Report

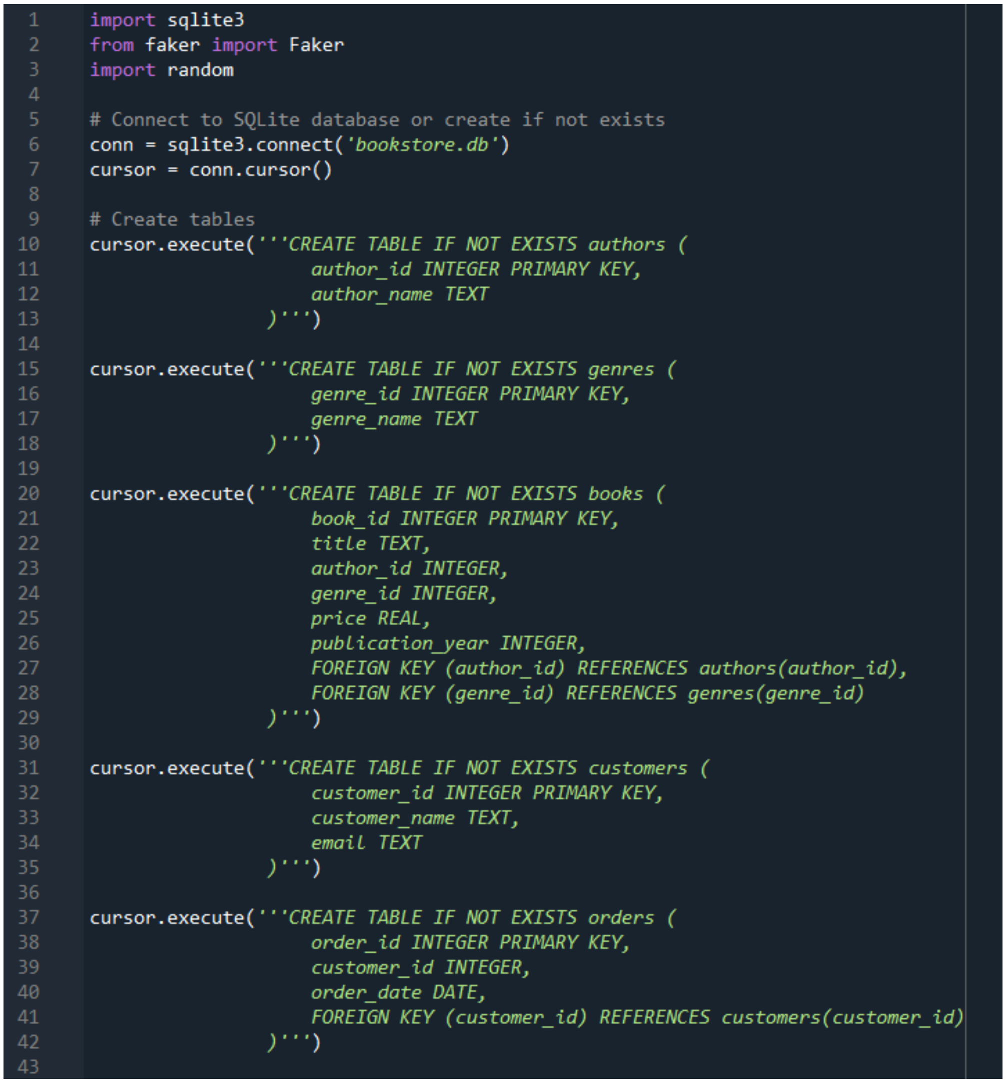
ID: 22071343

GitHub Code Link

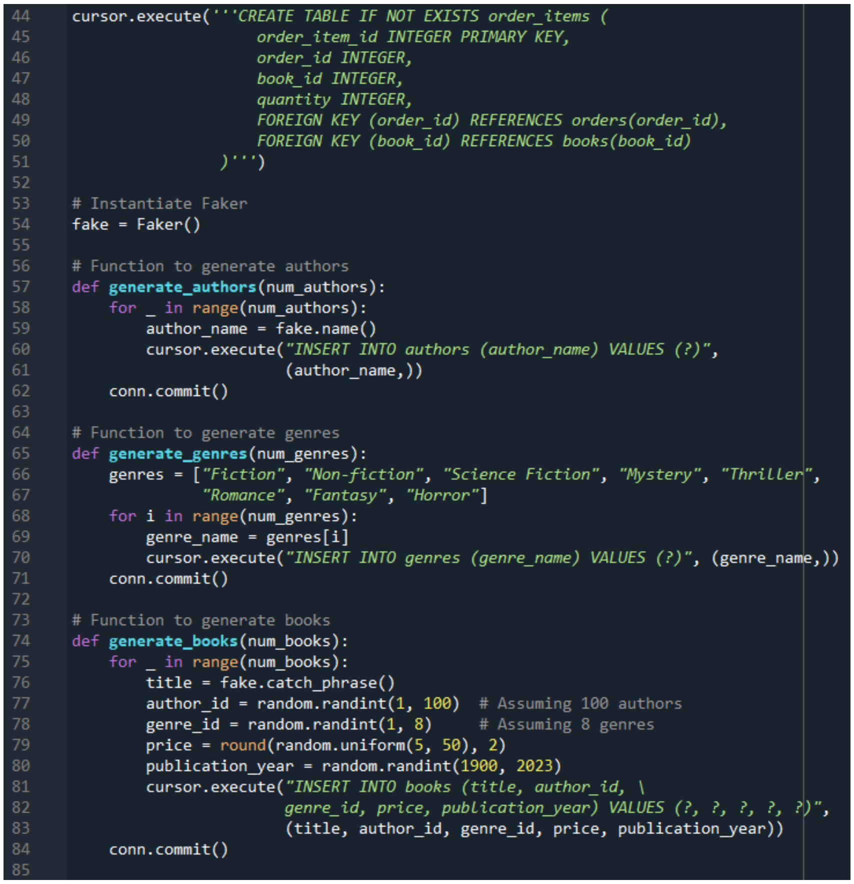
Creating a Fictional Online Bookstore Database

1. Data Generation Process:

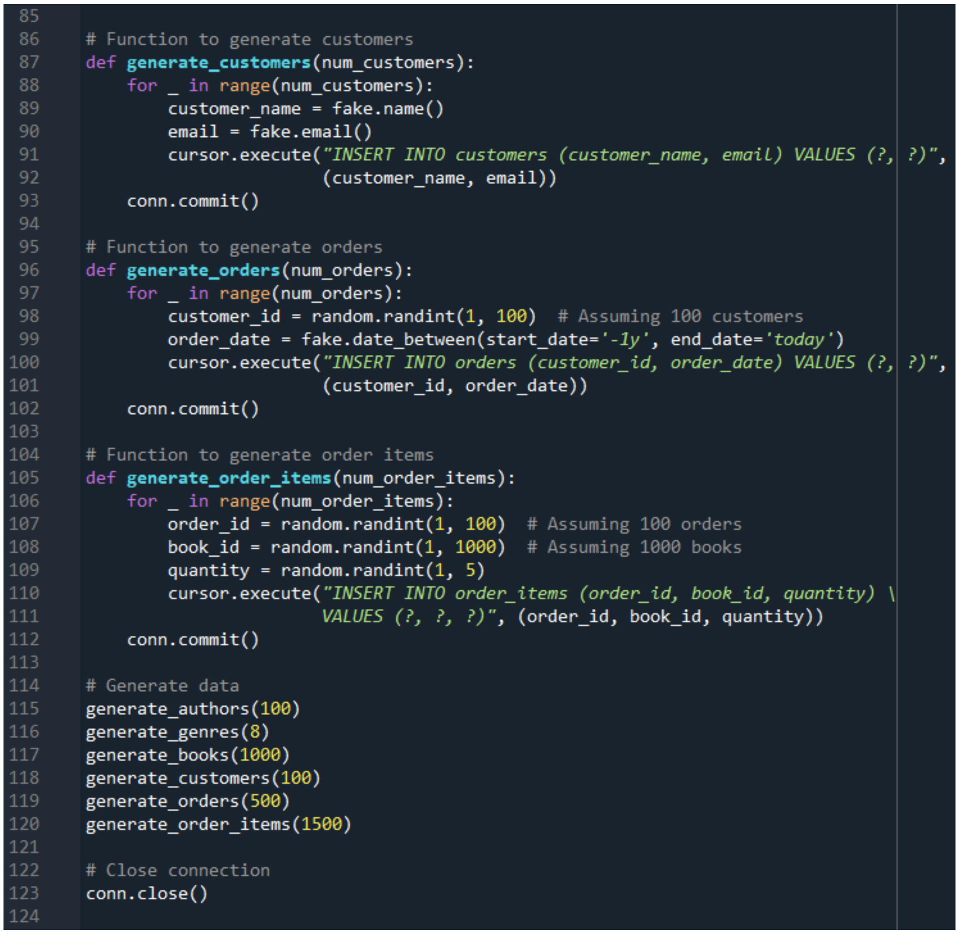
To generate the data for the fictional online bookstore database, Python along with the SQLite3 library and Faker library were utilized. The Faker library allowed for the creation of realistic randomized data. The data generation process involved creating tables for authors, genres, books, customers, orders, and order items. For each table, appropriate randomized data was generated using Faker's various data providers such as name, email, date, catch_phrase, and Others.



```python
import sqlite3
from faker import Faker
import random

# Connect to SQLite database or create if not exists
conn = sqlite3.connect('bookstore.db')
cursor = conn.cursor()

# Create tables
cursor.execute('''CREATE TABLE IF NOT EXISTS authors (
                    author_id INTEGER PRIMARY KEY,
                    author_name TEXT
                )''')

cursor.execute('''CREATE TABLE IF NOT EXISTS genres (
                    genre_id INTEGER PRIMARY KEY,
                    genre_name TEXT
                )''')

cursor.execute('''CREATE TABLE IF NOT EXISTS books (
                    book_id INTEGER PRIMARY KEY,
                    title TEXT,
                    author_id INTEGER,
                    genre_id INTEGER,
                    price REAL,
                    publication_year INTEGER,
                    FOREIGN KEY (author_id) REFERENCES authors(author_id),
                    FOREIGN KEY (genre_id) REFERENCES genres(genre_id)
                )''')

cursor.execute('''CREATE TABLE IF NOT EXISTS customers (
                    customer_id INTEGER PRIMARY KEY,
                    customer_name TEXT,
                    email TEXT
                )''')

cursor.execute('''CREATE TABLE IF NOT EXISTS orders (
                    order_id INTEGER PRIMARY KEY,
                    customer_id INTEGER,
                    order_date DATE,
                    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
                )''')

```



```python
cursor.execute('''CREATE TABLE IF NOT EXISTS order_items (
                    order_item_id INTEGER PRIMARY KEY,
                    order_id INTEGER,
                    book_id INTEGER,
                    quantity INTEGER,
                    FOREIGN KEY (order_id) REFERENCES orders(order_id),
                    FOREIGN KEY (book_id) REFERENCES books(book_id)
                )''')

# Instantiate Faker
fake = Faker()

# Function to generate authors
def generate_authors(num_authors):
    for _ in range(num_authors):
        author_name = fake.name()
        cursor.execute("INSERT INTO authors (author_name) VALUES (?)",
                       (author_name,))
    conn.commit()

# Function to generate genres
def generate_genres(num_genres):
    genres = ["Fiction", "Non-fiction", "Science Fiction", "Mystery", "Thriller",
              "Romance", "Fantasy", "Horror"]
    for i in range(num_genres):
        genre_name = genres[i]
        cursor.execute("INSERT INTO genres (genre_name) VALUES (?)", (genre_name,))
    conn.commit()

# Function to generate books
def generate_books(num_books):
    for _ in range(num_books):
        title = fake.catch_phrase()
        author_id = random.randint(1, 100)  # Assuming 100 authors
        genre_id = random.randint(1, 8)     # Assuming 8 genres
        price = round(random.uniform(5, 50), 2)
        publication_year = random.randint(1900, 2023)
        cursor.execute("INSERT INTO books (title, author_id, \
                       genre_id, price, publication_year) VALUES (?, ?, ?, ?, ?)",
                       (title, author_id, genre_id, price, publication_year))
    conn.commit()

```



```python

# Function to generate customers
def generate_customers(num_customers):
    for _ in range(num_customers):
        customer_name = fake.name()
        email = fake.email()
        cursor.execute("INSERT INTO customers (customer_name, email) VALUES (?, ?)",
                       (customer_name, email))
    conn.commit()

# Function to generate orders
def generate_orders(num_orders):
    for _ in range(num_orders):
        customer_id = random.randint(1, 100)  # Assuming 100 customers
        order_date = fake.date_between(start_date='-1y', end_date='today')
        cursor.execute("INSERT INTO orders (customer_id, order_date) VALUES (?, ?)",
                       (customer_id, order_date))
    conn.commit()

# Function to generate order items
def generate_order_items(num_order_items):
    for _ in range(num_order_items):
        order_id = random.randint(1, 100)  # Assuming 100 orders
        book_id = random.randint(1, 1000)  # Assuming 1000 books
        quantity = random.randint(1, 5)
        cursor.execute("INSERT INTO order_items (order_id, book_id, quantity) \
                       VALUES (?, ?, ?)", (order_id, book_id, quantity))
    conn.commit()

# Generate data
generate_authors(100)
generate_genres(8)
generate_books(1000)
generate_customers(100)
generate_orders(500)
generate_order_items(1500)

# Close connection
conn.close()

```

2. Database Schema:

The database schema comprises six tables:

- **Authors:** Contains information about book authors.
    - Columns: **author_id** (Primary Key), **author_name**.
- **Genres:** Contains different genres of books.
    - Columns: **genre_id** (Primary Key), **genre_name**.
- **Books:** Holds details about individual books.
    - Columns: **book_id** (Primary Key), **title**, **author_id** (Foreign Key), **genre_id** (Foreign Key), **price**, **publication_year**.
- **Customers:** Stores information about bookstore customers.
    - Columns: **customer_id** (Primary Key), **customer_name**, **email**.
- **Orders:** Contains details of orders placed by customers.
    - Columns: **order_id** (Primary Key), **customer_id** (Foreign Key), **order_date**.
- **Order Items:** Stores information about individual items within an order.
    - Columns: **order_item_id** (Primary Key), **order_id** (Foreign Key), **book_id** (Foreign Key), **quantity**.

3. Justification for Separate Tables and Ethical Discussion:

Separating the data into multiple tables enhances data integrity, allows for efficient querying, and follows the principles of database normalization. For instance:

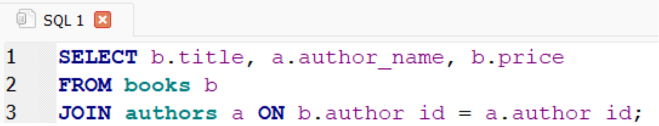
- Authors and genres are kept in separate tables to prevent data redundancy and ensure each author/genre is stored only once.
- The orders and order items are separated to handle one-to-many relationships efficiently, where one order can contain multiple items.

Ethically, it's essential to consider privacy and consent when dealing with customer data. While the data generated by Faker is not real, in a real-world scenario, obtaining consent and ensuring data privacy would be paramount.

4. Example Queries:

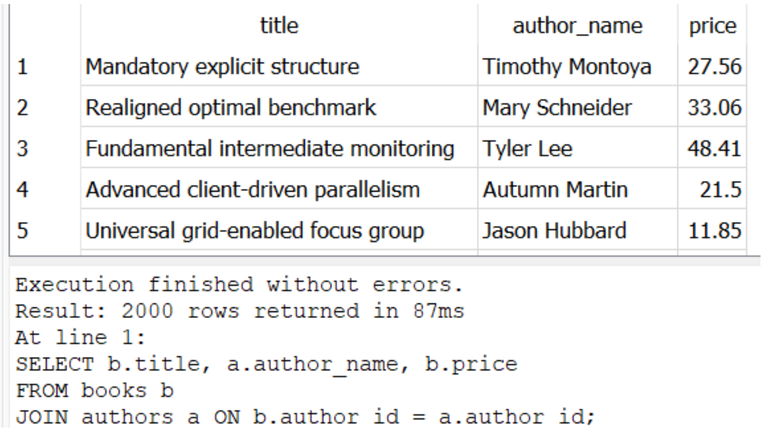
Below are example queries demonstrating different data types and joins:

1. **Selecting all books with their titles, authors, and prices:**



```sql
SELECT b.title, a.author_name, b.price
FROM books b
JOIN authors a ON b.author id = a.author id;
```

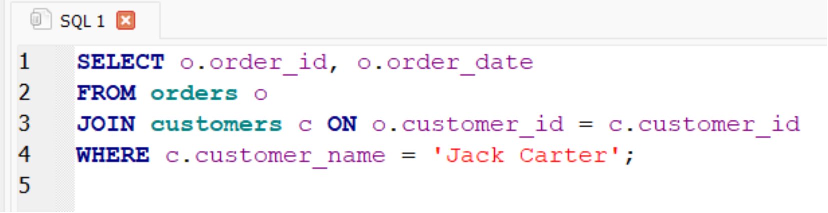
Result:



	title	author_name	price
1	Mandatory explicit structure	Timothy Montoya	27.56
2	Realigned optimal benchmark	Mary Schneider	33.06
3	Fundamental intermediate monitoring	Tyler Lee	48.41
4	Advanced client-driven parallelism	Autumn Martin	21.5
5	Universal grid-enabled focus group	Jason Hubbard	11.85

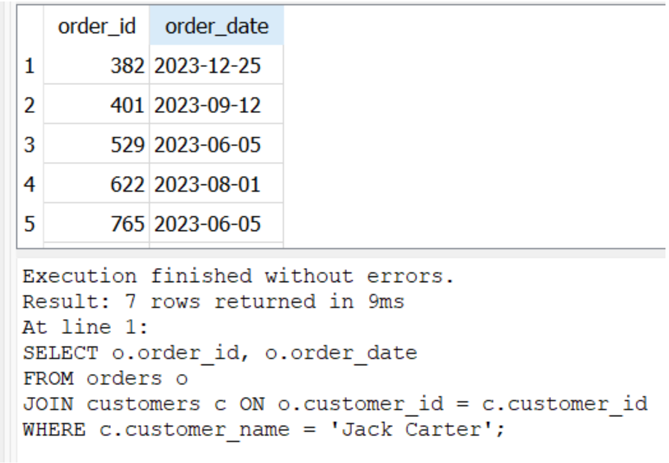
```
Execution finished without errors.
Result: 2000 rows returned in 87ms
At line 1:
SELECT b.title, a.author_name, b.price
FROM books b
JOIN authors a ON b.author id = a.author id;
```

2. **Selecting orders placed by a specific customer:**



```sql
SELECT o.order_id, o.order_date
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
WHERE c.customer_name = 'Jack Carter';

```

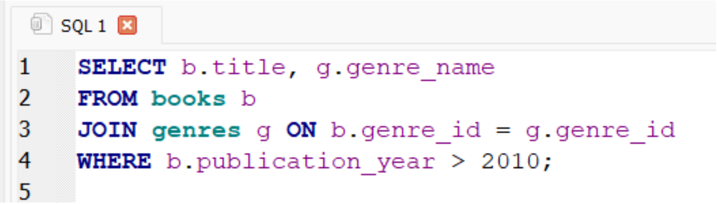
Result:



	order_id	order_date
1	382	2023-12-25
2	401	2023-09-12
3	529	2023-06-05
4	622	2023-08-01
5	765	2023-06-05

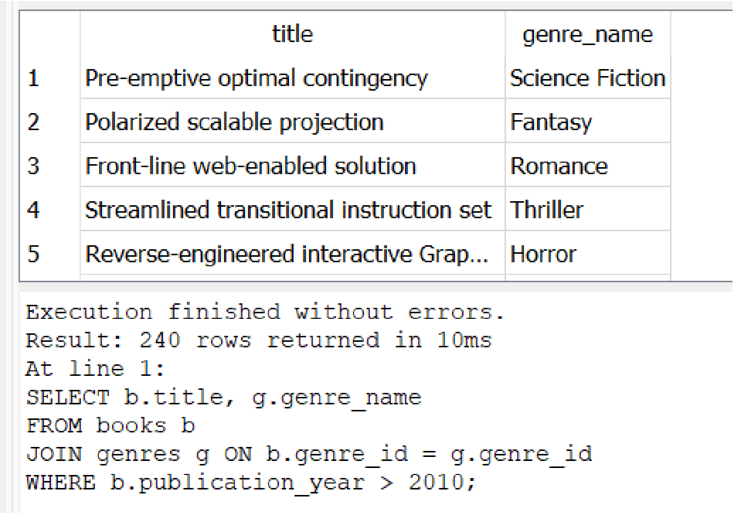
```
Execution finished without errors.
Result: 7 rows returned in 9ms
At line 1:
SELECT o.order_id, o.order_date
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
WHERE c.customer_name = 'Jack Carter';
```

3. Selecting books published after 2010 along with their genres:



```sql
SELECT b.title, g.genre_name
FROM books b
JOIN genres g ON b.genre_id = g.genre_id
WHERE b.publication_year > 2010;
```

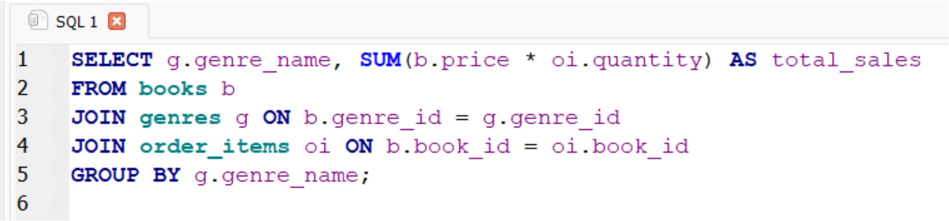
Result:



	title	genre_name
1	Pre-emptive optimal contingency	Science Fiction
2	Polarized scalable projection	Fantasy
3	Front-line web-enabled solution	Romance
4	Streamlined transitional instruction set	Thriller
5	Reverse-engineered interactive Grap...	Horror

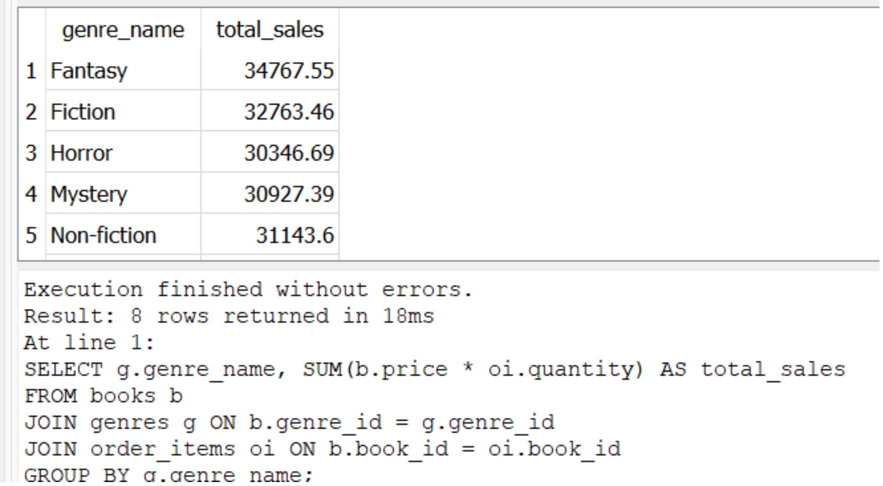
```
Execution finished without errors.
Result: 240 rows returned in 10ms
At line 1:
SELECT b.title, g.genre_name
FROM books b
JOIN genres g ON b.genre_id = g.genre_id
WHERE b.publication_year > 2010;
```

4. Selecting total sales per genre:



```sql
SELECT g.genre_name, SUM(b.price * oi.quantity) AS total_sales
FROM books b
JOIN genres g ON b.genre_id = g.genre_id
JOIN order_items oi ON b.book_id = oi.book_id
GROUP BY g.genre_name;
```

Result:



	genre_name	total_sales
1	Fantasy	34767.55
2	Fiction	32763.46
3	Horror	30346.69
4	Mystery	30927.39
5	Non-fiction	31143.6

```
Execution finished without errors.
Result: 8 rows returned in 18ms
At line 1:
SELECT g.genre_name, SUM(b.price * oi.quantity) AS total_sales
FROM books b
JOIN genres g ON b.genre_id = g.genre_id
JOIN order_items oi ON b.book_id = oi.book_id
GROUP BY g.genre_name;
```

Conclusion:

In conclusion, the creation of the fictional online bookstore database involved thoughtful consideration of data generation, schema design, ethical considerations, and example queries. The database provides a robust foundation for managing and analyzing bookstore data effectively.

Code Repository Link:

https://github.com/muhammadomer1live/22071343_sql_assignment