ASSIGNMENT-16.1

NAME: VITTAM VENKATESH

HALLTICKET NUMBER:2403A52419

BATCH:15

SUBJECT:AI CODING

TASK-1

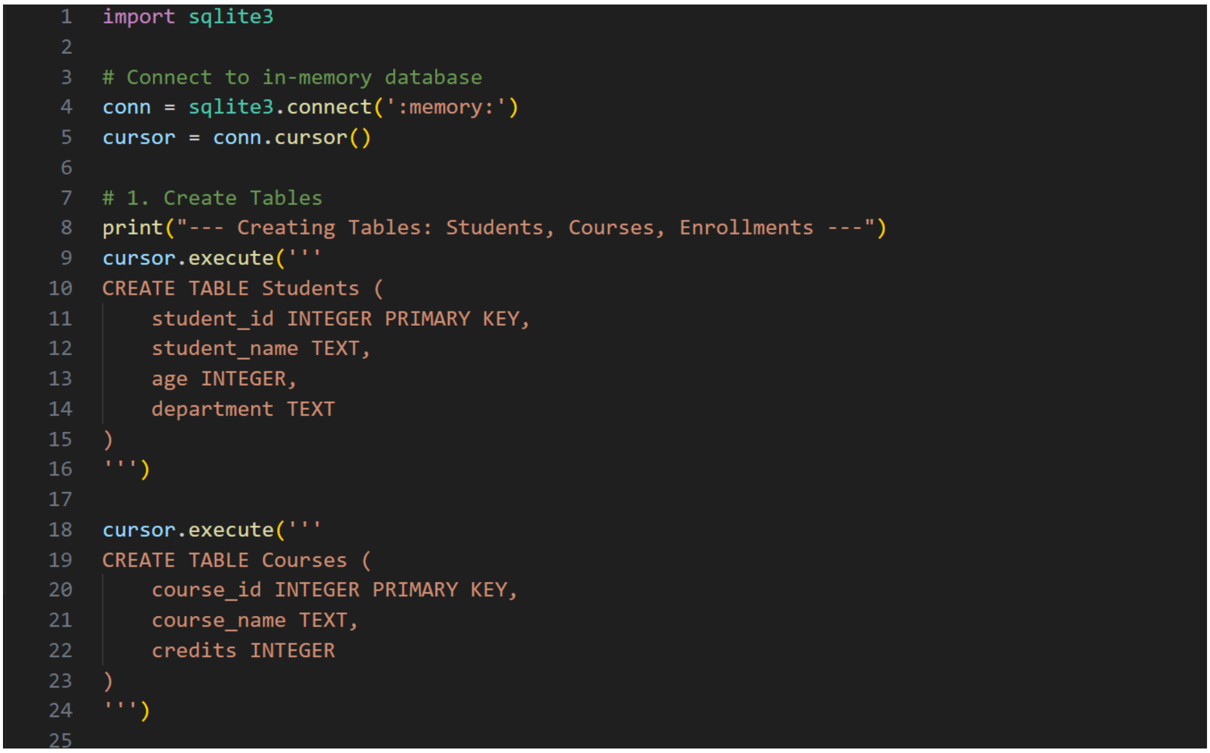
QUESTION:

1 – Student Information System Schema
Task:
Design a database schema for a Student Information System and
generate queries using AI.
Instructions:
• Tables: Students, Courses, Enrollments.
• Define primary keys, foreign keys, and relationships.
• Generate queries:
o Insert a new student record.
o Fetch all courses enrolled by a student.
o Count number of students in each course.
Expected Output:
• SQL CREATE TABLE statements and queries for student–course
relationships

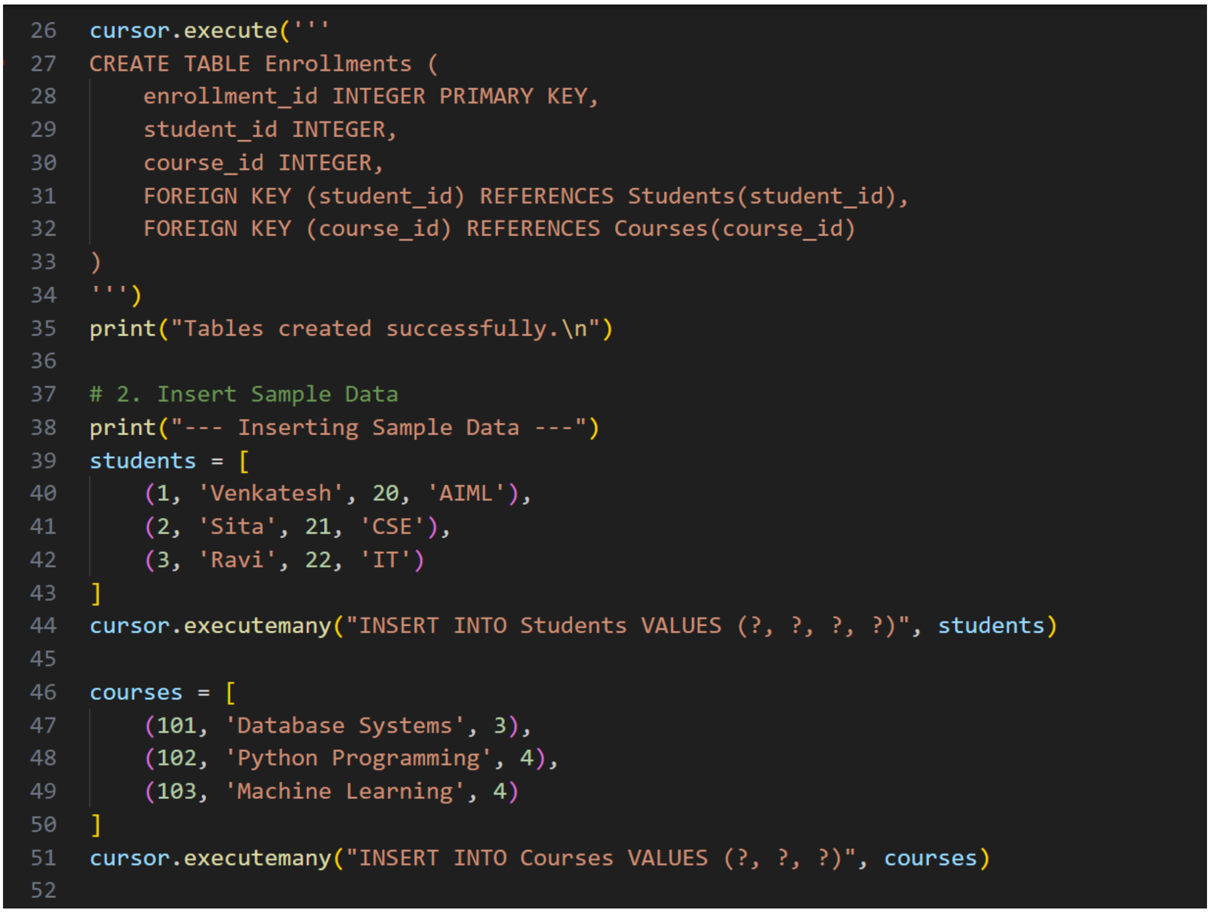
PROMPT:

Use AI to design a database schema for a Student Information System with three tables — Students, Courses, and Enrollment . Define the primary and foreign keys. Generate SQL queries to insert a student, fetch all courses enrolled by a student, and count the number of students in each course

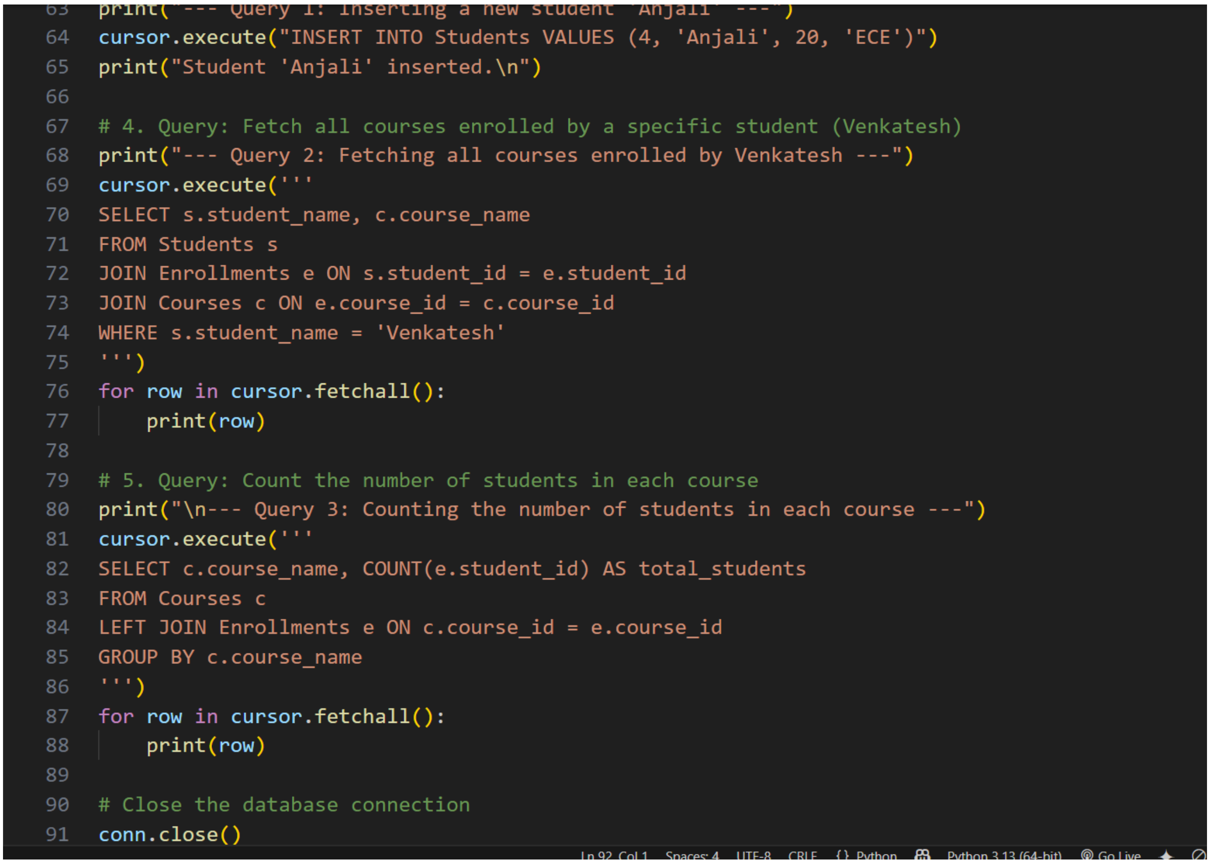
CODE:



```python
import sqlite3

# Connect to in-memory database
conn = sqlite3.connect(':memory:')
cursor = conn.cursor()

# 1. Create Tables
print("--- Creating Tables: Students, Courses, Enrollments ---")
cursor.execute('''
CREATE TABLE Students (
    student_id INTEGER PRIMARY KEY,
    student_name TEXT,
    age INTEGER,
    department TEXT
)
''')

cursor.execute('''
CREATE TABLE Courses (
    course_id INTEGER PRIMARY KEY,
    course_name TEXT,
    credits INTEGER
)
''')

```

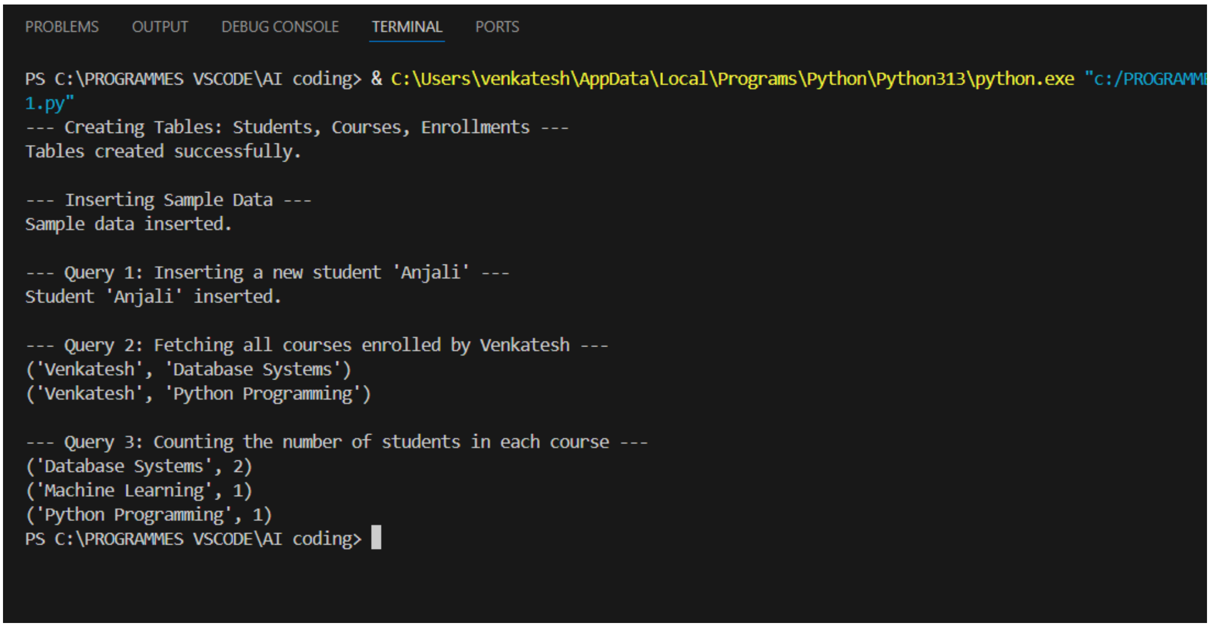


```python
cursor.execute('''
CREATE TABLE Enrollments (
    enrollment_id INTEGER PRIMARY KEY,
    student_id INTEGER,
    course_id INTEGER,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
)
''')
print("Tables created successfully.\n")

# 2. Insert Sample Data
print("--- Inserting Sample Data ---")
students = [
    (1, 'Venkatesh', 20, 'AIML'),
    (2, 'Sita', 21, 'CSE'),
    (3, 'Ravi', 22, 'IT')
]
cursor.executemany("INSERT INTO Students VALUES (?, ?, ?, ?)", students)

courses = [
    (101, 'Database Systems', 3),
    (102, 'Python Programming', 4),
    (103, 'Machine Learning', 4)
]
cursor.executemany("INSERT INTO Courses VALUES (?, ?, ?)", courses)

```



```python
print("--- Query 1: Inserting a new student 'Anjali' ---")
cursor.execute("INSERT INTO Students VALUES (4, 'Anjali', 20, 'ECE')")
print("Student 'Anjali' inserted.\n")

# 4. Query: Fetch all courses enrolled by a specific student (Venkatesh)
print("--- Query 2: Fetching all courses enrolled by Venkatesh ---")
cursor.execute('''
SELECT s.student_name, c.course_name
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
JOIN Courses c ON e.course_id = c.course_id
WHERE s.student_name = 'Venkatesh'
''')
for row in cursor.fetchall():
    print(row)

# 5. Query: Count the number of students in each course
print("\n--- Query 3: Counting the number of students in each course ---")
cursor.execute('''
SELECT c.course_name, COUNT(e.student_id) AS total_students
FROM Courses c
LEFT JOIN Enrollments e ON c.course_id = e.course_id
GROUP BY c.course_name
''')
for row in cursor.fetchall():
    print(row)

# Close the database connection
conn.close()
```

OUTPUT:



```
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGRAMME
1.py"
--- Creating Tables: Students, Courses, Enrollments ---
Tables created successfully.

--- Inserting Sample Data ---
Sample data inserted.

--- Query 1: Inserting a new student 'Anjali' ---
Student 'Anjali' inserted.

--- Query 2: Fetching all courses enrolled by Venkatesh ---
('Venkatesh', 'Database Systems')
('Venkatesh', 'Python Programming')

--- Query 3: Counting the number of students in each course ---
('Database Systems', 2)
('Machine Learning', 1)
('Python Programming', 1)
PS C:\PROGRAMMES VSCODE\AI coding> 
```

OBSERVATION:

AI successfully generated the database schema with relationships between the tables using primary and foreign keys. It also provided correct SQL queries for inserting a student record, retrieving courses enrolled by a student, and counting students in each course. The output was accurate and easy to understand.

TASK-2

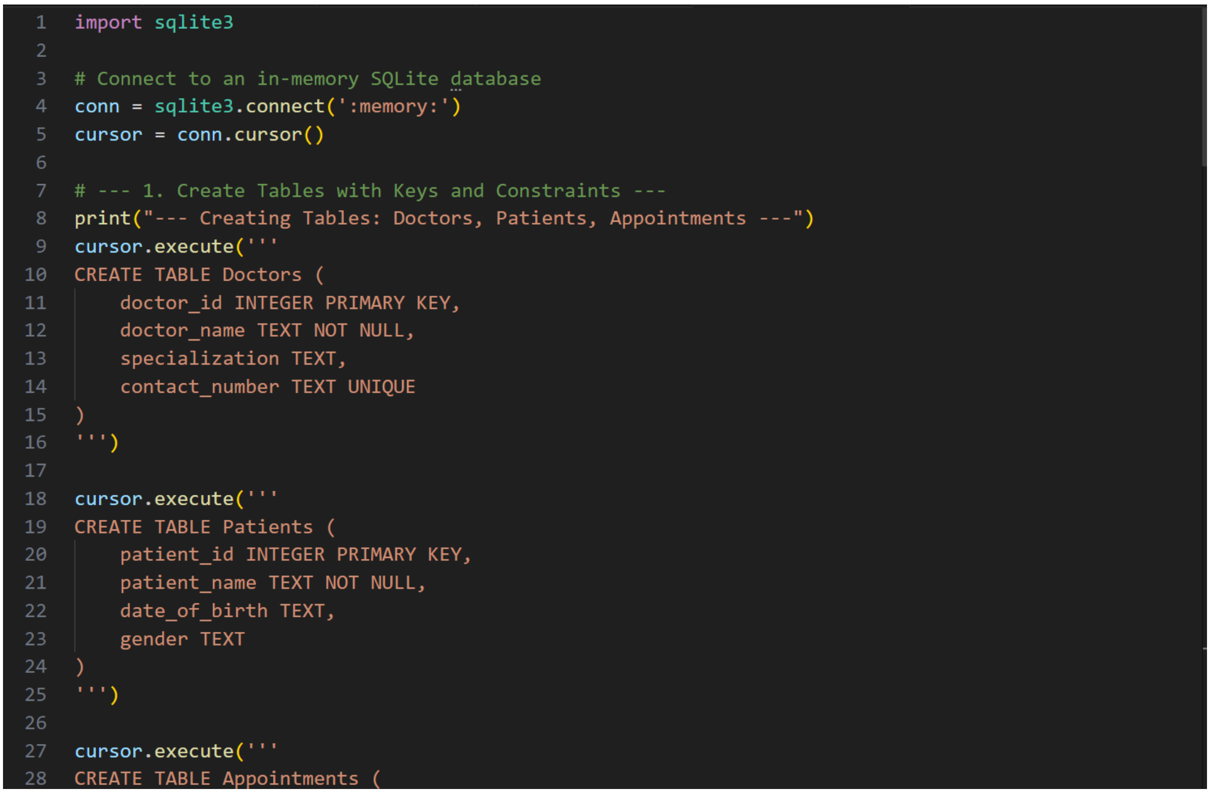
QUESTION:

2 – Hospital Management Database
Task:
Create schema and queries for a Hospital Management System.
Instructions:
• Tables: Doctors, Patients, Appointments.
• Use AI to define constraints (unique IDs, valid dates).
• Generate queries:
o List all appointments for a specific doctor.
o Retrieve patient history by patient ID.
o Count total patients treated by each doctor.
Expected Output:
• Normalized schema and SQL queries with joins

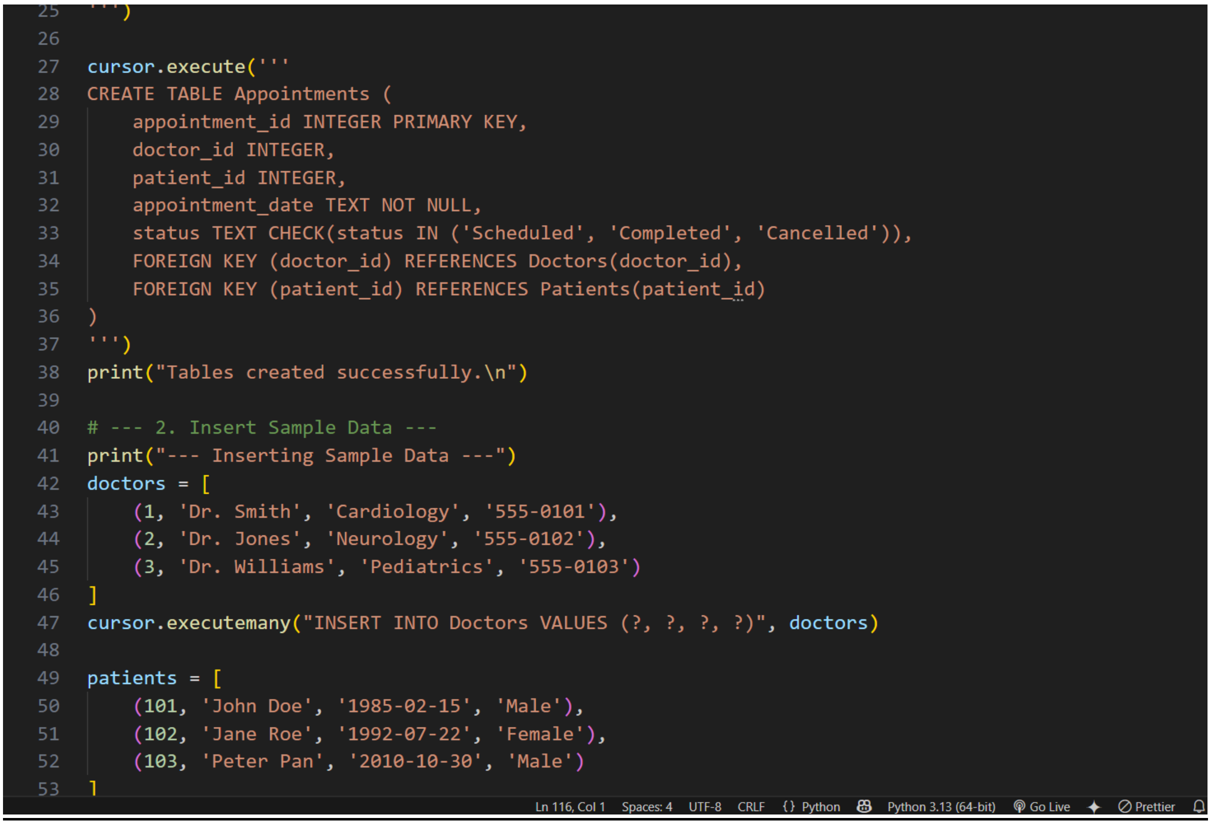
PROMPT:

Use AI to create a Hospital Management System database with three tables — Doctors, Patients, and Appointments. Define primary keys, foreign keys, and constraints for valid data. Generate SQL queries to list all appointments for a doctor, get patient history by ID, and count total patients treated by each doctor.

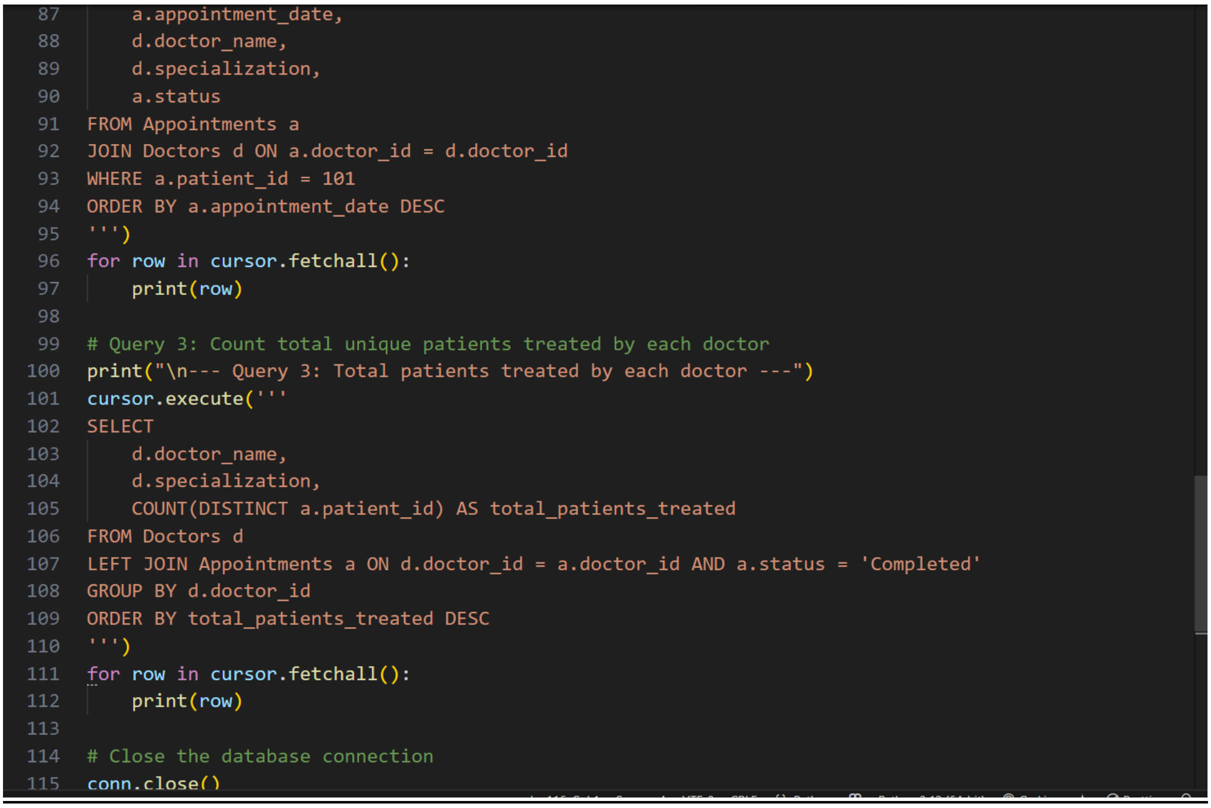
CODE:



```python
import sqlite3

# Connect to an in-memory SQLite database
conn = sqlite3.connect(':memory:')
cursor = conn.cursor()

# --- 1. Create Tables with Keys and Constraints ---
print("--- Creating Tables: Doctors, Patients, Appointments ---")
cursor.execute('''
CREATE TABLE Doctors (
    doctor_id INTEGER PRIMARY KEY,
    doctor_name TEXT NOT NULL,
    specialization TEXT,
    contact_number TEXT UNIQUE
)
''')

cursor.execute('''
CREATE TABLE Patients (
    patient_id INTEGER PRIMARY KEY,
    patient_name TEXT NOT NULL,
    date_of_birth TEXT,
    gender TEXT
)
''')

cursor.execute('''
CREATE TABLE Appointments (
```

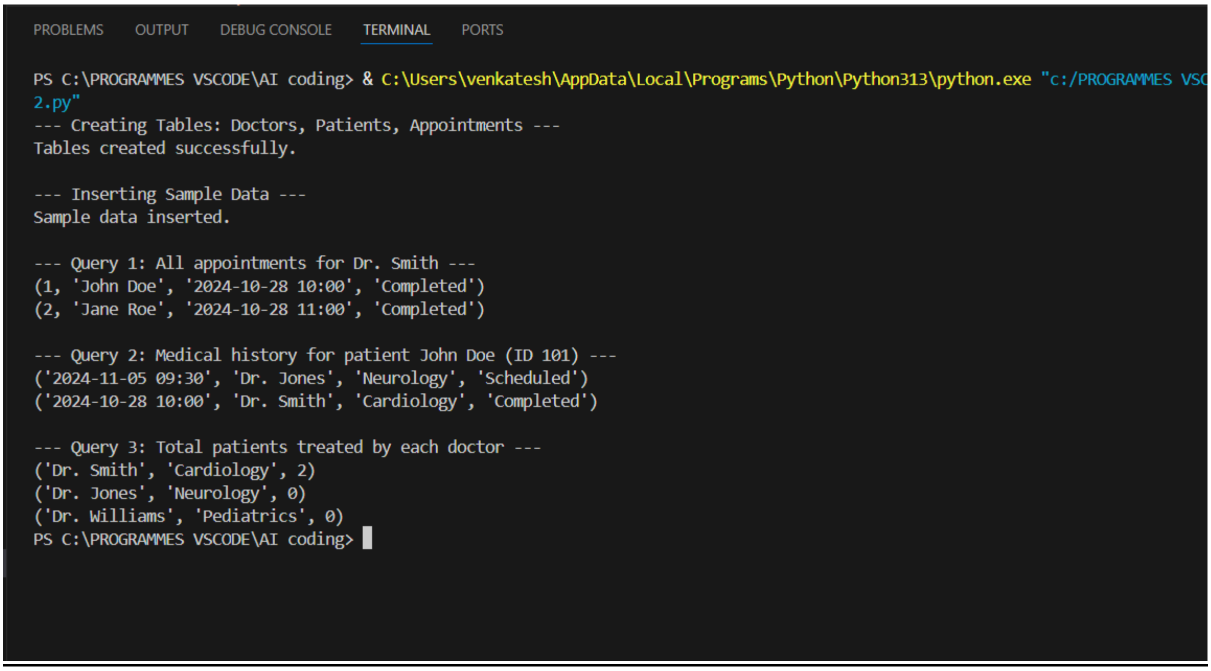


```
''')

cursor.execute('''
CREATE TABLE Appointments (
    appointment_id INTEGER PRIMARY KEY,
    doctor_id INTEGER,
    patient_id INTEGER,
    appointment_date TEXT NOT NULL,
    status TEXT CHECK(status IN ('Scheduled', 'Completed', 'Cancelled')),
    FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id),
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id)
)
''')
print("Tables created successfully.\n")

# --- 2. Insert Sample Data ---
print("--- Inserting Sample Data ---")
doctors = [
    (1, 'Dr. Smith', 'Cardiology', '555-0101'),
    (2, 'Dr. Jones', 'Neurology', '555-0102'),
    (3, 'Dr. Williams', 'Pediatrics', '555-0103')
]
cursor.executemany("INSERT INTO Doctors VALUES (?, ?, ?, ?)", doctors)

patients = [
    (101, 'John Doe', '1985-02-15', 'Male'),
    (102, 'Jane Roe', '1992-07-22', 'Female'),
    (103, 'Peter Pan', '2010-10-30', 'Male')
]
```



```
    a.appointment_date,
    d.doctor_name,
    d.specialization,
    a.status
FROM Appointments a
JOIN Doctors d ON a.doctor_id = d.doctor_id
WHERE a.patient_id = 101
ORDER BY a.appointment_date DESC
''')
for row in cursor.fetchall():
    print(row)

# Query 3: Count total unique patients treated by each doctor
print("\n--- Query 3: Total patients treated by each doctor ---")
cursor.execute('''
SELECT
    d.doctor_name,
    d.specialization,
    COUNT(DISTINCT a.patient_id) AS total_patients_treated
FROM Doctors d
LEFT JOIN Appointments a ON d.doctor_id = a.doctor_id AND a.status = 'Completed'
GROUP BY d.doctor_id
ORDER BY total_patients_treated DESC
''')
for row in cursor.fetchall():
    print(row)

# Close the database connection
conn.close()
```

OUTPUT:



```
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGRAMMES VSC
2.py"
--- Creating Tables: Doctors, Patients, Appointments ---
Tables created successfully.

--- Inserting Sample Data ---
Sample data inserted.

--- Query 1: All appointments for Dr. Smith ---
(1, 'John Doe', '2024-10-28 10:00', 'Completed')
(2, 'Jane Roe', '2024-10-28 11:00', 'Completed')

--- Query 2: Medical history for patient John Doe (ID 101) ---
('2024-11-05 09:30', 'Dr. Jones', 'Neurology', 'Scheduled')
('2024-10-28 10:00', 'Dr. Smith', 'Cardiology', 'Completed')

--- Query 3: Total patients treated by each doctor ---
('Dr. Smith', 'Cardiology', 2)
('Dr. Jones', 'Neurology', 0)
('Dr. Williams', 'Pediatrics', 0)
PS C:\PROGRAMMES VSCODE\AI coding> █
```

OBSERVATION:

AI generated a well-structured database schema with relationships between doctors, patients, and appointments. It used primary and foreign keys correctly and included constraints for data validity. The queries for listing appointments, viewing patient history, and counting patients per doctor were accurate and easy to understand.

TASK-3

QUESTION:

3 – Library Database
Task:
Design schema for a Library Management System.
Instructions:

• Tables: Books, Members, Loans.

• Use AI to suggest indexing strategy for faster queries.

• Generate queries:

o Retrieve all books currently issued.

o Find overdue books (loan date > 30 days).

o Count number of books loaned by each member.

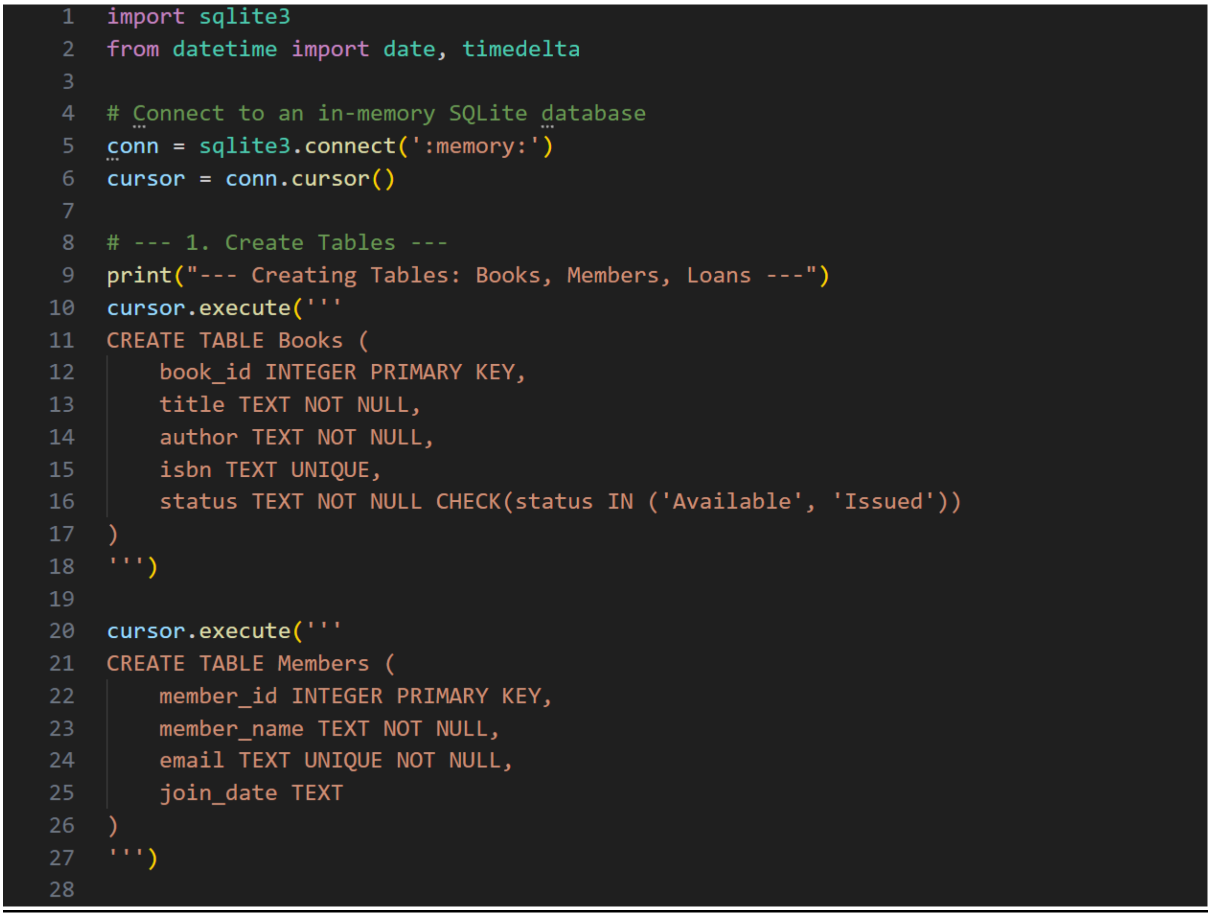
Expected Output:

• Schema + SQL queries demonstrating joins and conditions

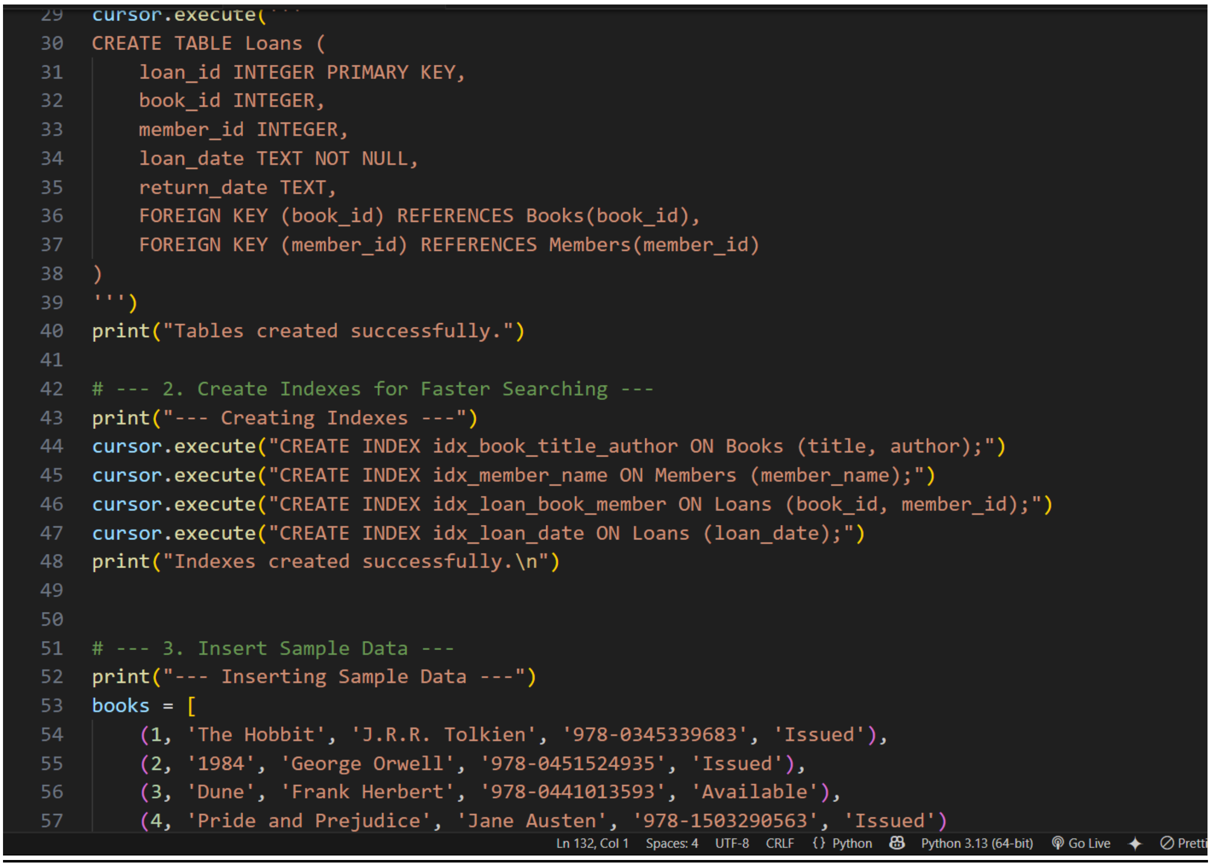
PROMPT:

Use AI to design a Library Management System database with three tables Books, Members, and Loans. Define primary keys, foreign keys, and suggest indexes for faster searching. Generate SQL queries to get all books currently issued, find overdue books loaned more than 30 days ago), and count how many books each member has borrowed.

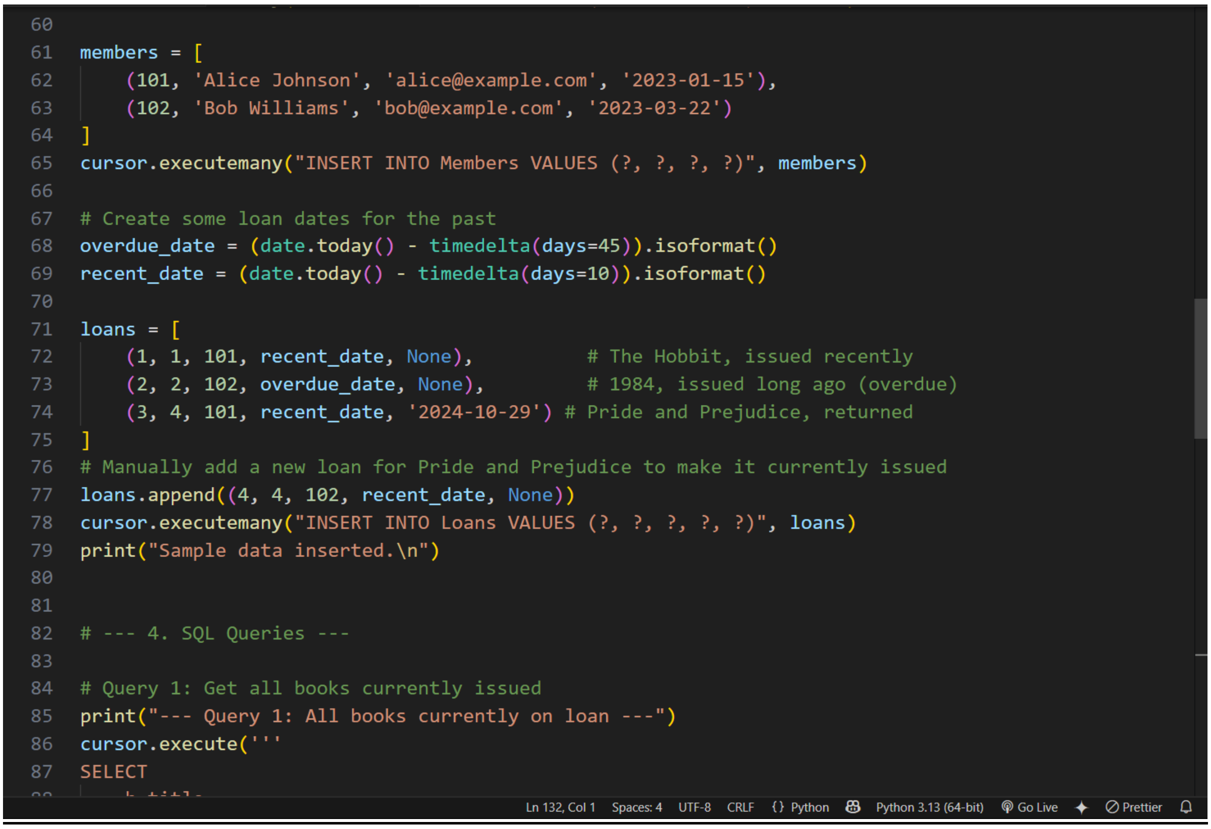
CODE:



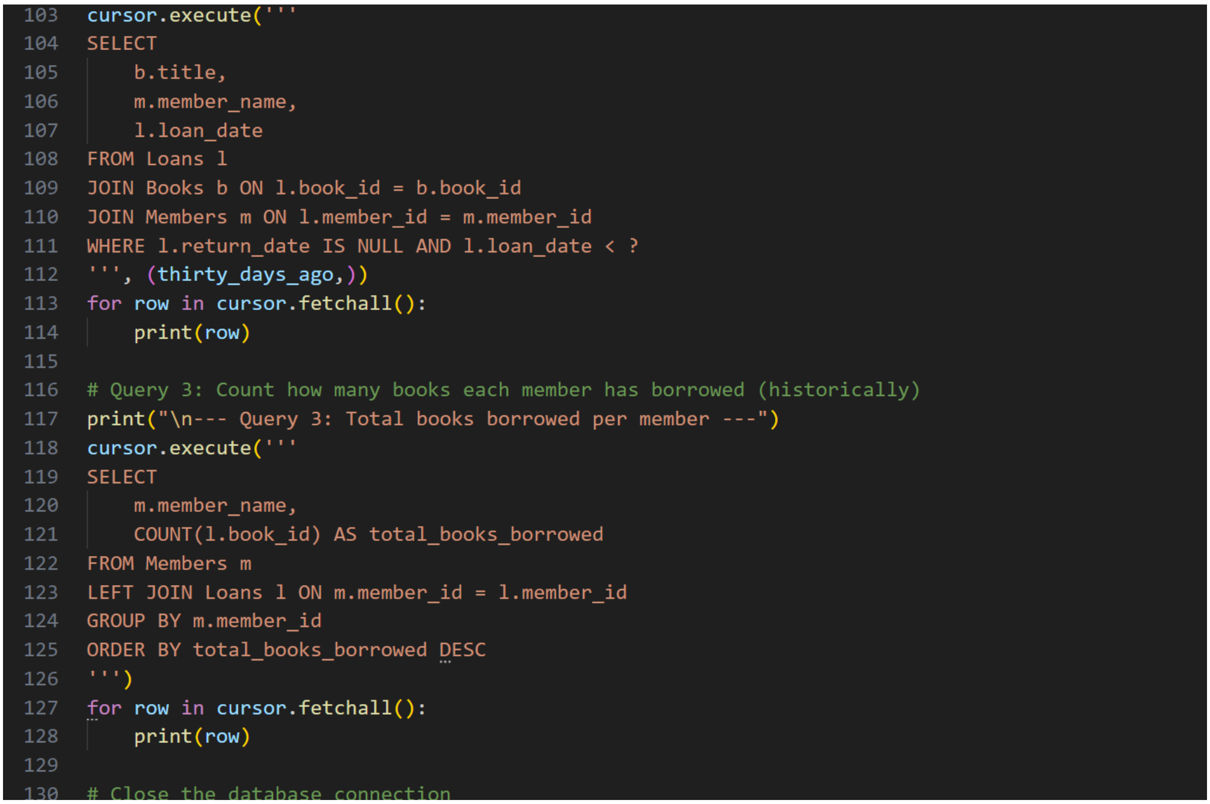
```
import sqlite3
from datetime import date, timedelta

# Connect to an in-memory SQLite database
conn = sqlite3.connect(':memory:')
cursor = conn.cursor()

# --- 1. Create Tables ---
print("--- Creating Tables: Books, Members, Loans ---")
cursor.execute('''
CREATE TABLE Books (
    book_id INTEGER PRIMARY KEY,
    title TEXT NOT NULL,
    author TEXT NOT NULL,
    isbn TEXT UNIQUE,
    status TEXT NOT NULL CHECK(status IN ('Available', 'Issued'))
)
''')

cursor.execute('''
CREATE TABLE Members (
    member_id INTEGER PRIMARY KEY,
    member_name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    join_date TEXT
)
''')

```



```
cursor.execute('''
CREATE TABLE Loans (
    loan_id INTEGER PRIMARY KEY,
    book_id INTEGER,
    member_id INTEGER,
    loan_date TEXT NOT NULL,
    return_date TEXT,
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
    FOREIGN KEY (member_id) REFERENCES Members(member_id)
)
''')
print("Tables created successfully.")

# --- 2. Create Indexes for Faster Searching ---
print("--- Creating Indexes ---")
cursor.execute("CREATE INDEX idx_book_title_author ON Books (title, author);")
cursor.execute("CREATE INDEX idx_member_name ON Members (member_name);")
cursor.execute("CREATE INDEX idx_loan_book_member ON Loans (book_id, member_id);")
cursor.execute("CREATE INDEX idx_loan_date ON Loans (loan_date);")
print("Indexes created successfully.\n")


# --- 3. Insert Sample Data ---
print("--- Inserting Sample Data ---")
books = [
    (1, 'The Hobbit', 'J.R.R. Tolkien', '978-0345339683', 'Issued'),
    (2, '1984', 'George Orwell', '978-0451524935', 'Issued'),
    (3, 'Dune', 'Frank Herbert', '978-0441013593', 'Available'),
    (4, 'Pride and Prejudice', 'Jane Austen', '978-1503290563', 'Issued')
```

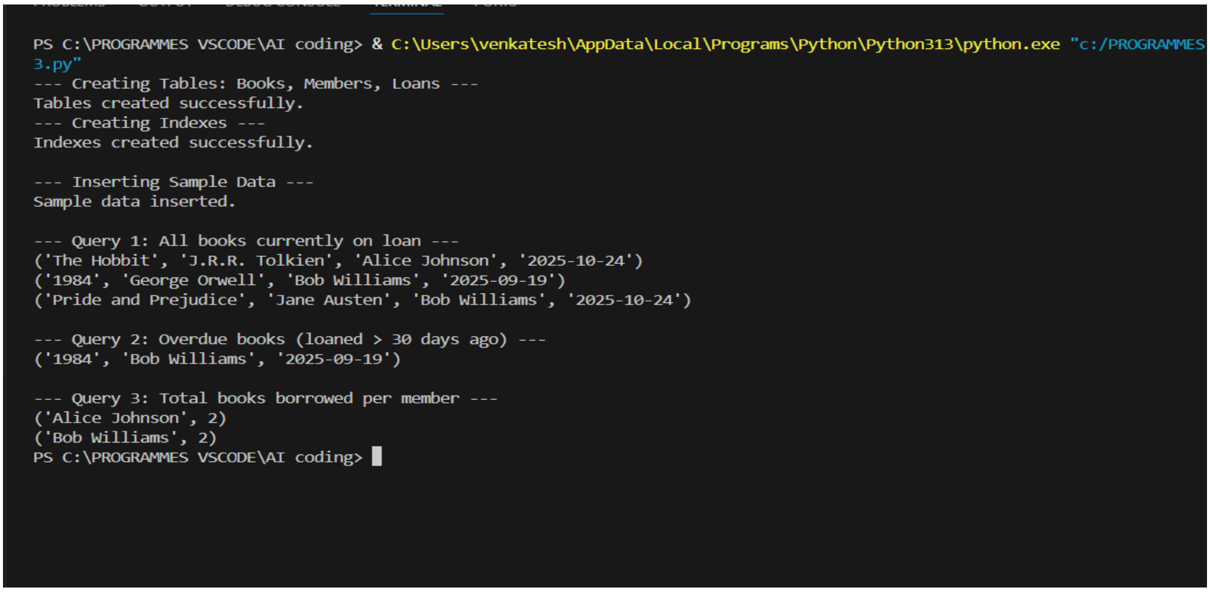


```

members = [
    (101, 'Alice Johnson', 'alice@example.com', '2023-01-15'),
    (102, 'Bob Williams', 'bob@example.com', '2023-03-22')
]
cursor.executemany("INSERT INTO Members VALUES (?, ?, ?, ?)", members)

# Create some loan dates for the past
overdue_date = (date.today() - timedelta(days=45)).isoformat()
recent_date = (date.today() - timedelta(days=10)).isoformat()

loans = [
    (1, 1, 101, recent_date, None),         # The Hobbit, issued recently
    (2, 2, 102, overdue_date, None),        # 1984, issued long ago (overdue)
    (3, 4, 101, recent_date, '2024-10-29') # Pride and Prejudice, returned
]
# Manually add a new loan for Pride and Prejudice to make it currently issued
loans.append((4, 4, 102, recent_date, None))
cursor.executemany("INSERT INTO Loans VALUES (?, ?, ?, ?, ?)", loans)
print("Sample data inserted.\n")


# --- 4. SQL Queries ---

# Query 1: Get all books currently issued
print("--- Query 1: All books currently on loan ---")
cursor.execute('''
SELECT
    b.title,
```



```
cursor.execute('''
SELECT
    b.title,
    m.member_name,
    l.loan_date
FROM Loans l
JOIN Books b ON l.book_id = b.book_id
JOIN Members m ON l.member_id = m.member_id
WHERE l.return_date IS NULL AND l.loan_date < ?
''', (thirty_days_ago,))
for row in cursor.fetchall():
    print(row)

# Query 3: Count how many books each member has borrowed (historically)
print("\n--- Query 3: Total books borrowed per member ---")
cursor.execute('''
SELECT
    m.member_name,
    COUNT(l.book_id) AS total_books_borrowed
FROM Members m
LEFT JOIN Loans l ON m.member_id = l.member_id
GROUP BY m.member_id
ORDER BY total_books_borrowed DESC
''')
for row in cursor.fetchall():
    print(row)

# Close the database connection
```

OUTPUT:



```
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGRAMMES
3.py"
--- Creating Tables: Books, Members, Loans ---
Tables created successfully.
--- Creating Indexes ---
Indexes created successfully.

--- Inserting Sample Data ---
Sample data inserted.

--- Query 1: All books currently on loan ---
('The Hobbit', 'J.R.R. Tolkien', 'Alice Johnson', '2025-10-24')
('1984', 'George Orwell', 'Bob Williams', '2025-09-19')
('Pride and Prejudice', 'Jane Austen', 'Bob Williams', '2025-10-24')

--- Query 2: Overdue books (loaned > 30 days ago) ---
('1984', 'Bob Williams', '2025-09-19')

--- Query 3: Total books borrowed per member ---
('Alice Johnson', 2)
('Bob Williams', 2)
PS C:\PROGRAMMES VSCODE\AI coding> █
```

OBSERVATION:

AI created a normalized database schema with proper relationships between books, members, and loans. It suggested using indexes on frequently searched columns like member_id and book_id to improve performance. The generated SQL queries for issued books, overdue books, and total loans per member were correct, efficient, and easy to execute.

TASK-4

QUESTION:

4 – Real-Time Application: Online Shopping Database
Scenario:
Design a database for an E-commerce platform.
Requirements:

- Tables: Users, Products, Orders, OrderDetails.
- Generate queries to:

o Retrieve all orders by a user.
o Find the most purchased product.
o Calculate total revenue in a given month.

- AI should also suggest normalization improvements and query optimization.

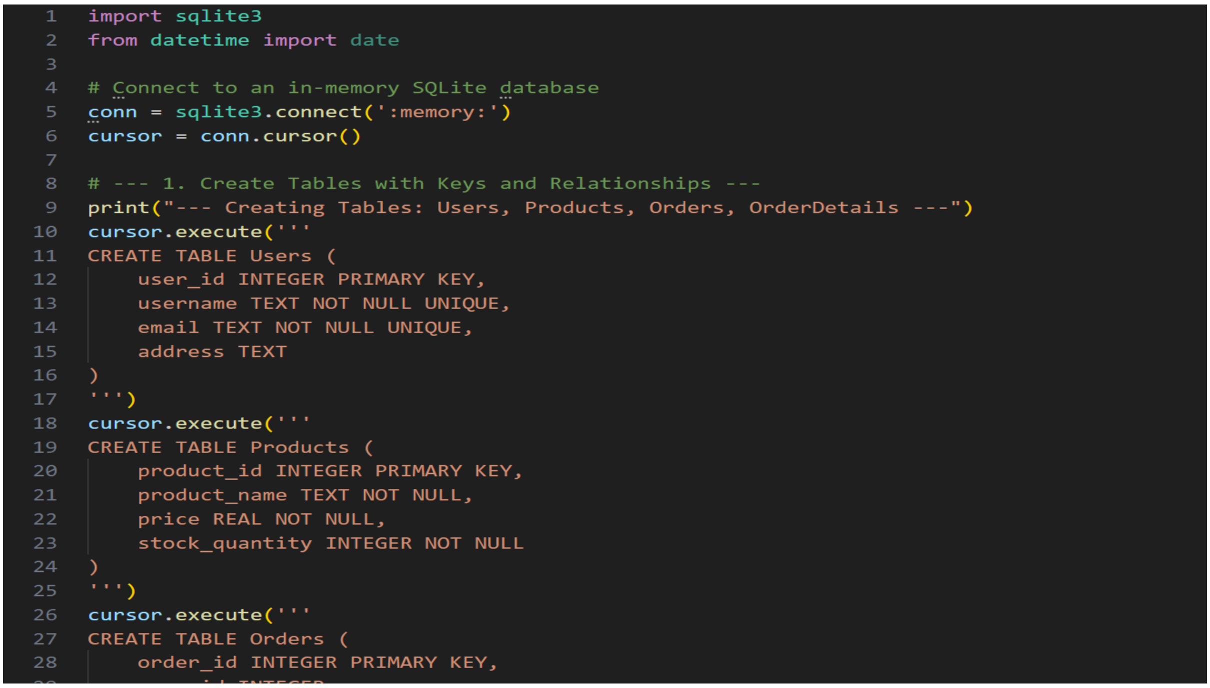
Expected Output:

- Complete schema with relationships + SQL queries for analytics

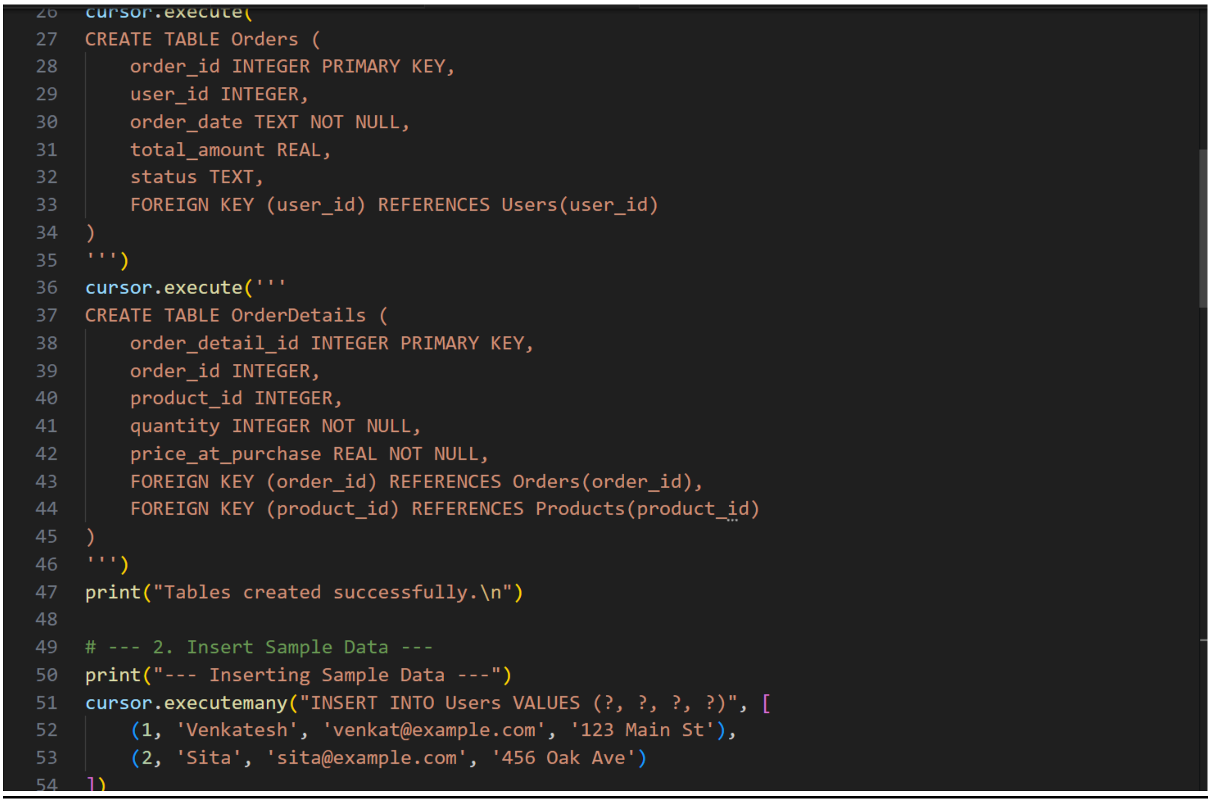
PROMPT:

Use AI to design an Online Shopping Database with four tables Users, Products, Orders, and OrderDetails. Define primary and foreign keys with proper relationships. Generate SQL queries to fetch all orders by a user, find the most purchased product, and calculate total revenue for a given month. Also, suggest normalization and query optimization techniques

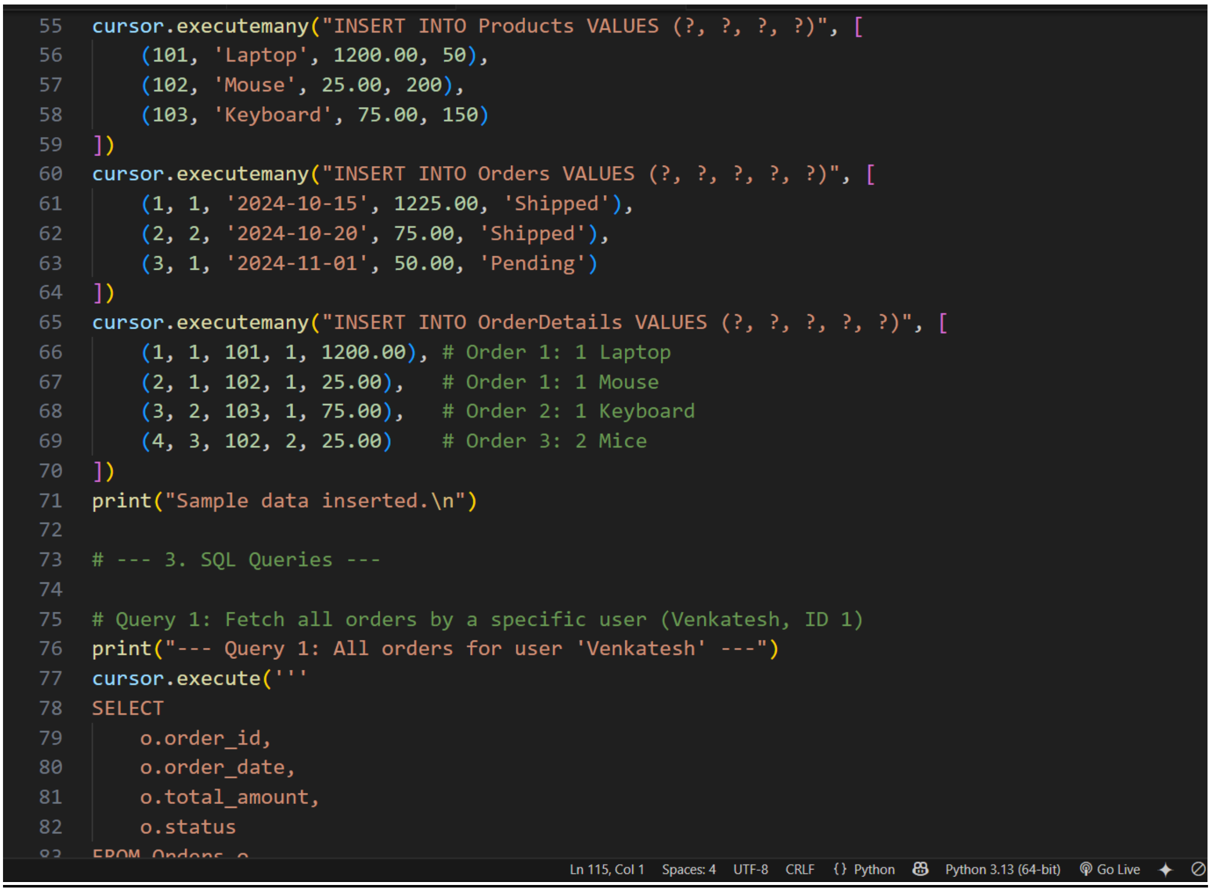
CODE:



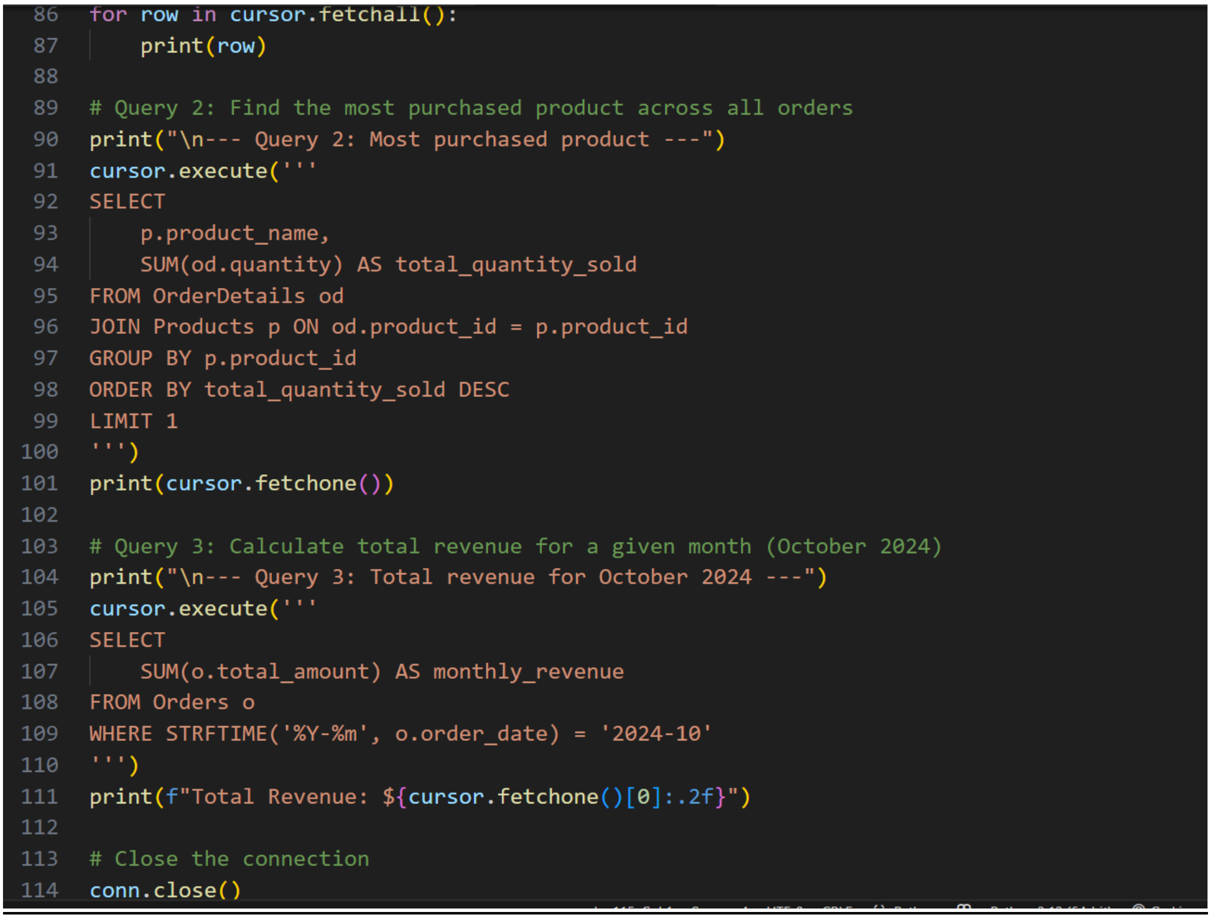
```
import sqlite3
from datetime import date

# Connect to an in-memory SQLite database
conn = sqlite3.connect(':memory:')
cursor = conn.cursor()

# --- 1. Create Tables with Keys and Relationships ---
print("--- Creating Tables: Users, Products, Orders, OrderDetails ---")
cursor.execute('''
CREATE TABLE Users (
    user_id INTEGER PRIMARY KEY,
    username TEXT NOT NULL UNIQUE,
    email TEXT NOT NULL UNIQUE,
    address TEXT
)
''')
cursor.execute('''
CREATE TABLE Products (
    product_id INTEGER PRIMARY KEY,
    product_name TEXT NOT NULL,
    price REAL NOT NULL,
    stock_quantity INTEGER NOT NULL
)
''')
cursor.execute('''
CREATE TABLE Orders (
    order_id INTEGER PRIMARY KEY,
    user_id INTEGER
```



```
cursor.execute(
CREATE TABLE Orders (
    order_id INTEGER PRIMARY KEY,
    user_id INTEGER,
    order_date TEXT NOT NULL,
    total_amount REAL,
    status TEXT,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
)
''')
cursor.execute('''
CREATE TABLE OrderDetails (
    order_detail_id INTEGER PRIMARY KEY,
    order_id INTEGER,
    product_id INTEGER,
    quantity INTEGER NOT NULL,
    price_at_purchase REAL NOT NULL,
    FOREIGN KEY (order_id) REFERENCES Orders(order_id),
    FOREIGN KEY (product_id) REFERENCES Products(product_id)
)
''')
print("Tables created successfully.\n")

# --- 2. Insert Sample Data ---
print("--- Inserting Sample Data ---")
cursor.executemany("INSERT INTO Users VALUES (?, ?, ?, ?)", [
    (1, 'Venkatesh', 'venkat@example.com', '123 Main St'),
    (2, 'Sita', 'sita@example.com', '456 Oak Ave')
])
```

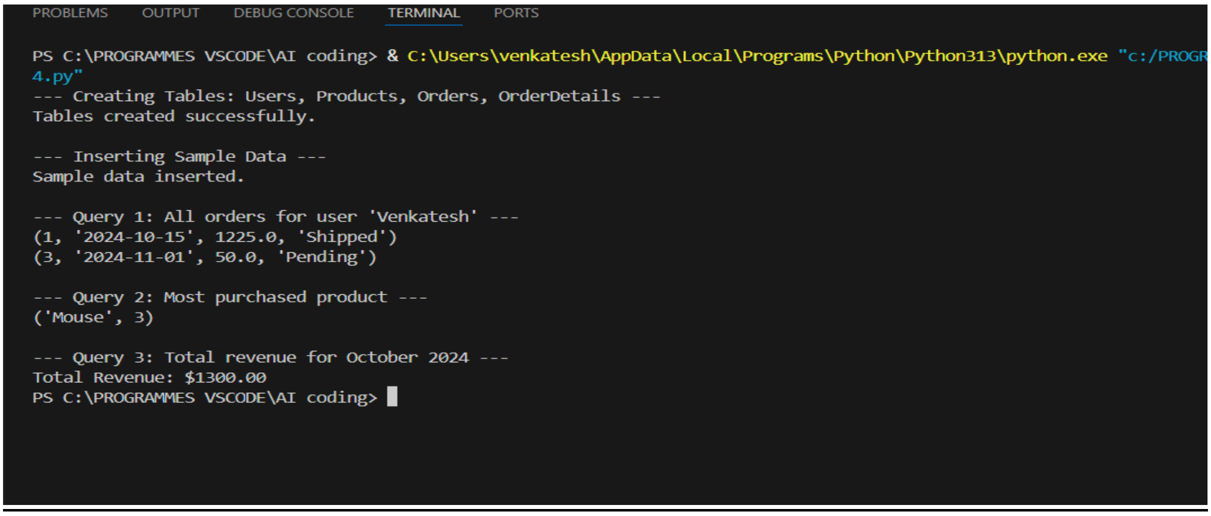


```
cursor.executemany("INSERT INTO Products VALUES (?, ?, ?, ?)", [
    (101, 'Laptop', 1200.00, 50),
    (102, 'Mouse', 25.00, 200),
    (103, 'Keyboard', 75.00, 150)
])
cursor.executemany("INSERT INTO Orders VALUES (?, ?, ?, ?, ?)", [
    (1, 1, '2024-10-15', 1225.00, 'Shipped'),
    (2, 2, '2024-10-20', 75.00, 'Shipped'),
    (3, 1, '2024-11-01', 50.00, 'Pending')
])
cursor.executemany("INSERT INTO OrderDetails VALUES (?, ?, ?, ?, ?)", [
    (1, 1, 101, 1, 1200.00), # Order 1: 1 Laptop
    (2, 1, 102, 1, 25.00),   # Order 1: 1 Mouse
    (3, 2, 103, 1, 75.00),   # Order 2: 1 Keyboard
    (4, 3, 102, 2, 25.00)    # Order 3: 2 Mice
])
print("Sample data inserted.\n")

# --- 3. SQL Queries ---

# Query 1: Fetch all orders by a specific user (Venkatesh, ID 1)
print("--- Query 1: All orders for user 'Venkatesh' ---")
cursor.execute('''
SELECT
    o.order_id,
    o.order_date,
    o.total_amount,
    o.status
FROM Orders o
```



```python
for row in cursor.fetchall():
    print(row)

# Query 2: Find the most purchased product across all orders
print("\n--- Query 2: Most purchased product ---")
cursor.execute('''
SELECT
    p.product_name,
    SUM(od.quantity) AS total_quantity_sold
FROM OrderDetails od
JOIN Products p ON od.product_id = p.product_id
GROUP BY p.product_id
ORDER BY total_quantity_sold DESC
LIMIT 1
''')
print(cursor.fetchone())

# Query 3: Calculate total revenue for a given month (October 2024)
print("\n--- Query 3: Total revenue for October 2024 ---")
cursor.execute('''
SELECT
    SUM(o.total_amount) AS monthly_revenue
FROM Orders o
WHERE STRFTIME('%Y-%m', o.order_date) = '2024-10'
''')
print(f"Total Revenue: ${cursor.fetchone()[0]:.2f}")

# Close the connection
conn.close()
```

OUTPUT:



```
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGR
4.py"
--- Creating Tables: Users, Products, Orders, OrderDetails ---
Tables created successfully.

--- Inserting Sample Data ---
Sample data inserted.

--- Query 1: All orders for user 'Venkatesh' ---
(1, '2024-10-15', 1225.0, 'Shipped')
(3, '2024-11-01', 50.0, 'Pending')

--- Query 2: Most purchased product ---
('Mouse', 3)

--- Query 3: Total revenue for October 2024 ---
Total Revenue: $1300.00
PS C:\PROGRAMMES VSCODE\AI coding> █
```

OBSERVATION:

AI successfully generated a normalized database schema with clear relationships between users, products, and orders. It included foreign keys for data consistency and suggested indexing for faster query execution. The SQL queries for user orders, top-selling products, and monthly revenue were accurate, efficient, and easy to understand.