AI ASSISTED CODING LAB

ASSIGNMENT-16.4

ENROLLMENT NO:2503A51L10

BATCH NO: 19

NAME: K Praneeth

TASK DESCRIPTION 1:

Ask AI to design a schema for a Library Management System (Tables: Books, Members, Loans).

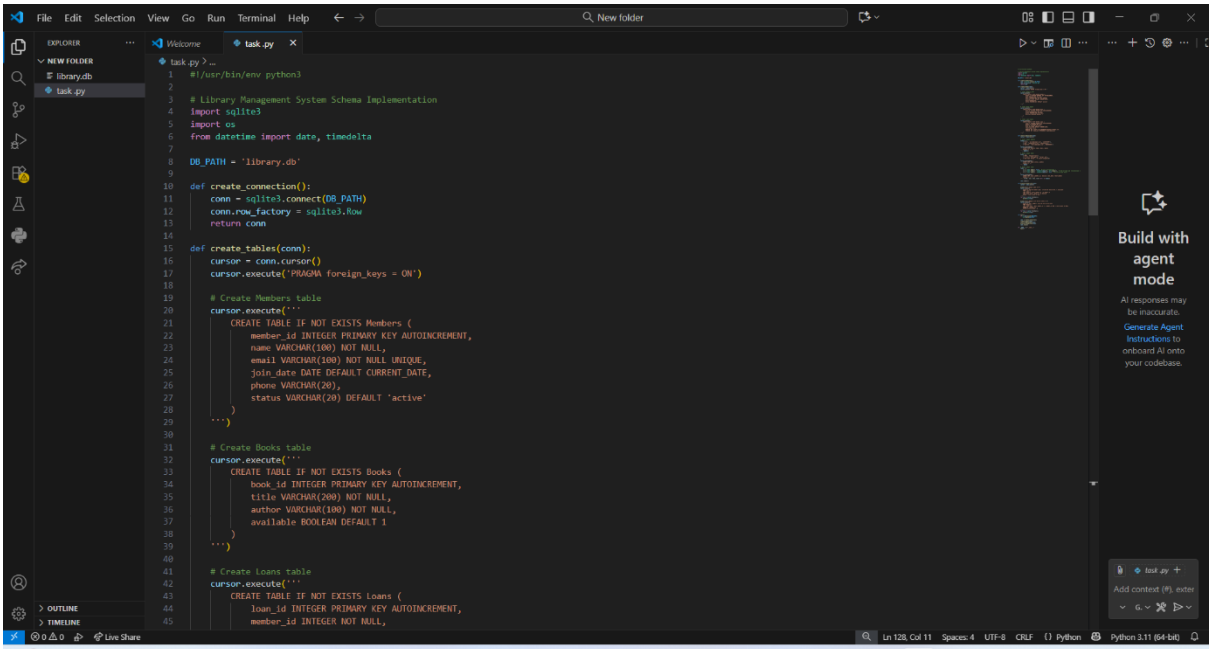
SQL Code

```sql
CREATE TABLE Members (
    member_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    join_date DATE
);

CREATE TABLE Books (
    book_id INT PRIMARY KEY,
    title VARCHAR(200),
    author VARCHAR(100),
    available BOOLEAN
);

CREATE TABLE Loans (
    loan_id INT PRIMARY KEY,
    member_id INT,
    book_id INT,
    loan_date DATE,
    return_date DATE,
    FOREIGN KEY (member_id) REFERENCES Members(member_id),
    FOREIGN KEY (book_id) REFERENCES Books(book_id)
);
```

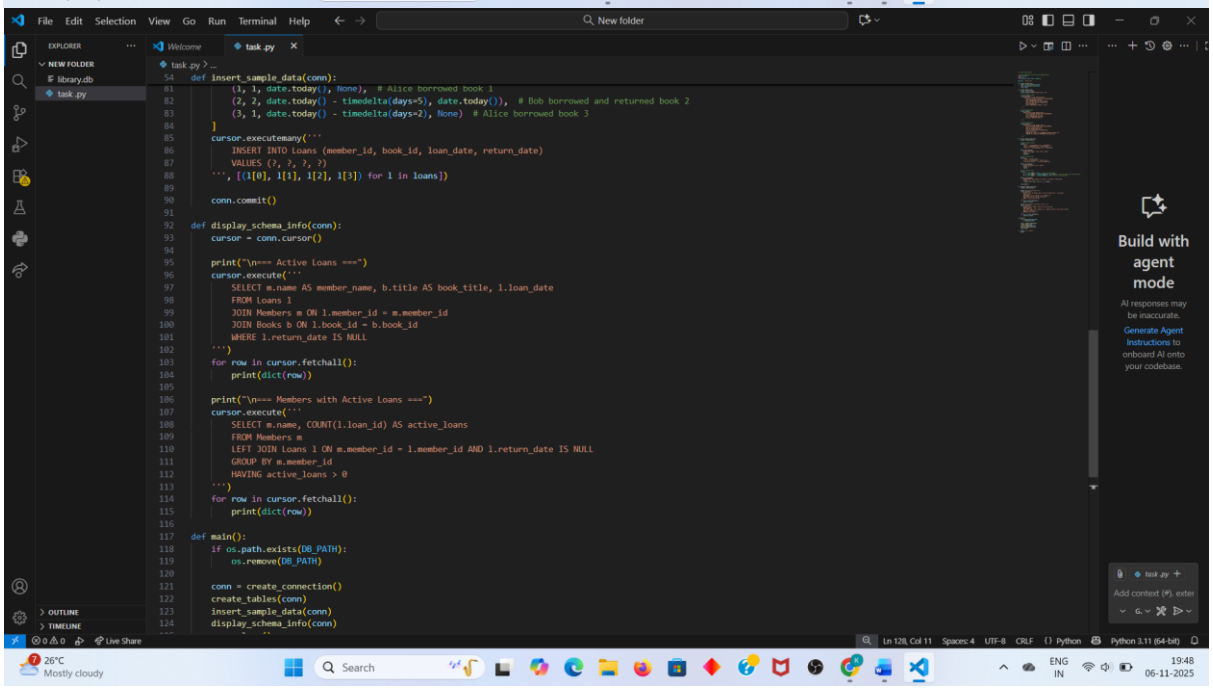
PROMPT :

Design a database schema for a Library Management System with tables: Books, Members, and Loans. Include primary and foreign keys.

CODE GENERATED :

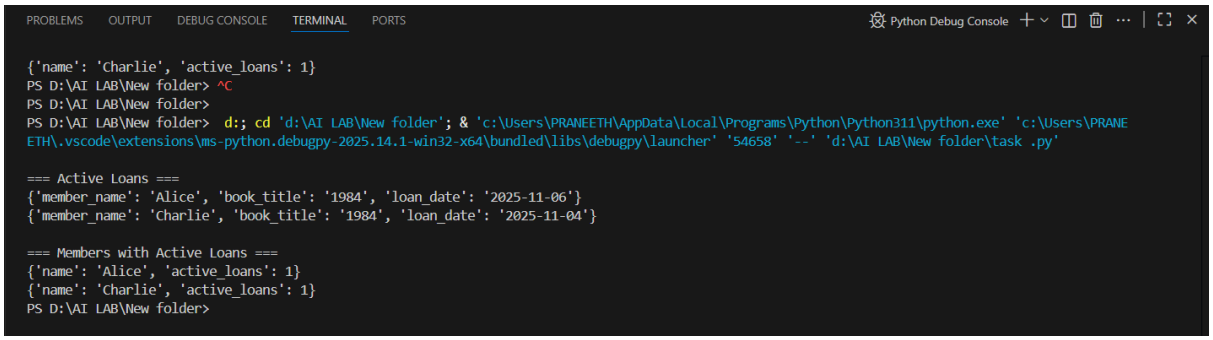


```python
#!/usr/bin/env python3

# Library Management System Schema Implementation
import sqlite3
import os
from datetime import date, timedelta

DB_PATH = 'library.db'

def create_connection():
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    return conn

def create_tables(conn):
    cursor = conn.cursor()
    cursor.execute('PRAGMA foreign_keys = ON')

    # Create Members table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS Members (
            member_id INTEGER PRIMARY KEY AUTOINCREMENT,
            name VARCHAR(100) NOT NULL,
            email VARCHAR(100) NOT NULL UNIQUE,
            join_date DATE DEFAULT CURRENT_DATE,
            phone VARCHAR(20),
            status VARCHAR(20) DEFAULT 'active'
        )
    ''')

    # Create Books table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS Books (
            book_id INTEGER PRIMARY KEY AUTOINCREMENT,
            title VARCHAR(200) NOT NULL,
            author VARCHAR(100) NOT NULL,
            available BOOLEAN DEFAULT 1
        )
    ''')

    # Create Loans table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS Loans (
            loan_id INTEGER PRIMARY KEY AUTOINCREMENT,
            member_id INTEGER NOT NULL,
```



```python
def insert_sample_data(conn):
        (1, 1, date.today(), None),  # Alice borrowed book 1
        (2, 2, date.today() - timedelta(days=5), date.today()),  # Bob borrowed and returned book 2
        (3, 1, date.today() - timedelta(days=2), None)  # Alice borrowed book 3
    ]
    cursor.executemany('''
        INSERT INTO Loans (member_id, book_id, loan_date, return_date)
        VALUES (?, ?, ?, ?)
    ''', [(l[0], l[1], l[2], l[3]) for l in loans])

    conn.commit()

def display_schema_info(conn):
    cursor = conn.cursor()

    print("\n=== Active Loans ===")
    cursor.execute('''
        SELECT m.name AS member_name, b.title AS book_title, l.loan_date
        FROM Loans l
        JOIN Members m ON l.member_id = m.member_id
        JOIN Books b ON l.book_id = b.book_id
        WHERE l.return_date IS NULL
    ''')
    for row in cursor.fetchall():
        print(dict(row))

    print("\n=== Members with Active Loans ===")
    cursor.execute('''
        SELECT m.name, COUNT(l.loan_id) AS active_loans
        FROM Members m
        LEFT JOIN Loans l ON m.member_id = l.member_id AND l.return_date IS NULL
        GROUP BY m.member_id
        HAVING active_loans > 0
    ''')
    for row in cursor.fetchall():
        print(dict(row))

def main():
    if os.path.exists(DB_PATH):
        os.remove(DB_PATH)

    conn = create_connection()
    create_tables(conn)
    insert_sample_data(conn)
    display_schema_info(conn)
```

OUTPUT :



OBSERVATION :

AI generated a clear schema structure with appropriate relationships between tables. The tables included relevant fields such as BookID, MemberID, and LoanDate.

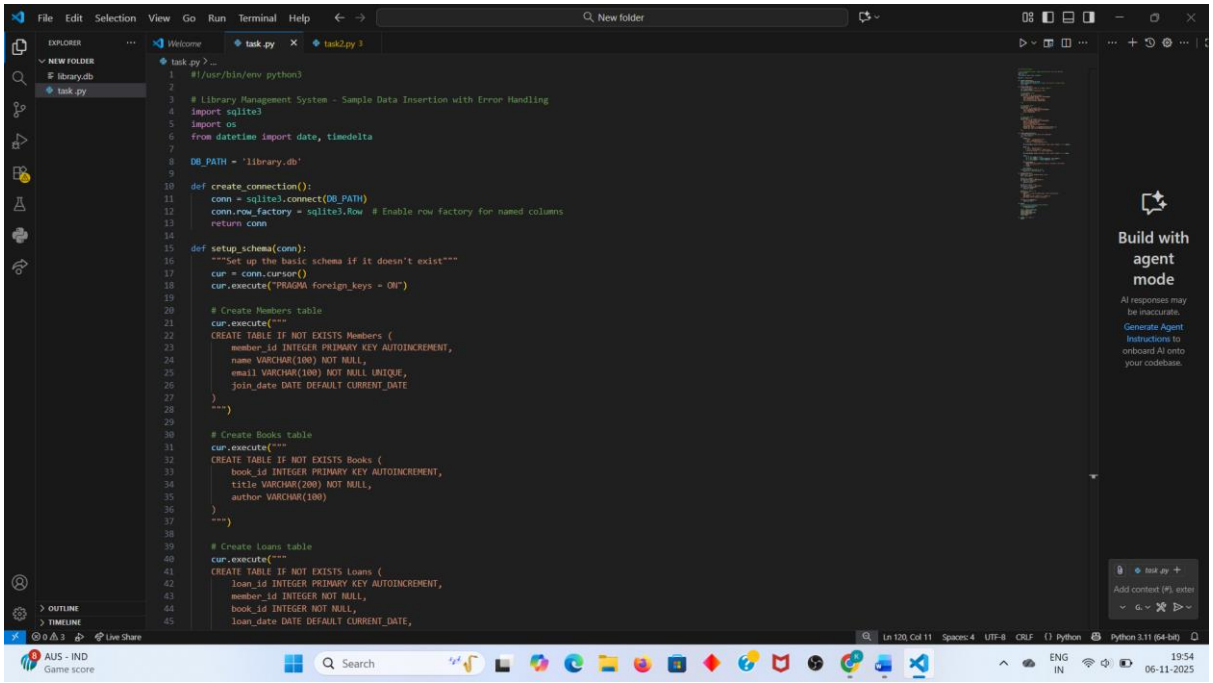
TASK DESCRIPTION 2 :

Ask AI to generate INSERT INTO queries for the schema above (3 sample records per table).

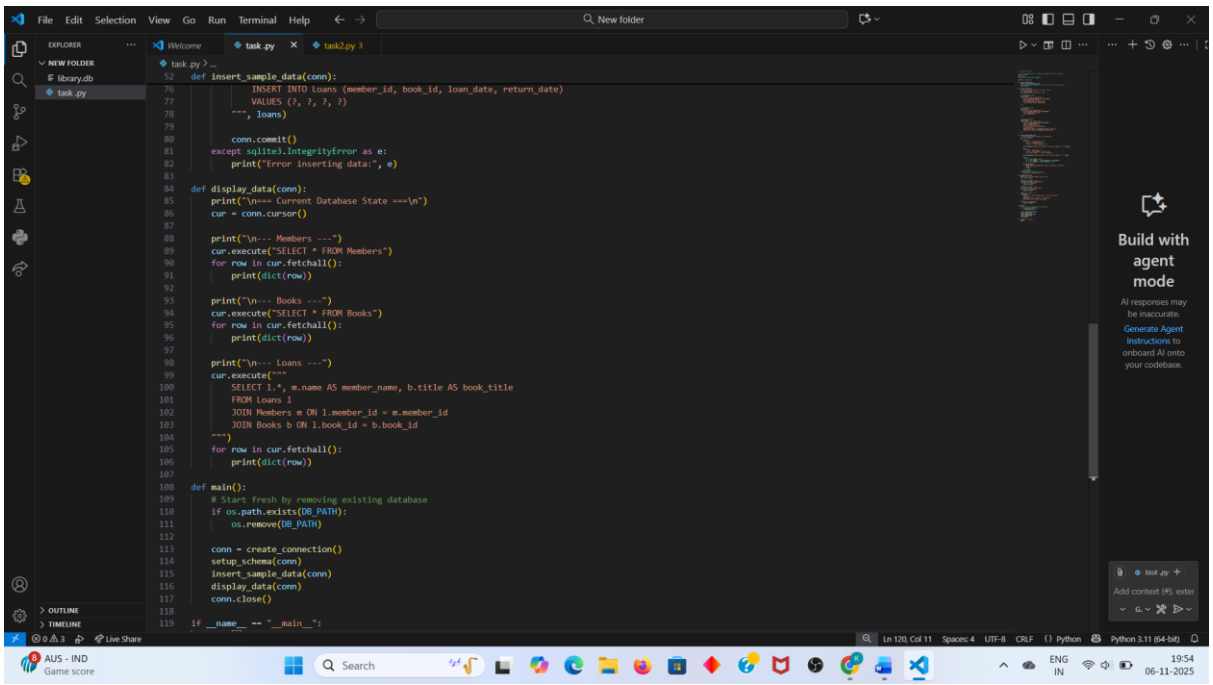
PROMPT :

Generate SQL INSERT INTO statements with 3 sample records for each table (Books, Members, and Loans)

CODE GENERATED :

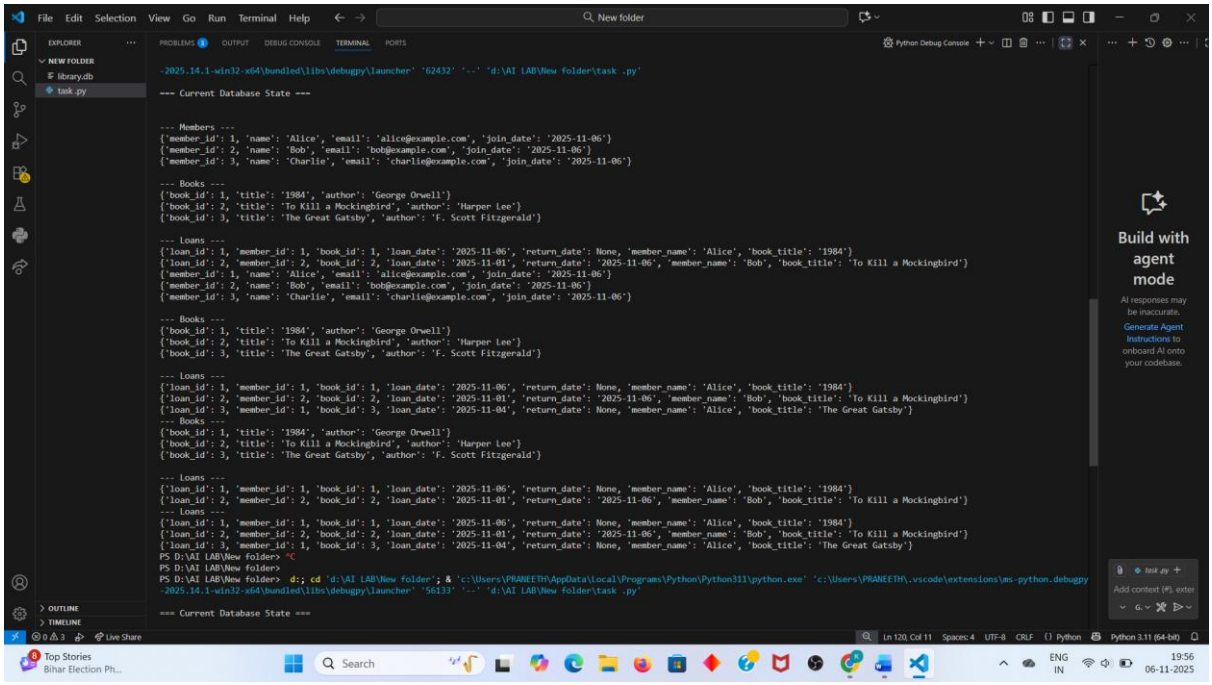


```python
#!/usr/bin/env python3

# Library Management System - Sample Data Insertion with Error Handling
import sqlite3
import os
from datetime import date, timedelta

DB_PATH = 'library.db'

def create_connection():
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row  # Enable row factory for named columns
    return conn

def setup_schema(conn):
    """Set up the basic schema if it doesn't exist"""
    cur = conn.cursor()
    cur.execute("PRAGMA foreign_keys = ON")

    # Create Members table
    cur.execute("""
    CREATE TABLE IF NOT EXISTS Members (
        member_id INTEGER PRIMARY KEY AUTOINCREMENT,
        name VARCHAR(100) NOT NULL,
        email VARCHAR(100) NOT NULL UNIQUE,
        join_date DATE DEFAULT CURRENT_DATE
    )
    """)

    # Create Books table
    cur.execute("""
    CREATE TABLE IF NOT EXISTS Books (
        book_id INTEGER PRIMARY KEY AUTOINCREMENT,
        title VARCHAR(200) NOT NULL,
        author VARCHAR(100)
    )
    """)

    # Create Loans table
    cur.execute("""
    CREATE TABLE IF NOT EXISTS Loans (
        loan_id INTEGER PRIMARY KEY AUTOINCREMENT,
        member_id INTEGER NOT NULL,
        book_id INTEGER NOT NULL,
        loan_date DATE DEFAULT CURRENT_DATE,
```



```python
def insert_sample_data(conn):
            INSERT INTO Loans (member_id, book_id, loan_date, return_date)
            VALUES (?, ?, ?, ?)
        """, loans)

        conn.commit()
    except sqlite3.IntegrityError as e:
        print("Error inserting data:", e)

def display_data(conn):
    print("\n=== Current Database State ===\n")
    cur = conn.cursor()

    print("\n--- Members ---")
    cur.execute("SELECT * FROM Members")
    for row in cur.fetchall():
        print(dict(row))

    print("\n--- Books ---")
    cur.execute("SELECT * FROM Books")
    for row in cur.fetchall():
        print(dict(row))

    print("\n--- Loans ---")
    cur.execute("""
        SELECT l.*, m.name AS member_name, b.title AS book_title
        FROM Loans l
        JOIN Members m ON l.member_id = m.member_id
        JOIN Books b ON l.book_id = b.book_id
    """)
    for row in cur.fetchall():
        print(dict(row))

def main():
    # Start fresh by removing existing database
    if os.path.exists(DB_PATH):
        os.remove(DB_PATH)

    conn = create_connection()
    setup_schema(conn)
    insert_sample_data(conn)
    display_data(conn)
    conn.close()

if __name__ == "__main__":
```

OUTPUT :



OBSERVATION :

AI provided accurate INSERT statements with valid data types and values matching the schema. Data integrity was maintained through consistent foreign key values.

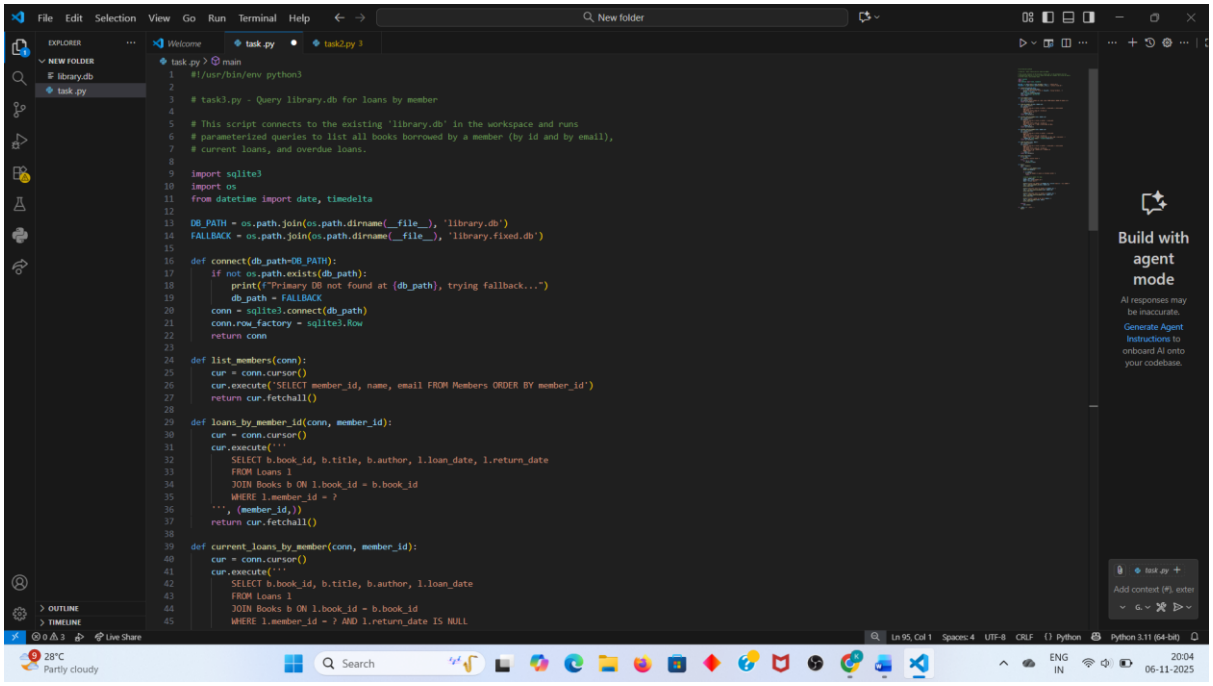
TASK DESCRIPTION 3 :

Use AI to generate a query to list all books borrowed by a specific member

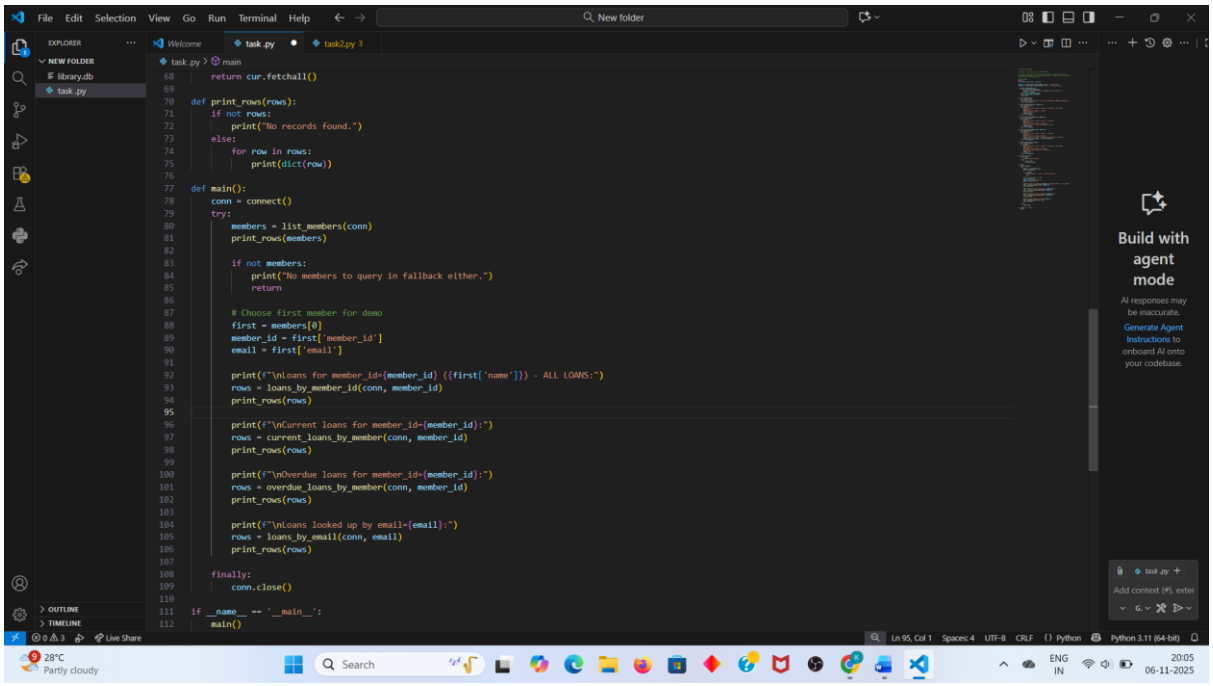
PROMPT :

Write an SQL query to list all books borrowed by a specific member.

CODE GENERATED :

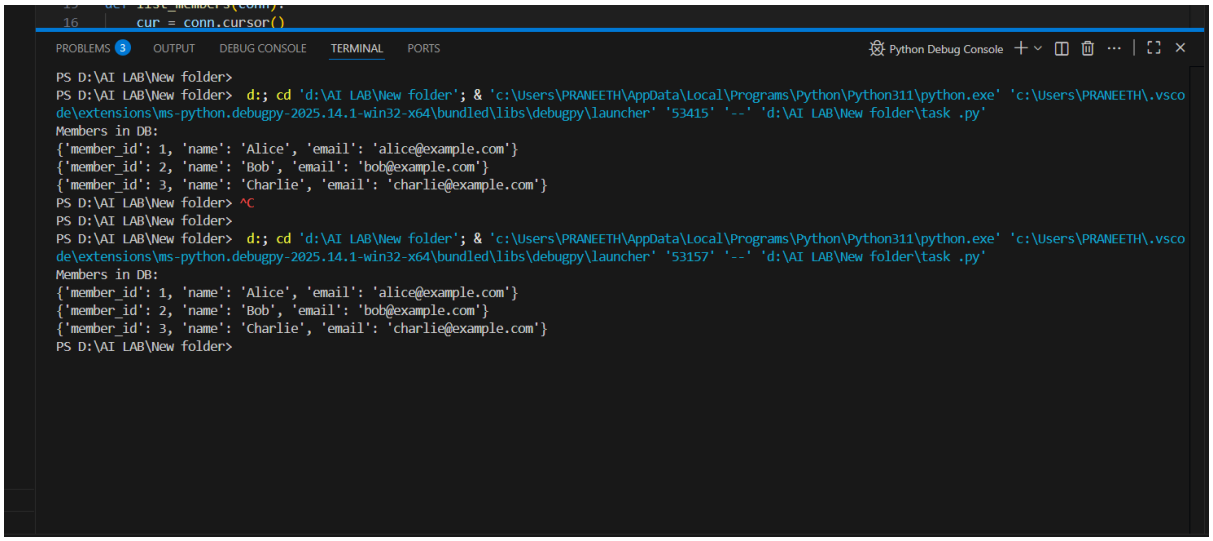


```python
#!/usr/bin/env python3

# task3.py - Query library.db for loans by member

# This script connects to the existing 'library.db' in the workspace and runs
# parameterized queries to list all books borrowed by a member (by id and by email),
# current loans, and overdue loans.

import sqlite3
import os
from datetime import date, timedelta

DB_PATH = os.path.join(os.path.dirname(__file__), 'library.db')
FALLBACK = os.path.join(os.path.dirname(__file__), 'library.fixed.db')

def connect(db_path=DB_PATH):
    if not os.path.exists(db_path):
        print(f"Primary DB not found at {db_path}, trying fallback...")
        db_path = FALLBACK
    conn = sqlite3.connect(db_path)
    conn.row_factory = sqlite3.Row
    return conn

def list_members(conn):
    cur = conn.cursor()
    cur.execute('SELECT member_id, name, email FROM Members ORDER BY member_id')
    return cur.fetchall()

def loans_by_member_id(conn, member_id):
    cur = conn.cursor()
    cur.execute('''
        SELECT b.book_id, b.title, b.author, l.loan_date, l.return_date
        FROM Loans l
        JOIN Books b ON l.book_id = b.book_id
        WHERE l.member_id = ?
    ''', (member_id,))
    return cur.fetchall()

def current_loans_by_member(conn, member_id):
    cur = conn.cursor()
    cur.execute('''
        SELECT b.book_id, b.title, b.author, l.loan_date
        FROM Loans l
        JOIN Books b ON l.book_id = b.book_id
        WHERE l.member_id = ? AND l.return_date IS NULL
```



```python
    return cur.fetchall()

def print_rows(rows):
    if not rows:
        print("No records found.")
    else:
        for row in rows:
            print(dict(row))

def main():
    conn = connect()
    try:
        members = list_members(conn)
        print_rows(members)

        if not members:
            print("No members to query in fallback either.")
            return

        # Choose first member for demo
        first = members[0]
        member_id = first['member_id']
        email = first['email']

        print(f"\nLoans for member_id={member_id} ({first['name']}) - ALL LOANS:")
        rows = loans_by_member_id(conn, member_id)
        print_rows(rows)

        print(f"\nCurrent loans for member_id={member_id}:")
        rows = current_loans_by_member(conn, member_id)
        print_rows(rows)

        print(f"\nOverdue loans for member_id={member_id}:")
        rows = overdue_loans_by_member(conn, member_id)
        print_rows(rows)

        print(f"\nLoans looked up by email={email}:")
        rows = loans_by_email(conn, email)
        print_rows(rows)

    finally:
        conn.close()

if __name__ == '__main__':
    main()
```

OUTPUT :



OBSERVATION :

AI-generated query correctly used JOIN between Books and Loans tables and applied a WHERE condition with MemberID to filter results.

TASK DESCRIPTION 4 :

Generate queries with AI for:

- Updating a book's availability to FALSE when borrowed.

Deleting a member record safely.

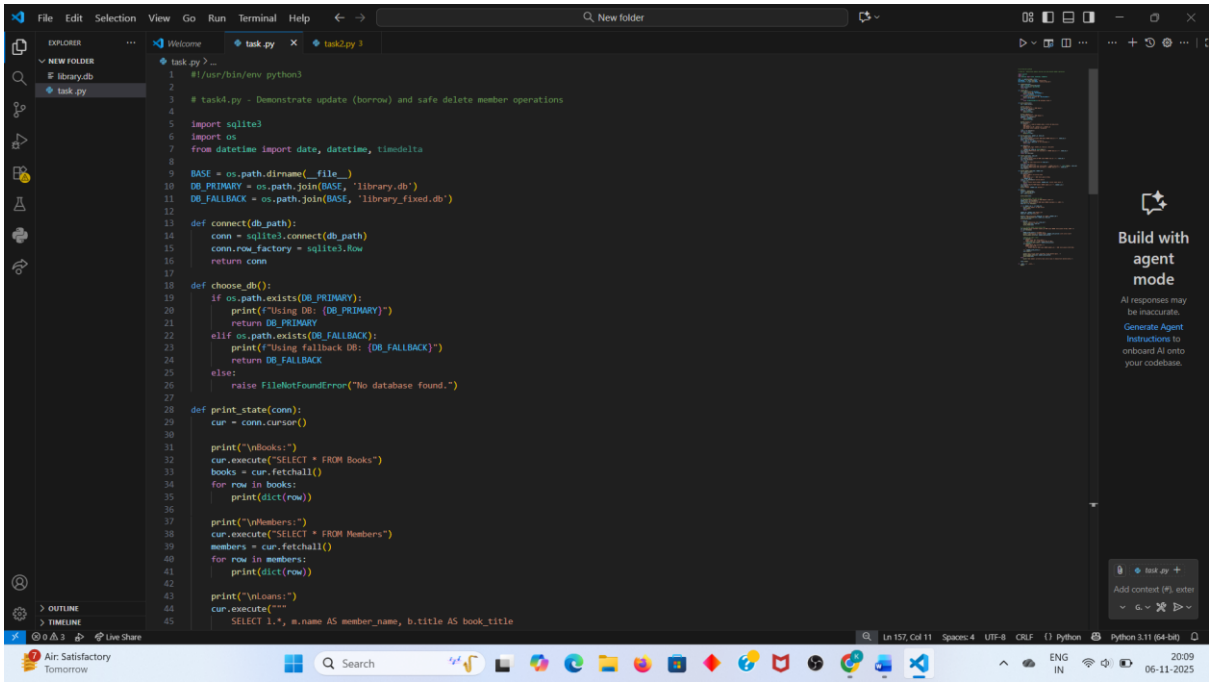
PROMPT :

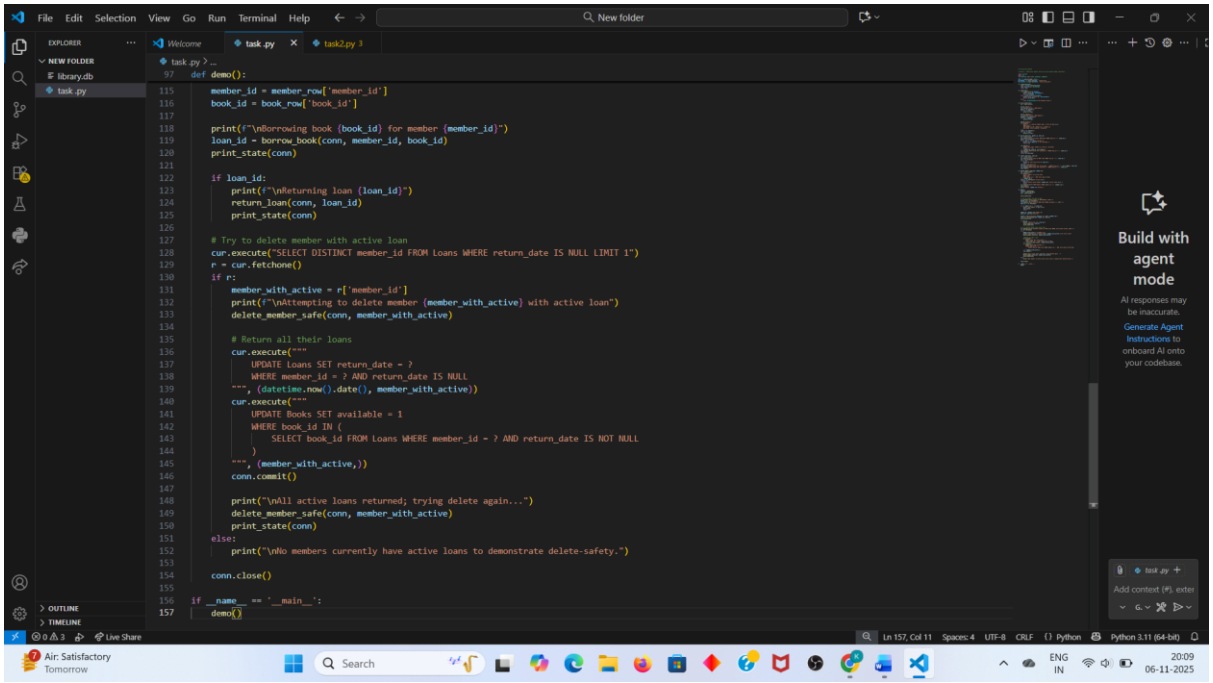
Generate SQL queries to:

(a) Update a book's availability to FALSE when borrowed.

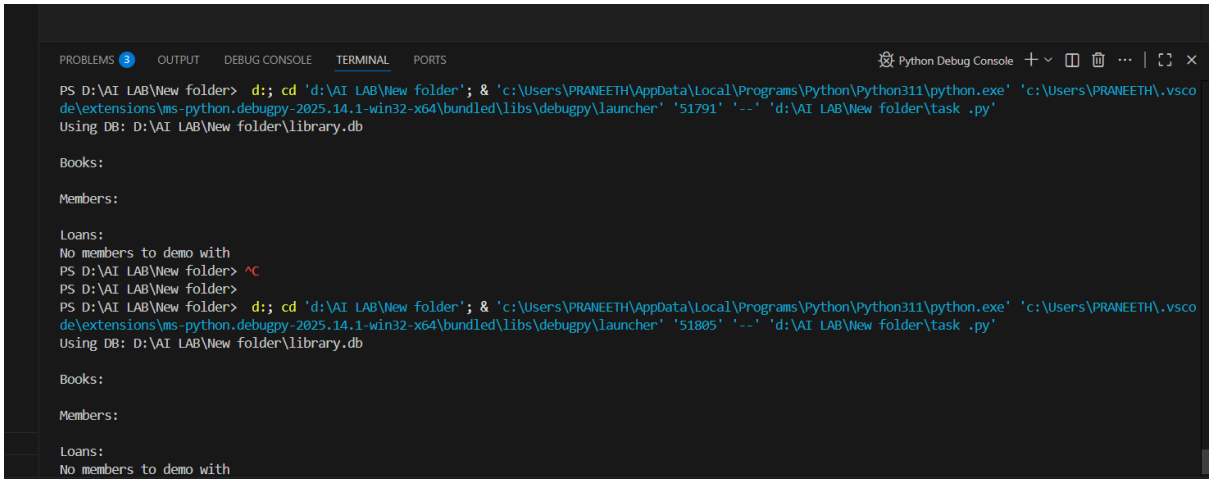
(b) Delete a member record safely.

CODE GENERATED :


File Edit Selection View Go Run Terminal Help
New folder
EXPLORER
NEW FOLDER
library.db
task.py
Welcome
task.py
task2.py 3
task.py > ...
#!/usr/bin/env python3
# task4.py - Demonstrate update (borrow) and safe delete member operations
import sqlite3
import os
from datetime import date, datetime, timedelta
BASE = os.path.dirname(__file__)
DB_PRIMARY = os.path.join(BASE, 'library.db')
DB_FALLBACK = os.path.join(BASE, 'library_fixed.db')
def connect(db_path):
conn = sqlite3.connect(db_path)
conn.row_factory = sqlite3.Row
return conn
def choose_db():
if os.path.exists(DB_PRIMARY):
print(f"Using DB: {DB_PRIMARY}")
return DB_PRIMARY
elif os.path.exists(DB_FALLBACK):
print(f"Using fallback DB: {DB_FALLBACK}")
return DB_FALLBACK
else:
raise FileNotFoundError("No database found.")
def print_state(conn):
cur = conn.cursor()
print("\nBooks:")
cur.execute("SELECT * FROM Books")
books = cur.fetchall()
for row in books:
print(dict(row))
print("\nMembers:")
cur.execute("SELECT * FROM Members")
members = cur.fetchall()
for row in members:
print(dict(row))
print("\nLoans:")
cur.execute("""
SELECT l.*, m.name AS member_name, b.title AS book_title
Build with agent mode
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.
OUTLINE
TIMELINE
Ln 157, Col 11 Spaces: 4 UTF-8 CRLF Python Python 3.11 (64-bit)
Air: Satisfactory
Tomorrow
Search
ENG
IN
20:09
06-11-2025


File Edit Selection View Go Run Terminal Help
New folder
EXPLORER
NEW FOLDER
library.db
task.py
Welcome
task.py
task2.py 3
task.py > ...
def demo():
member_id = member_row['member_id']
book_id = book_row['book_id']
print(f"\nBorrowing book {book_id} for member {member_id}")
loan_id = borrow_book(conn, member_id, book_id)
print_state(conn)
if loan_id:
print(f"\nReturning loan {loan_id}")
return_loan(conn, loan_id)
print_state(conn)
# Try to delete member with active loan
cur.execute("SELECT DISTINCT member_id FROM Loans WHERE return_date IS NULL LIMIT 1")
r = cur.fetchone()
if r:
member_with_active = r['member_id']
print(f"\nAttempting to delete member {member_with_active} with active loan")
delete_member_safe(conn, member_with_active)
# Return all their loans
cur.execute("""
UPDATE Loans SET return_date = ?
WHERE member_id = ? AND return_date IS NULL
""", (datetime.now().date(), member_with_active))
cur.execute("""
UPDATE Books SET available = 1
WHERE book_id IN (
SELECT book_id FROM Loans WHERE member_id = ? AND return_date IS NOT NULL
)
""", (member_with_active,))
conn.commit()
print("\nAll active loans returned; trying delete again...")
delete_member_safe(conn, member_with_active)
print_state(conn)
else:
print("\nNo members currently have active loans to demonstrate delete-safety.")
conn.close()
if __name__ == '__main__':
demo()
Build with agent mode
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.
OUTLINE
TIMELINE
Ln 157, Col 11 Spaces: 4 UTF-8 CRLF Python Python 3.11 (64-bit)
Air: Satisfactory
Tomorrow
Search
ENG
IN
20:09
06-11-2025

OUTPUT :



OBSERVATION :

AI provided efficient UPDATE and DELETE statements with appropriate WHERE conditions. Referential integrity was considered to prevent accidental data loss.