

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).

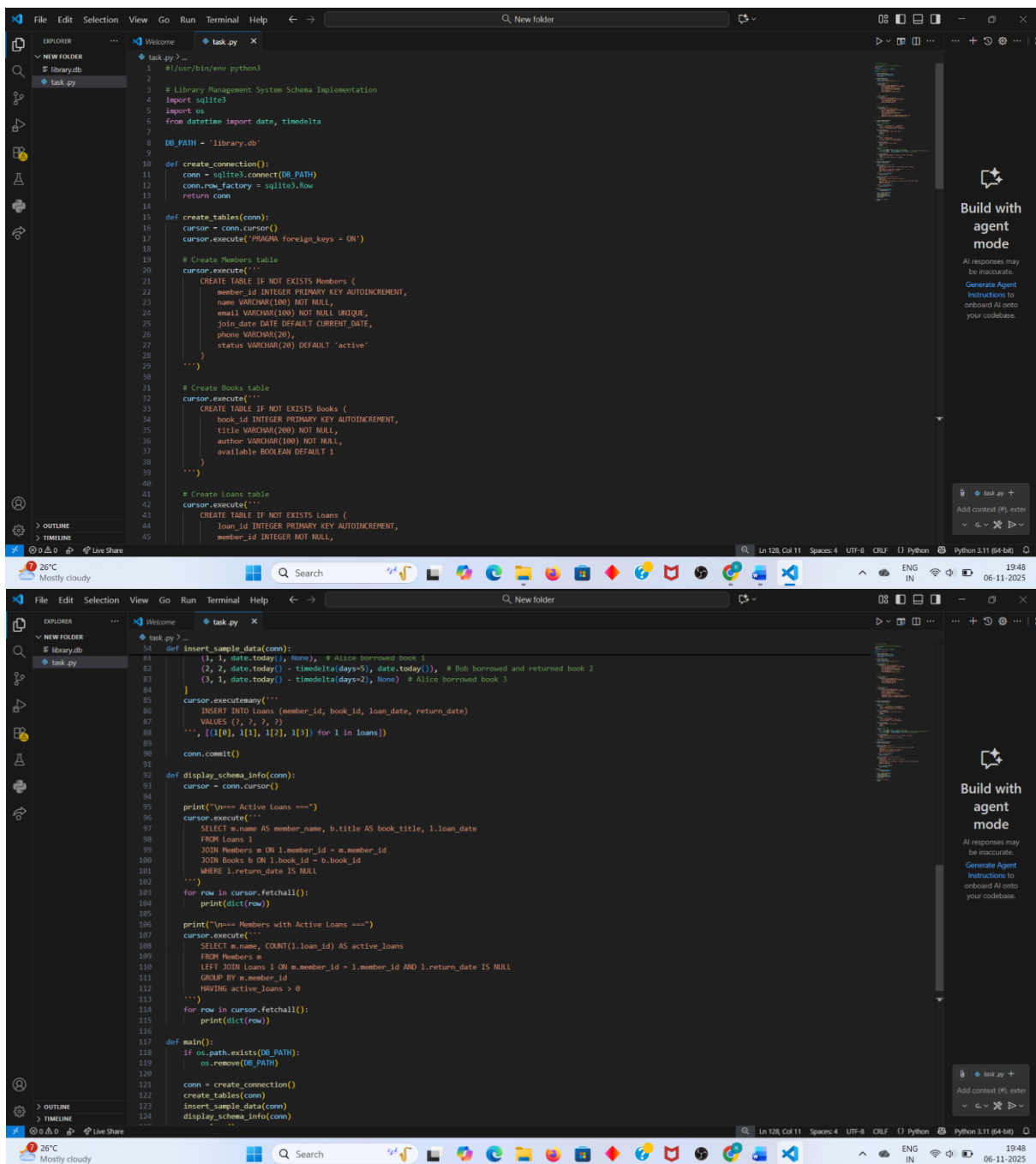
SQL Code

```
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 displays two screenshots of a Visual Studio Code editor window, showing the implementation of a Library Management System database schema and data insertion.

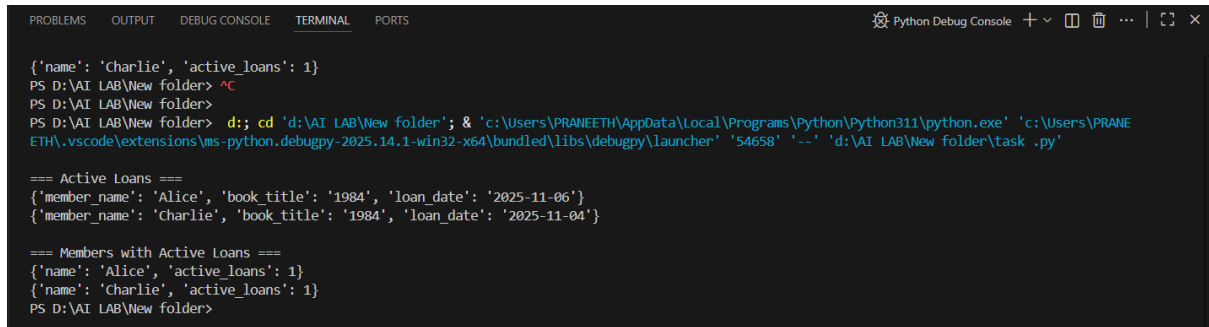
Top Screenshot: The code defines the database connection and table creation functions.

```
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,
```

Bottom Screenshot: The code continues with data insertion and schema display functions.

```
54 def insert_sample_data(conn):
55     cursor = conn.cursor()
56     cursor.execute("""
57     INSERT INTO Loans (member_id, book_id, loan_date, return_date)
58     VALUES (1, 1, date.today(), None), # Alice borrowed book 1
59            (2, 2, date.today() - timedelta(days=5), date.today()), # Bob borrowed and returned book 2
60            (3, 1, date.today() - timedelta(days=2), None) # Alice borrowed book 3
61     """)
62     conn.commit()
63
64 def display_schema_info(conn):
65     cursor = conn.cursor()
66
67     print("\n=== Active Loans ===")
68     cursor.execute("""
69     SELECT m.name AS member_name, b.title AS book_title, l.loan_date
70     FROM Loans l
71     JOIN Members m ON l.member_id = m.member_id
72     JOIN Books b ON l.book_id = b.book_id
73     WHERE l.return_date IS NULL
74     """)
75     for row in cursor.fetchall():
76         print(dict(row))
77
78     print("\n=== Members with Active Loans ===")
79     cursor.execute("""
80     SELECT m.name, COUNT(l.loan_id) AS active_loans
81     FROM Members m
82     LEFT JOIN Loans l ON m.member_id = l.member_id AND l.return_date IS NULL
83     GROUP BY m.member_id
84     HAVING active_loans > 0
85     """)
86     for row in cursor.fetchall():
87         print(dict(row))
88
89 def main():
90     if os.path.exists(DB_PATH):
91         os.remove(DB_PATH)
92
93     conn = create_connection()
94     create_tables(conn)
95     insert_sample_data(conn)
96     display_schema_info(conn)
97
98 if __name__ == '__main__':
99     main()
```

OUTPUT :



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console + - [ ] [x] ... | [ ] [x] X

{'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\bundle\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 :

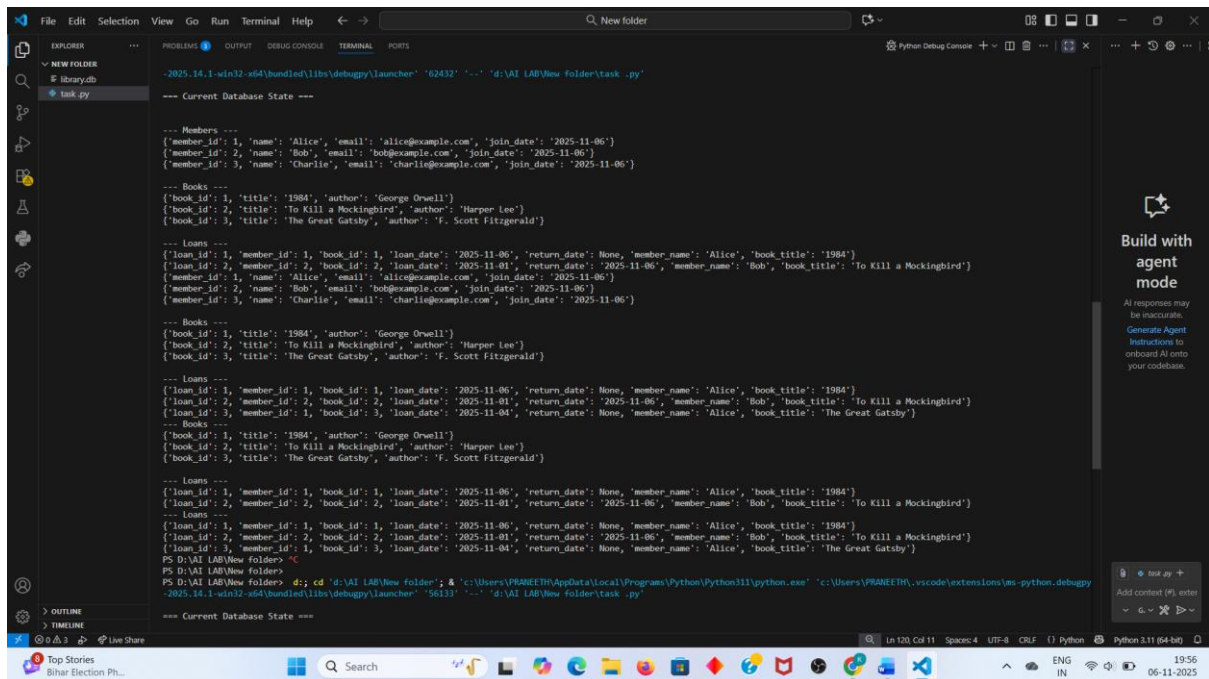
Generate SQL INSERT INTO statements with 3 sample records for each table (Books, Members, and Loans)

CODE GENERATED :

```
1 #!/usr/bin/env python3
2
3 # Library Management System - Sample Data Insertion with Error Handling
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 # Enable row factory for named columns
13     return conn
14
15 def setup_schema(conn):
16     """Set up the basic schema if it doesn't exist"""
17     cur = conn.cursor()
18     cur.execute("PRAGMA foreign_keys = ON")
19
20     # Create Members table
21     cur.execute("""
22     CREATE TABLE IF NOT EXISTS Members (
23         member_id INTEGER PRIMARY KEY AUTOINCREMENT,
24         name VARCHAR(100) NOT NULL,
25         email VARCHAR(100) NOT NULL UNIQUE,
26         join_date DATE DEFAULT CURRENT_DATE
27     )
28     """)
29
30     # Create Books Table
31     cur.execute("""
32     CREATE TABLE IF NOT EXISTS Books (
33         book_id INTEGER PRIMARY KEY AUTOINCREMENT,
34         title VARCHAR(100) NOT NULL,
35         author VARCHAR(100)
36     )
37     """)
38
39     # Create Loans Table
40     cur.execute("""
41     CREATE TABLE IF NOT EXISTS Loans (
42         loan_id INTEGER PRIMARY KEY AUTOINCREMENT,
43         member_id INTEGER NOT NULL,
44         book_id INTEGER NOT NULL,
45         loan_date DATE DEFAULT CURRENT_DATE,
```

```
52 def insert_sample_data(conn):
53     cur = conn.cursor()
54     cur.execute("""
55     INSERT INTO Loans (member_id, book_id, loan_date, return_date)
56     VALUES (?, ?, ?, ?)
57     """, loans)
58
59     conn.commit()
60
61 except sqlite3.IntegrityError as e:
62     print("Error inserting data:", e)
63
64 def display_data(conn):
65     print("\n== Current Database State ==\n")
66     cur = conn.cursor()
67
68     print("\n--- Members ---")
69     cur.execute("SELECT * FROM Members")
70     for row in cur.fetchall():
71         print(dict(row))
72
73     print("\n--- Books ---")
74     cur.execute("SELECT * FROM Books")
75     for row in cur.fetchall():
76         print(dict(row))
77
78     print("\n--- Loans ---")
79     cur.execute("""
80     SELECT l.*, m.name AS member_name, b.title AS book_title
81     FROM Loans l
82     JOIN Members m ON l.member_id = m.member_id
83     JOIN Books b ON l.book_id = b.book_id
84     """)
85     for row in cur.fetchall():
86         print(dict(row))
87
88 def main():
89     # Start fresh by removing existing database
90     if os.path.exists(DB_PATH):
91         os.remove(DB_PATH)
92
93     conn = create_connection()
94     setup_schema(conn)
95     insert_sample_data(conn)
96     display_data(conn)
97     conn.close()
98
99 if __name__ == "__main__":
100     main()
```

OUTPUT :



```
2025.14.1-win32-x64\handed\libs\debuggy\launcher" 62432' ... 'd:\AI LAB\New folder\task .py'

--- Current Database State ---

--- Members ---
{'member_id': 1, 'name': 'Alice', 'email': 'alice@example.com', 'join_date': '2025-11-06'}
{'member_id': 2, 'name': 'Bob', 'email': 'bob@example.com', 'join_date': '2025-11-06'}
{'member_id': 3, 'name': 'Charlie', 'email': 'charlie@example.com', 'join_date': '2025-11-06'}

--- Books ---
({'book_id': 1, 'title': '1984', 'author': 'George Orwell'})
({'book_id': 2, 'title': 'To Kill a Mockingbird', 'author': 'Harper Lee'})
({'book_id': 3, 'title': 'The Great Gatsby', 'author': 'F. Scott Fitzgerald'})

--- Loans ---
({'loan_id': 1, 'member_id': 1, 'book_id': 1, 'loan_date': '2025-11-06', 'return_date': None, 'member_name': 'Alice', 'book_title': '1984'})
({'loan_id': 2, 'member_id': 2, 'book_id': 2, 'loan_date': '2025-11-01', 'return_date': '2025-11-06', 'member_name': 'Bob', 'book_title': 'To Kill a Mockingbird'})
({'member_id': 1, 'name': 'Alice', 'email': 'alice@example.com', 'join_date': '2025-11-06'})
({'member_id': 2, 'name': 'Bob', 'email': 'bob@example.com', 'join_date': '2025-11-06'})
({'member_id': 3, 'name': 'Charlie', 'email': 'charlie@example.com', 'join_date': '2025-11-06'})

--- Books ---
({'book_id': 1, 'title': '1984', 'author': 'George Orwell'})
({'book_id': 2, 'title': 'To Kill a Mockingbird', 'author': 'Harper Lee'})
({'book_id': 3, 'title': 'The Great Gatsby', 'author': 'F. Scott Fitzgerald'})

--- Loans ---
({'loan_id': 1, 'member_id': 1, 'book_id': 1, 'loan_date': '2025-11-06', 'return_date': None, 'member_name': 'Alice', 'book_title': '1984'})
({'loan_id': 2, 'member_id': 2, 'book_id': 2, 'loan_date': '2025-11-01', 'return_date': '2025-11-06', 'member_name': 'Bob', 'book_title': 'To Kill a Mockingbird'})
({'loan_id': 3, 'member_id': 1, 'book_id': 3, 'loan_date': '2025-11-04', 'return_date': None, 'member_name': 'Alice', 'book_title': 'The Great Gatsby'})

--- Books ---
({'book_id': 1, 'title': '1984', 'author': 'George Orwell'})
({'book_id': 2, 'title': 'To Kill a Mockingbird', 'author': 'Harper Lee'})
({'book_id': 3, 'title': 'The Great Gatsby', 'author': 'F. Scott Fitzgerald'})

--- Loans ---
({'loan_id': 1, 'member_id': 1, 'book_id': 1, 'loan_date': '2025-11-06', 'return_date': None, 'member_name': 'Alice', 'book_title': '1984'})
({'loan_id': 2, 'member_id': 2, 'book_id': 2, 'loan_date': '2025-11-01', 'return_date': '2025-11-06', 'member_name': 'Bob', 'book_title': 'To Kill a Mockingbird'})
({'loan_id': 3, 'member_id': 1, 'book_id': 3, 'loan_date': '2025-11-04', 'return_date': None, 'member_name': 'Alice', 'book_title': 'The Great Gatsby'})

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\handed\libs\debuggy\launcher' 56133' ... 'd:\AI LAB\New folder\task .py'

--- Current Database State ---
```

OBSERVATION :

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

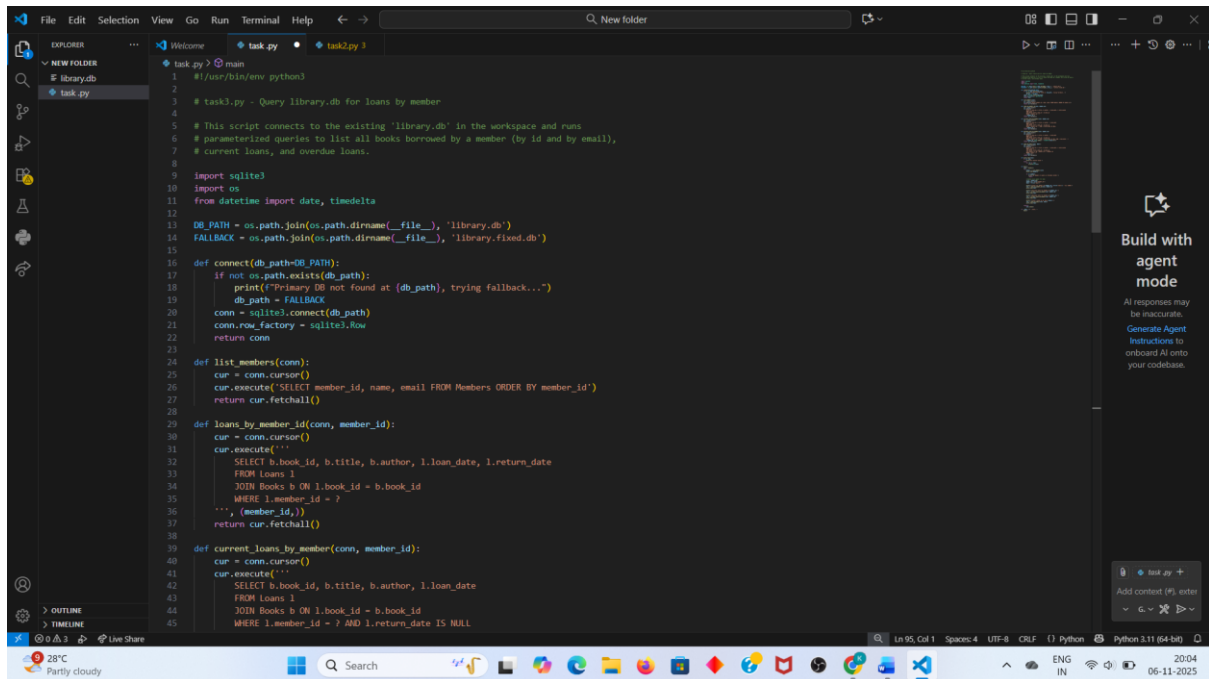
TASK DESCRIPTION 3 :

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

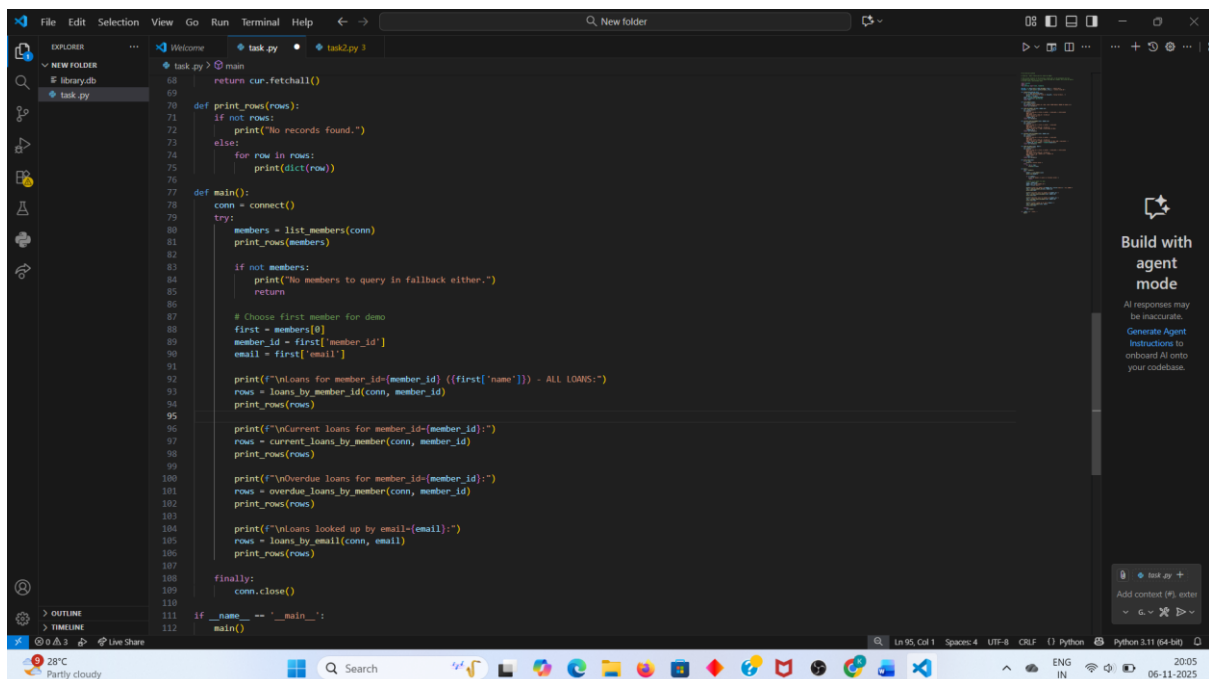
PROMPT :

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

CODE GENERATED :

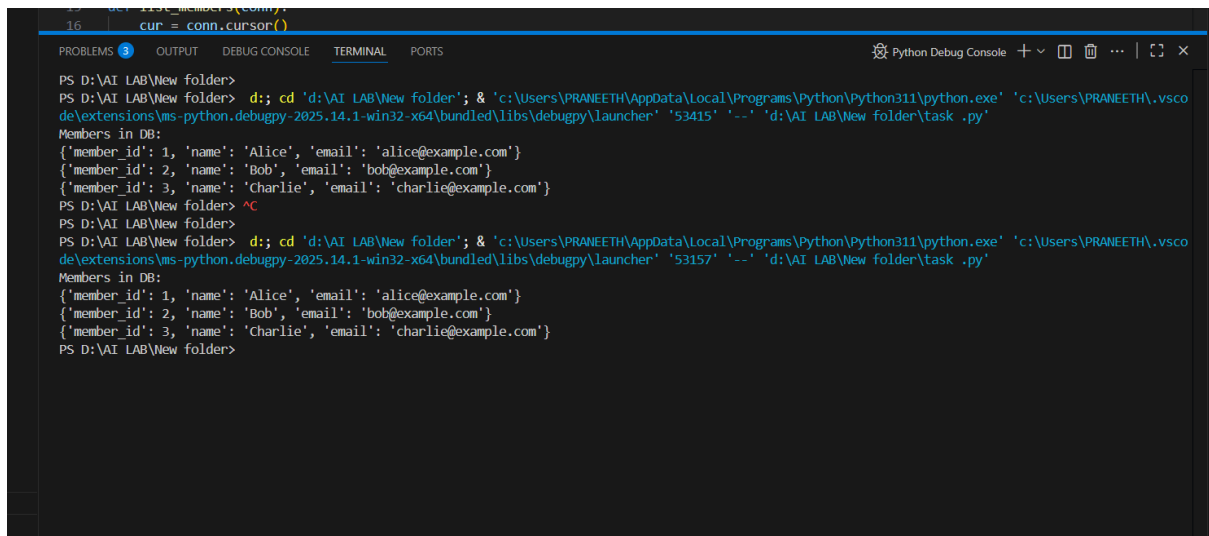


```
1 # task1.py - Query library.db for loans by member
2
3 # This script connects to the existing 'library.db' in the workspace and runs
4 # parameterized queries to list all books borrowed by a member (by id and by email),
5 # current loans, and overdue loans.
6
7 import sqlite3
8 import os
9 from datetime import date, timedelta
10
11 DB_PATH = os.path.join(os.path.dirname(__file__), 'library.db')
12 FALLBACK = os.path.join(os.path.dirname(__file__), 'library.fixed.db')
13
14 def connect(db_path=DB_PATH):
15     if not os.path.exists(db_path):
16         print(f"Primary DB not found at {db_path}, trying fallback...")
17         db_path = FALLBACK
18     conn = sqlite3.connect(db_path)
19     conn.row_factory = sqlite3.Row
20     return conn
21
22 def list_members(conn):
23     cur = conn.cursor()
24     cur.execute("SELECT member_id, name, email FROM Members ORDER BY member_id")
25     return cur.fetchall()
26
27 def loans_by_member_id(conn, member_id):
28     cur = conn.cursor()
29     cur.execute("""
30         SELECT b.book_id, b.title, b.author, l.loan_date, l.return_date
31         FROM Loans l
32         JOIN Books b ON l.book_id = b.book_id
33         WHERE l.member_id = ?
34     """, (member_id,))
35     return cur.fetchall()
36
37 def current_loans_by_member(conn, member_id):
38     cur = conn.cursor()
39     cur.execute("""
40         SELECT b.book_id, b.title, b.author, l.loan_date
41         FROM Loans l
42         JOIN Books b ON l.book_id = b.book_id
43         WHERE l.member_id = ? AND l.return_date IS NULL
44     """, (member_id,))
45     return cur.fetchall()
```



```
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(f"\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(f"\nCurrent loans for member_id={member_id}:")
97         rows = current_loans_by_member(conn, member_id)
98         print_rows(rows)
99
100         print(f"\nOverdue loans for member_id={member_id}:")
101         rows = overdue_loans_by_member(conn, member_id)
102         print_rows(rows)
103
104         print(f"\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()
```

OUTPUT :



```
16 cur = conn.cursor()

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\bundle\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>
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\bundle\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:

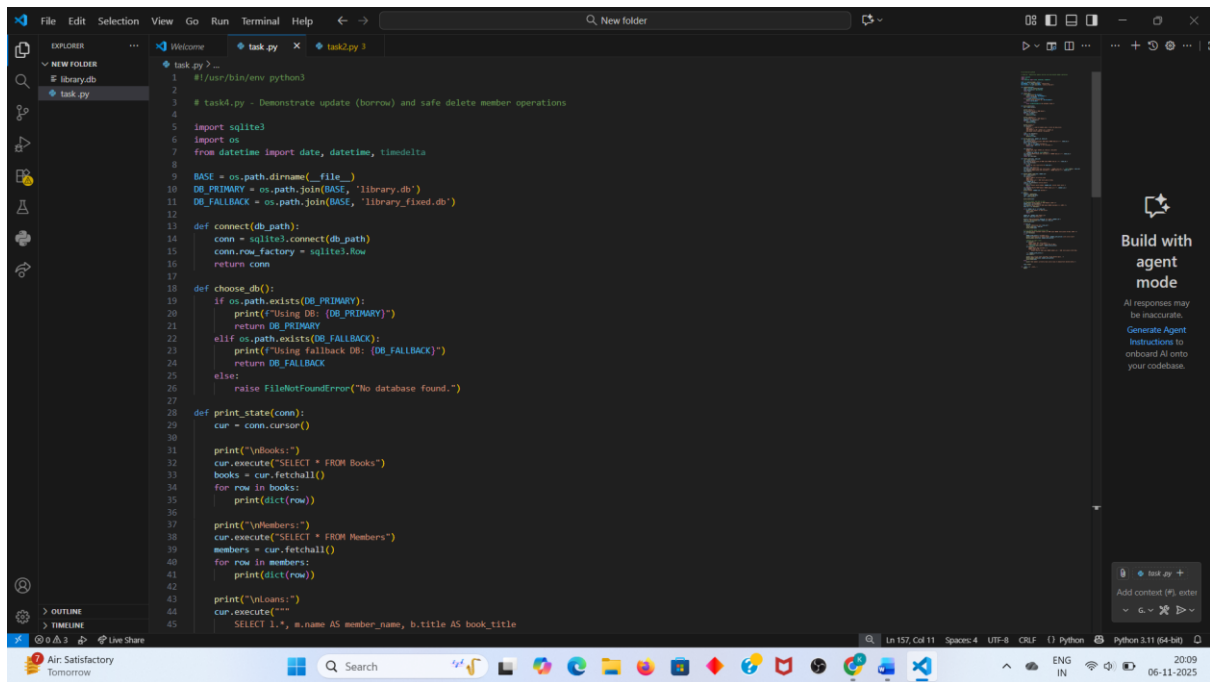
- Updating a book's availability to FALSE when borrowed.
- Deleting a member record safely.

PROMPT :

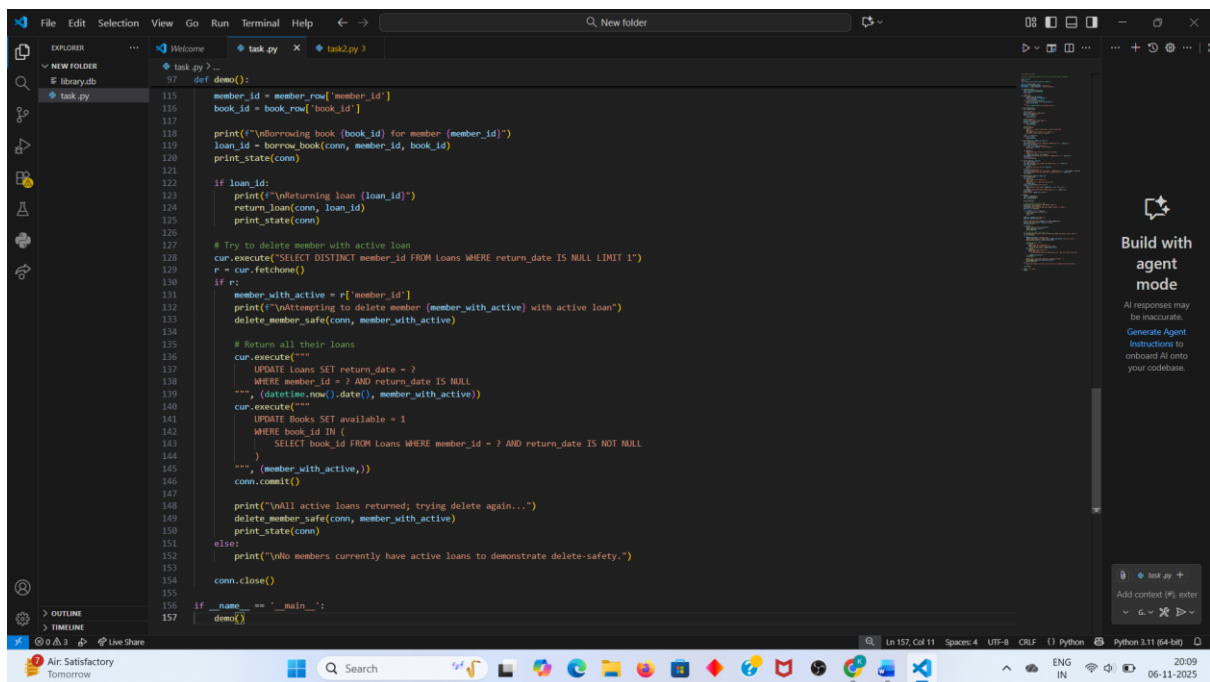
Generate SQL queries to:

- (a) Update a book's availability to FALSE when borrowed.
- (b) Delete a member record safely.

CODE GENERATED :

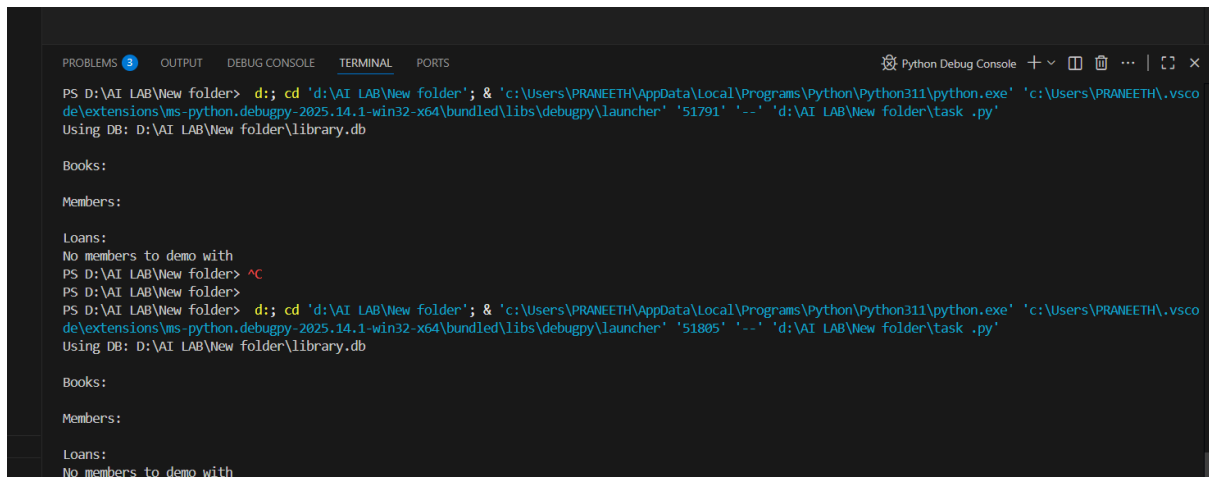


```
1 #!/usr/bin/env python3
2
3 # task4.py - Demonstrate update (borrow) and safe delete member operations
4
5 import sqlite3
6 import os
7 from datetime import date, datetime, timedelta
8
9 BASE = os.path.dirname(__file__)
10 DB_PRIMARY = os.path.join(BASE, 'library.db')
11 DB_FALLBACK = os.path.join(BASE, '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 l.*, 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_state(conn)
104
105     if loan_id:
106         print(f"\nreturning loan {loan_id}")
107         return_loan(conn, loan_id)
108         print_state(conn)
109
110     # Try to delete member with active loan
111     cur.execute("SELECT DISTINCT member_id FROM Loans WHERE return_date IS NULL LIMIT 1")
112     r = cur.fetchone()
113     if r:
114         member_with_active = r['member_id']
115         print(f"\nAttempting to delete member {member_with_active} with active loan")
116         delete_member_safe(conn, member_with_active)
117
118     # Return all their loans
119     cur.execute("""
120         UPDATE Loans SET return_date = ?
121         WHERE member_id = ? AND return_date IS NULL
122         """, (datetime.now().date(), member_with_active))
123     cur.execute("""
124         UPDATE Books SET available = 1
125         WHERE book_id IN (
126             SELECT book_id FROM Loans WHERE member_id = ? AND return_date IS NOT NULL
127         )
128         """, (member_with_active,))
129     conn.commit()
130
131     print("\nAll active loans returned; trying delete again...")
132     delete_member_safe(conn, member_with_active)
133     print_state(conn)
134
135     print("\nNo members currently have active loans to demonstrate delete-safety.")
136
137 conn.close()
138
139 if __name__ == '__main__':
140     demo()
```

OUTPUT :



```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS Python Debug Console + - [ ] [x] ... | [ ] [x] X

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\bundle\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\bundle\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.