN KEY (RestaurantID) REFERENCES Restaurants (RestaurantID)
    )
''')

# Create Dishes table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS Dishes (
        DishID INTEGER PRIMARY KEY,
        DishName TEXT NOT NULL,
        Cuisine TEXT NOT NULL,
        Price REAL NOT NULL
    )
''')

# Commit changes to save the table structure
conn.commit()

# Set seed for reproducibility
np.random.seed(100)

# Set the number of rows in the dataset
n_restaurant=1000
```

```
# Generate random data for Restaurant Table
restaurant_ids = range(1,n_restaurant+1 )  #nominal data
restaurant_names = [fake.company() for _ in range(n_restaurant)]  #nominal data
cuisines = ['Italian', 'Mexican', 'Chinese', 'Indian', 'American'] #ordinal data
price_ranges = np.random.uniform(5.0, 100.0, n_restaurant) #ratio data
addresses = [fake.address() for _ in range(n_restaurant)] #nominal data
phone_numbers = [fake.numerify('07##-#######')  for _ in range(n_restaurant)] #nominal data
websites = ['https://www.' + fake.company().lower() + '.com' for _ in range(n_restaurant)] #nominal data

# Create DataFrame for restuarent data
restaurant_data = pd.DataFrame({
    'RestaurantID': restaurant_ids,
    'RestaurantName': restaurant_names,
    'Cuisine': np.random.choice(cuisines,n_restaurant),
    'PriceRange': price_ranges,
    'Address': addresses,
    'PhoneNumber': phone_numbers,
    'Website': websites
})

# Generate random data for customer table
customer_ids = range(1, n_restaurant+1)   # Nominal data
first_names = [fake.first_name() for _ in range(n_restaurant)]   # Nominal data
last_names = [fake.last_name() for _ in range(n_restaurant)]   # Nominal data
email_addresses = [fake.email() for _ in range(n_restaurant)]   # Nominal data
addresses = [fake.address() for _ in range(n_restaurant)]  # Nominal data
ratings = ['Poor', 'Average', 'Good', 'Excellent']  # ordinal data
review_ratings = np.random.choice(ratings, 1000, p=[0.1, 0.2, 0.4, 0.3])

# Create DataFrame for custemer
customer_data = pd.DataFrame({
    'CustomerID': customer_ids,
    'FirstName': first_names,
    'LastName': last_names,
```

```
    'EmailAddress': email_addresses,
    'Address': addresses,
    'PhoneNumber': phone_numbers,
    'RestaurantID': restaurant_ids,
    'ReviewRating': review_ratings,
})
# Generate random data for order table
menu_categories = ['Appetizer', 'Main Course', 'Dessert','Light Food']
menu_items = {
    'Appetizer': ['Salad', 'Spring Rolls', 'Bruschetta'],
    'Main Course': ['Pasta', 'Grilled Chicken', 'Steak'],
    'Dessert': ['Chocolate Cake', 'Ice Cream', 'Fruit Tart'],
    'Light Food': ['Vegetable Wrap', 'Quinoa Salad', 'Fruit Smoothie'] }
menu_category_data = np.random.choice(menu_categories, n_restaurant)  # Nominal data
menu_item_data = [np.random.choice(menu_items[category]) for category in menu_category_data]  # Nominal data
quantities_data = np.random.randint(1, 5, n_restaurant)  #ordinal data
customer_rating_data = np.random.choice(ratings, n_restaurant, p=[0.1, 0.2, 0.4, 0.3])  # Ordinal data
start_date = datetime(2023, 1, 1)
end_date = datetime(2023, 12, 1)
order_timestamps = [start_date + timedelta(minutes=np.random.randint(1, 1440)) for _ in range(n_restaurant)]  #Interval data
order_amount = price_ranges*quantities_data  #Ratio data:

# Create DataFrame for orders
order_data = pd.DataFrame({
    'OrderID': range(1, n_restaurant + 1),
    'Quantity': quantities_data,
    'MenuItemID': range(1,n_restaurant + 1),
    'MenuCategory': menu_category_data,
    'MenuItem': menu_item_data,
    'CustomerID': customer_ids,
    'RestaurantID': restaurant_ids,
    'CustomerRating': customer_rating_data,
    'OrderTimestamp': order_timestamps,
```

```
    'OrderAmount': order_amount
})

# Generate random data for dishes tablee
dish_data = pd.DataFrame({
    'DishID': range(1, n_restaurant + 1),  #nominal data
    'DishName':menu_item_data ,  #nominal data
    'Cuisine': np.random.choice(cuisines,n_restaurant),  #ordinal data
    'Price': price_ranges  #ratio data
})

# Print the restaurant data
print("Restaurant Table:")
print(restaurant_data)
# Print the customer Table
print("\nCustomer Table:")
print(customer_data)
# Print the order Table
print("\nOrder Table:")
print(order_data)
#print the dishes Table
print("\nDishes Table:")
print(dish_data)

# Insert data into tables, save this data in sqlite database
restaurant_data.to_sql('Restaurants', conn, index=False, if_exists='replace')
customer_data.to_sql('Customers', conn, index=False, if_exists='replace')
dish_data.to_sql('Dishes', conn, index=False, if_exists='replace')
order_data.to_sql('Orders', conn, index=False, if_exists='replace')

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