

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

```
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 :

The image shows two side-by-side screenshots of a code editor interface, likely Visual Studio Code, displaying Python scripts for a library management system. Both screenshots show the same file structure and code content, with minor differences in the status bar at the bottom.

File Structure:

- Explorer: library.db
- task.py

Code Content (task.py):

```
task.py > ...
1 #!/usr/bin/env python3
2
3 # Library Management System Schema Implementation
4 import sqlite3
5 import os
6 from datetime import date, timedelta
7
8 DB_PATH = 'library.db'
9
10 def create_connection():
11     conn = sqlite3.connect(DB_PATH)
12     conn.row_factory = sqlite3.Row
13     return conn
14
15 def create_tables(conn):
16     cursor = conn.cursor()
17     cursor.execute('PRAGMA foreign_keys = ON')
18
19     # Create Members table
20     cursor.execute('''
21         CREATE TABLE IF NOT EXISTS Members (
22             member_id INTEGER PRIMARY KEY AUTOINCREMENT,
23             name VARCHAR(100) NOT NULL,
24             email VARCHAR(100) NOT NULL UNIQUE,
25             join_date DATE DEFAULT CURRENT_DATE,
26             phone VARCHAR(20),
27             status VARCHAR(20) DEFAULT 'active'
28         );
29     ''')
30
31     # Create Books table
32     cursor.execute('''
33         CREATE TABLE IF NOT EXISTS Books (
34             book_id INTEGER PRIMARY KEY AUTOINCREMENT,
35             title VARCHAR(200) NOT NULL,
36             author VARCHAR(100) NOT NULL,
37             available BOOLEAN DEFAULT 1
38         );
39     ''')
40
41     # Create Loans table
42     cursor.execute('''
43         CREATE TABLE IF NOT EXISTS Loans (
44             loan_id INTEGER PRIMARY KEY AUTOINCREMENT,
45             member_id INTEGER NOT NULL,
46             book_id INTEGER NOT NULL,
47             loan_date DATE,
48             return_date DATE
49         );
50     ''')
51
52     # Insert sample data
53     cursor.executemany('
54         INSERT INTO Loans(member_id, book_id, loan_date, return_date)
55         VALUES (?, ?, ?, ?)
56     ', [(1, 1, date.today(), None), # Alice borrowed book 1
57          (2, 2, date.today() - timedelta(days=5), date.today()), # Bob borrowed and returned book 2
58          (3, 1, date.today() - timedelta(days=2), None) # Alice borrowed book 3
59     ])
60
61     cursor.executemany('
62         INSERT INTO Books(member_id, book_id, book_title, loan_date)
63         VALUES (?, ?, ?, ?)
64     ', [(1, 1, 'The Great Gatsby', date.today()),
65          (1, 2, 'Pride and Prejudice', date.today()),
66          (1, 3, 'War and Peace', date.today())
67      ])
68
69     conn.commit()
70
71     def display_schema_info(conn):
72         cursor = conn.cursor()
73
74         print("==== Active Loans ===")
75         cursor.execute('
76             SELECT l.member_name AS member_name, b.title AS book_title, l.loan_date
77             FROM Loans l
78             JOIN Members m ON l.member_id = m.member_id
79             JOIN Books b ON l.book_id = b.book_id
80             WHERE l.return_date IS NULL
81         ')
82         for row in cursor.fetchall():
83             print(dict(row))
84
85         print("==== Members with Active Loans ===")
86         cursor.execute('
87             SELECT m.name, COUNT(l.loan_id) AS active_loans
88             FROM Members m
89             LEFT JOIN Loans l ON m.member_id = l.member_id AND l.return_date IS NULL
90             GROUP BY m.member_id
91             HAVING active_loans > 0
92         ')
93         for row in cursor.fetchall():
94             print(dict(row))
95
96     def main():
97         if os.path.exists(DB_PATH):
98             os.remove(DB_PATH)
99
100        conn = create_connection()
101        create_tables(conn)
102        insert_sample_data(conn)
103        display_schema_info(conn)
104
105    if __name__ == '__main__':
106        main()
```

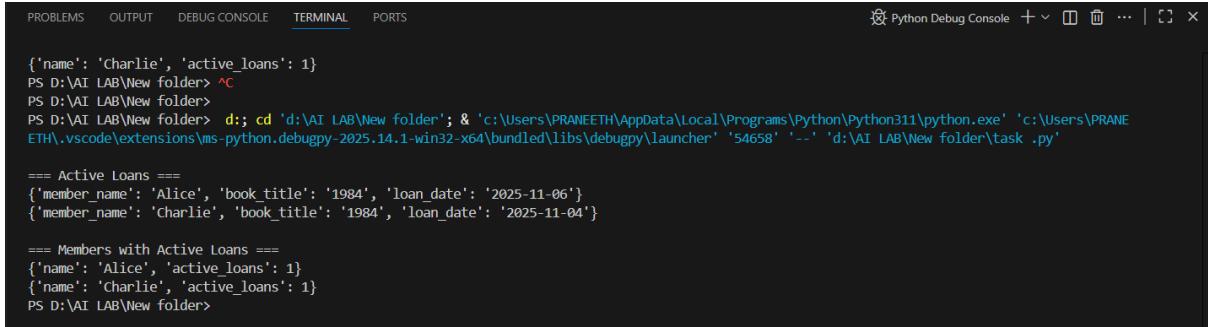
Status Bar (Top Screenshot):

- Ln 128, Col 11
- Spaces: 4
- UTF-8
- CR/LF
- Python
- Python 3.11 (64-bit)

Status Bar (Bottom Screenshot):

- Ln 128, Col 11
- Spaces: 4
- UTF-8
- CR/LF
- Python
- Python 3.11 (64-bit)

OUTPUT :



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console + □ □ ... [ ] ×

{'name': 'Charlie', 'active_loans': 1}
PS D:\AI LAB\New folder> ^C
PS D:\AI LAB\New folder>
PS D:\AI LAB\New folder> d;; cd 'd:\AI LAB\New folder'; & 'c:\Users\PRANEETH\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\PRANEETH\.vscode\extensions\ms-python.debugpy-2025.14.1-win32-x64\bundled\libs\debugpy\launcher' '54658' '--' 'd:\AI LAB\New folder\task .py'

== Active Loans ==
{'member_name': 'Alice', 'book_title': '1984', 'loan_date': '2025-11-06'}
{'member_name': 'Charlie', 'book_title': '1984', 'loan_date': '2025-11-04'}

== Members with Active Loans ==
{'name': 'Alice', 'active_loans': 1}
{'name': 'Charlie', 'active_loans': 1}
PS D:\AI LAB\New folder>
```

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 :

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The left sidebar has icons for Explorer, File Manager, Search, and others. The main area displays a Python script named `task.py`. The code is as follows:

```
task.py
1 #!/usr/bin/env python3
2
3 # Library Management System - Sample Data Insertion with Error Handling
4
5 import sqlite3
6 import os
7 from datetime import date, timedelta
8
9 DB_PATH = 'library.db'
10
11 def create_connection():
12     conn = sqlite3.connect(DB_PATH)
13     conn.row_factory = sqlite3.Row # Enable row factory for named columns
14     return conn
15
16 def setup_schema(conn):
17     """Set up the basic schema if it doesn't exist"""
18     cur = conn.cursor()
19     cur.execute("PRAGMA foreign_keys = ON")
20
21     # Create Members table
22     cur.execute("""
23         CREATE TABLE IF NOT EXISTS Members (
24             member_id INTEGER PRIMARY KEY AUTOINCREMENT,
25             name VARCHAR(100) NOT NULL,
26             email VARCHAR(100) NOT NULL UNIQUE,
27             join_date DATE DEFAULT CURRENT_DATE
28         );
29     """)
30
31     # Create Books table
32     cur.execute("""
33         CREATE TABLE IF NOT EXISTS Books (
34             book_id INTEGER PRIMARY KEY AUTOINCREMENT,
35             title VARCHAR(200) NOT NULL,
36             author VARCHAR(100)
37         );
38     """)
39
40     # Create Loans table
41     cur.execute("""
42         CREATE TABLE IF NOT EXISTS Loans (
43             loan_id INTEGER PRIMARY KEY AUTOINCREMENT,
44             member_id INTEGER NOT NULL,
45             book_id INTEGER NOT NULL,
46             loan_date DATE DEFAULT CURRENT_DATE,
47         );
48     """)
49
50     conn.commit()
51
52     print("Tables created successfully!")
53
54     # Insert sample data
55     cur.execute("INSERT INTO Members (name, email) VALUES ('John Doe', 'john.doe@example.com')")
56     cur.execute("INSERT INTO Books (title, author) VALUES ('The Great Gatsby', 'F. Scott Fitzgerald')")
57     cur.execute("INSERT INTO Loans (member_id, book_id, loan_date) VALUES (1, 1, '2023-10-01')")
```

The status bar at the bottom shows file statistics: Line 120, Col 11, Spaces: 4, UTF-8, CR LF, Python, Python 3.11 (64-bit), and a timestamp: 19:54 IN 06-11-2023.

The screenshot shows a Python IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** New folder.
- Sidebar:** Explorer (task.py selected), Library.db, Outline, Timeline, Live Share.
- Code Editor:** The main area displays Python code for a SQLite database task. The code includes functions for inserting sample data, displaying data, and a main function that removes the database if it exists and then creates it, sets up the schema, inserts sample data, and displays the data. It also includes a check for the __name__ variable.
- Right Panel:** A preview pane showing the results of the code execution, including tables for Members, Books, and Loans.
- Bottom Status Bar:** In 120, Col 11, Spaces: 4, UTF-8, CRLF, Python 3.11 (64-bit), ENG IN, 19:54, 06-11-2023.
- Bottom Icons:** Save, Run, Stop, Refresh, etc.

OUTPUT :

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows a folder named "library.db" and a file named "task.py".
- Terminal:** Displays the generated SQL code for the library management system. The code includes:
 - Creating a database and tables: `CREATE DATABASE library;` and `CREATE TABLE member (member_id INT PRIMARY KEY, name VARCHAR(50), email VARCHAR(100), join_date DATE);`.
 - Inserting data into the member table:

```
INSERT INTO member VALUES (1, 'Alice', 'alice@example.com', '2025-11-06');
INSERT INTO member VALUES (2, 'Bob', 'bob@example.com', '2025-11-06');
INSERT INTO member VALUES (3, 'Charlie', 'charlie@example.com', '2025-11-06');
```
 - Creating a book table: `CREATE TABLE book (book_id INT PRIMARY KEY, title VARCHAR(100), author VARCHAR(100));`.
 - Inserting data into the book table:

```
INSERT INTO book VALUES (1, '1984', 'George Orwell');
INSERT INTO book VALUES (2, 'To Kill a Mockingbird', 'Harper Lee');
INSERT INTO book VALUES (3, 'The Great Gatsby', 'F. Scott Fitzgerald');
```
 - Creating a loan table: `CREATE TABLE loan (loan_id INT PRIMARY KEY, member_id INT, book_id INT, loan_date DATE, return_date DATE);`.
 - Inserting data into the loan table:

```
INSERT INTO loan VALUES (1, 1, 1, '2025-11-06', '2025-11-06');
INSERT INTO loan VALUES (2, 2, 2, '2025-11-01', '2025-11-06');
INSERT INTO loan VALUES (3, 3, 3, '2025-11-04', '2025-11-06');
```
- Output Panel:** Shows the message "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." and a "Build with agent mode" button.
- Bottom Status Bar:** Shows the terminal command PS D:\VAI LAB\New folder> & cd d:\VAI LAB\New folder& & c:\Users\PRANEETH\AppData\Local\Programs\Python\Python311\python.exe 'c:\Users\PRANEETH\vscode\extensions\ms-python.debugpy\2025.14.1-win32-x64\bundled\libs\debugpy\launcher' '56133' ... 'd:\VAI LAB\New folder\task.py'
- System Tray:** Shows icons for battery, signal, and date/time (18:55, 06-11-2025).

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 :

```
File Edit Selection View Go Run Terminal Help <- > Q, New folder
EXPLORER ... Welcome task.py task2.py
NEW FOLDER library.db task.py
task.py > Q main
1 #!/usr/bin/env python3
2
3 # task3.py - Query library.db for loans by member
4
5 # This script connects to the existing 'library.db' in the workspace and runs
6 # parameterized queries to list all books borrowed by a member (by id and by email),
7 # current loans, and overdue loans.
8
9 import sqlite3
10 import os
11 from datetime import date, timedelta
12
13 DB_PATH = os.path.join(os.path.dirname(__file__), 'library.db')
14 FALBACK = os.path.join(os.path.dirname(__file__), 'library.Fixed.db')
15
16 def connect(db_path=DB_PATH):
17     if not os.path.exists(db_path):
18         print("Primary DB not found at {} (db_path), trying fallback...")
19     db_path = FALBACK
20     conn = sqlite3.connect(db_path)
21     conn.row_factory = sqlite3.Row
22     return conn
23
24 def list_members(conn):
25     cur = conn.cursor()
26     cur.execute("SELECT member_id, name, email FROM Members ORDER BY member_id")
27     return cur.fetchall()
28
29 def loans_by_member_id(conn, member_id):
30     cur = conn.cursor()
31     cur.execute(
32         "SELECT l.book_id, b.title, b.author, l.loan_date, l.return_date
33         FROM Loans l
34         JOIN Books b ON l.book_id = b.book_id
35         WHERE l.member_id = ?"
36         .format(member_id))
37     return cur.fetchall()
38
39 def current_loans_by_member(conn, member_id):
40     cur = conn.cursor()
41     cur.execute(
42         "SELECT l.book_id, b.title, b.author, l.loan_date
43         FROM Loans l
44         JOIN Books b ON l.book_id = b.book_id
45         WHERE l.member_id = ? AND l.return_date IS NULL"
46         .format(member_id))
47
48     return cur.fetchall()
49
50     def print_rows(rows):
51         if not rows:
52             print("No records found.")
53         else:
54             for row in rows:
55                 print(dict(row))
56
57     def main():
58         conn = connect()
59         try:
60             members = list_members(conn)
61             print_rows(members)
62
63             if not members:
64                 print("No members to query in fallback either.")
65             return
66
67             # Choose first member for demo
68             first = members[0]
69             member_id = first['member_id']
70             email = first['email']
71
72             print("\nLoans for member_id={member_id} ({first['name']}): ALL LOANS:")
73             rows = loans_by_member_id(conn, member_id)
74             print_rows(rows)
75
76             print("\nCurrent loans for member_id={member_id}:")
77             rows = current_loans_by_member(conn, member_id)
78             print_rows(rows)
79
80             print("\nOverdue loans for member_id={member_id}:")
81             rows = overdue_loans_by_member(conn, member_id)
82             print_rows(rows)
83
84             print("\nLoans looked up by email={email}:")
85             rows = loans_by_email(conn, email)
86             print_rows(rows)
87
88     finally:
89         conn.close()
90
91     if __name__ == '__main__':
92         main()
```

Build with agent mode

All responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

task ay + Add context (#), enter

G > D > v

Ln 95, Col 1 Spaces: 4 UTE-B CRLF Python Python 3.11 (64-bit)

28°C Partly cloudy

Search

Windows Taskbar

20:04 06-11-2025

```
File Edit Selection View Go Run Terminal Help <- > Q, New folder
EXPLORER ... Welcome task.py task2.py
NEW FOLDER library.db task.py
task.py > Q main
68     return cur.fetchall()
69
70 def print_rows(rows):
71     if not rows:
72         print("No records found.")
73     else:
74         for row in rows:
75             print(dict(row))
76
77 def main():
78     conn = connect()
79     try:
80         members = list_members(conn)
81         print_rows(members)
82
83         if not members:
84             print("No members to query in fallback either.")
85             return
86
87         # Choose first member for demo
88         first = members[0]
89         member_id = first['member_id']
90         email = first['email']
91
92         print("\nLoans for member_id={member_id} ({first['name']}): ALL LOANS:")
93         rows = loans_by_member_id(conn, member_id)
94         print_rows(rows)
95
96         print("\nCurrent loans for member_id={member_id}:")
97         rows = current_loans_by_member(conn, member_id)
98         print_rows(rows)
99
100        print("\nOverdue loans for member_id={member_id}:")
101        rows = overdue_loans_by_member(conn, member_id)
102        print_rows(rows)
103
104        print("\nLoans looked up by email={email}:")
105        rows = loans_by_email(conn, email)
106        print_rows(rows)
107
108    finally:
109        conn.close()
110
111    if __name__ == '__main__':
112        main()
```

Build with agent mode

All responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.

task ay + Add context (#), enter

G > D > v

Ln 95, Col 1 Spaces: 4 UTE-B CRLF Python Python 3.11 (64-bit)

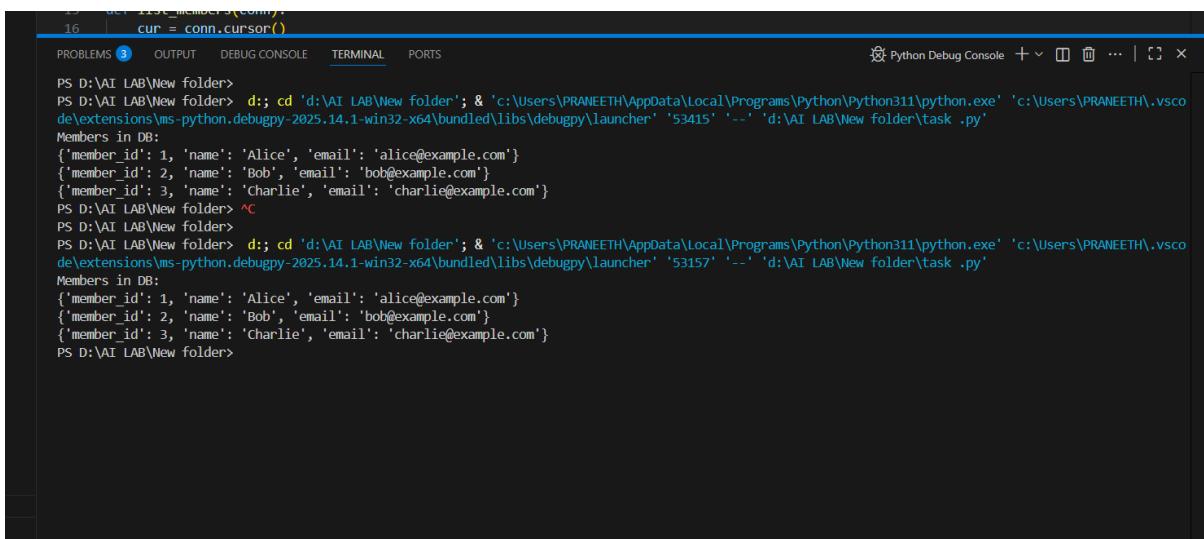
28°C Partly cloudy

Search

Windows Taskbar

20:05 06-11-2025

OUTPUT :



The screenshot shows a terminal window with the following content:

```
PS D:\AI LAB\New folder> cur = conn.cursor()
PS D:\AI LAB\New folder> d; cd 'd:\AI LAB\New folder'; & 'c:\Users\PRANEETH\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\PRANEETH\.vscodeextensions\ms-python.debugpy-2025.14.1-win32-x64\bundled\libs\debugpy\launcher' '53415' '--' 'd:\AI LAB\New folder\task .py'
Members in DB:
[{"member_id": 1, "name": "Alice", "email": "alice@example.com"}, {"member_id": 2, "name": "Bob", "email": "bob@example.com"}, {"member_id": 3, "name": "Charlie", "email": "charlie@example.com"}]
PS D:\AI LAB\New folder> ~c
PS D:\AI LAB\New folder> d; cd 'd:\AI LAB\New folder'; & 'c:\Users\PRANEETH\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\PRANEETH\.vscodeextensions\ms-python.debugpy-2025.14.1-win32-x64\bundled\libs\debugpy\launcher' '53157' '--' 'd:\AI LAB\New folder\task .py'
Members in DB:
[{"member_id": 1, "name": "Alice", "email": "alice@example.com"}, {"member_id": 2, "name": "Bob", "email": "bob@example.com"}, {"member_id": 3, "name": "Charlie", "email": "charlie@example.com"}]
PS D:\AI LAB\New folder>
```

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 :

```

1 #!/usr/bin/env python3
2
3 # task4.py - Demonstrate update (borrow) and safe delete member operations
4
5 import sqlite3
6 import os
7 from datetime import date, datetime, timedelta
8
9 BASE = os.path.dirname(__file__)
10 DB_PRIMARY = os.path.join(BASE, 'library.db')
11 DB_FALLBACK = os.path.join(BASE, 'library_fixed.db')
12
13 def connect(db_path):
14     conn = sqlite3.connect(db_path)
15     conn.row_factory = sqlite3.Row
16     return conn
17
18 def choose_db():
19     if os.path.exists(DB_PRIMARY):
20         print(f"Using DB: {DB_PRIMARY}")
21         return DB_PRIMARY
22     elif os.path.exists(DB_FALLBACK):
23         print(f"Using fallback DB: {DB_FALLBACK}")
24         return DB_FALLBACK
25     else:
26         raise FileNotFoundError("No database found.")
27
28 def print_state(conn):
29     cur = conn.cursor()
30
31     print("\nBooks:")
32     cur.execute("SELECT * FROM Books")
33     books = cur.fetchall()
34     for row in books:
35         print(dict(row))
36
37     print("\nMembers:")
38     cur.execute("SELECT * FROM Members")
39     members = cur.fetchall()
40     for row in members:
41         print(dict(row))
42
43     print("\nLoans:")
44     cur.execute("""
45             SELECT 1.* , m.name AS member_name, b.title AS book_title

```

```

97     def demo():
98         member_id = member_row['member_id']
99         book_id = book_row['book_id']
100
101         print(f"\nBorrowing book {book_id} for member {member_id}")
102         loan_id = borrow_book(conn, member_id, book_id)
103         print_loan(conn)
104
105         # Try to delete member with active loan
106         cur.execute("SELECT DISTINCT member_id FROM Loans WHERE return_date IS NULL LIMIT 1")
107         r = cur.fetchone()
108         if r:
109             member_with_active = r['member_id']
110             print(f"\nAttempting to delete member {member_with_active} with active loan")
111             delete_member_safe(conn, member_with_active)
112
113             # Returns all their loans
114             cur.execute("""
115                 UPDATE Loans SET return_date = ?
116                 WHERE member_id = ? AND return_date IS NULL
117                 """, (datetime.now().date(), member_with_active))
118
119             cur.execute("""
120                 UPDATE Books SET available = 1
121                 WHERE book_id IN (
122                     SELECT book_id FROM Loans WHERE member_id = ? AND return_date IS NOT NULL
123                     )
124                 """, (member_with_active,))
125             conn.commit()
126
127             print("\nAll active loans returned; trying delete again...")
128             delete_member_safe(conn, member_with_active)
129             print_loan(conn)
130         else:
131             print("\nNo members currently have active loans to demonstrate delete-safety.")
132
133         conn.close()
134
135     if __name__ == '__main__':
136         demo()

```

OUTPUT :

The screenshot shows a terminal window with the following text:

```
PROBLEMS ③ OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console + ×
```

```
PS D:\AI LAB\New folder> d; cd 'd:\AI LAB\New folder'; & 'c:\Users\PRANEETH\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\PRANEETH\.vscode\extensions\ms-python.debugpy-2025.14.1-win32-x64\bundled\libs\debugpy\launcher' '51791' '--' 'd:\AI LAB\New folder\task .py'
Using DB: D:\AI LAB\New folder\library.db

Books:
Members:
Loans:
No members to demo with
PS D:\AI LAB\New folder> ^C
PS D:\AI LAB\New folder>
PS D:\AI LAB\New folder> d; cd 'd:\AI LAB\New folder'; & 'c:\Users\PRANEETH\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\PRANEETH\.vscode\extensions\ms-python.debugpy-2025.14.1-win32-x64\bundled\libs\debugpy\launcher' '51805' '--' 'd:\AI LAB\New folder\task .py'
Using DB: D:\AI LAB\New folder\library.db

Books:
Members:
Loans:
No members to demo with
```

OBSERVATION :

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