

AI ASSISTED CODING LAB

ASSIGNMENT-16.4

NAME: K. Jashuva

ENROLLMENT NO:2503A51L16

BATCH NO: 19

TASK1

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 :

```
task1.py  X
ASSIGNMENT_16.4 > task1.py > ...
1  #!/usr/bin/env python3
2
3  # Library Management System Schema Implementation
4 > import sqlite3...
5
6  DB_PATH = 'library.db'
7
8  def create_connection():
9      conn = sqlite3.connect(DB_PATH)
10     conn.row_factory = sqlite3.Row
11     return conn
12
13 > def create_tables(conn): ...
14
15 > def insert_sample_data(conn):
16     cursor = conn.cursor()
17
18     # Insert sample members
19     members = [...]
20     cursor.executemany('INSERT INTO Members (name, email, phone) VALUES (?, ?, ?)', members)
21
22     # Insert sample books
23     books = [...]
24     cursor.executemany('...', ...)
25
26     # Insert a sample loan
27     loan_date = date.today()
28     due_date = loan_date + timedelta(days=14)
29     cursor.execute('...
30         INSERT INTO Loans (member_id, book_id, loan_date, due_date)
31         VALUES (1, 1, ?, ?)
32     ...', (loan_date.isoformat(), due_date.isoformat()))
33
34     conn.commit()
35
36 > def display_schema_info(conn): ...
37
38 > def main(): ...
39
40 if __name__ == "__main__":
41     main()
```

OUTPUT :

```
PS C:\Users\khaja\Downloads\ai retake\ASSIGNMENT_16.4> & C:/Users/khaja/anaconda3/python.exe "c:/Users/khaja/Downloads/ai retake/ASSIGNMENT_16.4/task1.py"

== Current Books Status ==
{'book_id': 1, 'title': 'The Great Gatsby', 'author': 'F. Scott Fitzgerald', 'total_copies': 2, 'available_copies': 1}
{'book_id': 2, 'title': 'To Kill a Mockingbird', 'author': 'Harper Lee', 'total_copies': 3, 'available_copies': 3}

== Active Loans ==
{'loan_id': 1, 'name': 'John Doe', 'title': 'The Great Gatsby', 'loan_date': '2025-11-22', 'due_date': '2025-12-06'}

== Members with Active Loans ==
{'name': 'John Doe', 'active_loans': 1}
○ PS C:\Users\khaja\Downloads\ai retake\ASSIGNMENT_16.4>
```

OBSERVATION :

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

TASK2

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:

The screenshot shows a code editor window with a dark theme. The file is named 'task2.py'. The code is a Python script for a Library Management System. It uses the sqlite3 library to interact with a database named 'library.db'. The script defines functions to create a connection, set up the schema (including creating tables for Members, Books, and Loans), and execute SQL statements. The code includes comments explaining the purpose of certain sections. The terminal tab at the top shows the command 'ASSIGNMENT_16.4 > task2.py > ...'.

```
task2.py <input>
ASSIGNMENT_16.4 > task2.py > ...
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
19     # Enable foreign keys
20     cur.execute('PRAGMA foreign_keys = ON')
21
22     # Create Members table
23     cur.execute('''
24     CREATE TABLE IF NOT EXISTS Members (
25         member_id INTEGER PRIMARY KEY AUTOINCREMENT,
26         name VARCHAR(100) NOT NULL,
27         email VARCHAR(100) NOT NULL UNIQUE,
28         join_date DATE DEFAULT CURRENT_DATE
29     )
30     ''')
31
32     # Create Books table
33     cur.execute('''
34     CREATE TABLE IF NOT EXISTS Books (
35         book_id INTEGER PRIMARY KEY AUTOINCREMENT,
36         title VARCHAR(200) NOT NULL,
37         author VARCHAR(100),
38         available BOOLEAN DEFAULT TRUE
39     )
40     ''')
41
42     # Create Loans table
43     cur.execute('''
44     CREATE TABLE IF NOT EXISTS Loans (
45         loan_id INTEGER PRIMARY KEY AUTOINCREMENT,
46         member_id INTEGER,
47         book_id INTEGER,
48         loan_date DATE,
49         return_date DATE,
50         FOREIGN KEY (member_id) REFERENCES Members(member_id),
51         FOREIGN KEY (book_id) REFERENCES Books(book_id)
52     )
53     ''')
```

```

task2.py x
ASSIGNMENT_16.4 > task2.py > ...
 57     def insert_sample_data(conn):
110     ]
111
112     for member_id, book_id, loan_date, return_date in loans:
113         try:
114             cur.execute('''
115                 INSERT INTO Loans (member_id, book_id, loan_date, return_date)
116                 VALUES (?, ?, ?, ?)
117                 ''', (member_id, book_id, loan_date, return_date))
118             print("Successfully created loan: Member {member_id}, Book {book_id}")
119         except sqlite3.IntegrityError as e:
120             print(f"Error creating loan (Member {member_id}, Book {book_id}): {str(e)}")
121         except sqlite3.Error as e:
122             print(f"Database error for loan (Member {member_id}, Book {book_id}): {str(e)}")
123
124     def display_data(conn):
125         """Display all inserted data"""
126         cur = conn.cursor()
127
128         print("\n--- Current Database State ---")
129
130         print("\n--- Members ---")
131         cur.execute('SELECT * FROM Members')
132         for row in cur.fetchall():
133             print(dict(row))
134
135         print("\n--- Books ---")
136         cur.execute('SELECT * FROM Books')
137         for row in cur.fetchall():
138             print(dict(row))
139
140         print("\n--- Loans ---")
141         cur.execute('''
142             SELECT l.*, m.name as member_name, b.title as book_title
143             FROM Loans l
144             JOIN Members m ON l.member_id = m.member_id
145             JOIN Books b ON l.book_id = b.book_id
146             ''')
147         for row in cur.fetchall():
148             print(dict(row))
149
150     def main():
151         # Start fresh by removing existing database
152         if os.path.exists(DB_PATH):
153             os.remove(DB_PATH)
154
155         # Create new database and insert sample data
156         conn = create_connection()
157         setup_schema(conn)
158         insert_sample_data(conn)
159         display_data(conn)
160         conn.close()
161
162     if __name__ == "__main__":
163         main()

```

OUTPUT :

```

PS C:\Users\khaja\Downloads\ai retake\ASSIGNMENT_16.4> & C:/Users/khaja/anaconda3/python.exe "c:/Users/khaja/Downloads/ai retake/ASSIGNMENT_16.4/ASSIGNMENT_16.4/task2.py"
*** Inserting Sample Data ***
--- Members Table Insertions ---
Successfully inserted member: Alice Johnson
Successfully inserted member: Bob Wilson
Successfully inserted member: Carol Smith
Error inserting member David Brown: UNIQUE constraint failed: Members.email
Error inserting member None: NOT NULL constraint failed: Members.name

--- Books Table Insertions ---
Successfully inserted book: The Great Adventure
Successfully inserted book: Mystery House
Successfully inserted book: Code Masters
Error inserting book None: NOT NULL constraint failed: Books.title
Successfully inserted book:

--- Loans Table Insertions ---
c:\Users\khaja\Downloads\ai retake\ASSIGNMENT_16.4\ASSIGNMENT_16.4\task2.py:114: DeprecationWarning: The default date adapter is deprecated as of Python 3.12; see the sqlite3 documentation for suggested replacement recipes
  cur.execute('''
Successfully created loan: Member 1, Book 1
Successfully created loan: Member 2, Book 2
Successfully created loan: Member 3, Book 3
Error creating loan (Member 99, Book 1): FOREIGN KEY constraint failed
Error creating loan (Member 1, Book 99): FOREIGN KEY constraint failed
Successfully created loan: Member 1, Book 1

--- Current Database State ---

--- Members ---
{'member_id': 1, 'name': 'Alice Johnson', 'email': 'alice@email.com', 'join_date': '2025-11-22'}
{'member_id': 2, 'name': 'Bob Wilson', 'email': 'bob@email.com', 'join_date': '2025-11-22'}
{'member_id': 3, 'name': 'Carol Smith', 'email': 'carol@email.com', 'join_date': '2025-11-22'}

--- Books ---
{'book_id': 1, 'title': 'The Great Adventure', 'author': 'John Author', 'available': 1}
{'book_id': 2, 'title': 'Mystery House', 'author': 'Sarah Writer', 'available': 1}
{'book_id': 3, 'title': 'Code Masters', 'author': 'Tech Team', 'available': 1}
{'book_id': 4, 'title': '', 'author': 'Empty Title Author', 'available': 1}

--- Loans ---
{'loan_id': 1, 'member_id': 1, 'book_id': 1, 'loan_date': '2025-11-22', 'return_date': None, 'member_name': 'Alice Johnson', 'book_title': 'The Great Adventure'}
{'loan_id': 2, 'member_id': 2, 'book_id': 2, 'loan_date': '2025-11-22', 'return_date': None, 'member_name': 'Bob Wilson', 'book_title': 'Mystery House'}
{'loan_id': 3, 'member_id': 3, 'book_id': 3, 'loan_date': '2025-11-22', 'return_date': None, 'member_name': 'Carol Smith', 'book_title': 'Code Masters'}
{'loan_id': 4, 'member_id': 1, 'loan_date': '2025-11-22', 'return_date': '2025-11-21', 'member_name': 'Alice Johnson', 'book_title': 'The Great Adventure'}
PS C:\Users\khaja\Downloads\ai retake\ASSIGNMENT_16.4>

```

OBSERVATION :

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

TASK3

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 :



```
task3.py
ASSIGNMENT_164 > task3.py > ...
1  #!/usr/bin/env python3
2
3 """
4 task3.py - Query library.db for loans by member
5
6 This script connects to the existing `library.db` in the workspace and runs
7 parameterized queries to list all books borrowed by a member (by id and by email),
8 current loans, and overdue loans.
9 """
10
11 import sqlite3
12 import os
13 from datetime import date
14
15 DB_PATH = os.path.join(os.path.dirname(__file__), 'library.db')
16 FALBACK_DB = os.path.join(os.path.dirname(__file__), 'library_fixed.db')
17
18
19 def connect(db_path=DB_PATH):
20     conn = sqlite3.connect(db_path)
21     conn.row_factory = sqlite3.Row
22     return conn
23
24
25 def list_members(conn):
26     cur = conn.cursor()
27     cur.execute('SELECT member_id, name, email FROM Members ORDER BY member_id')
28     return cur.fetchall()
29
30
31 def loans_by_member_id(conn, member_id):
32     cur = conn.cursor()
33     cur.execute('''
34         SELECT b.book_id, b.title, b.author, l.loan_date, l.due_date, l.return_date, l.status
35         FROM Loans l
36         JOIN Books b ON l.book_id = b.book_id
37         WHERE l.member_id = ?
38         ORDER BY l.loan_date DESC
39     ''', (member_id,))
40     return cur.fetchall()
41
42
43 def loans_by_email(conn, email):
44     cur = conn.cursor()
45     cur.execute('''
46         SELECT b.book_id, b.title, b.author, l.loan_date, l.due_date, l.return_date, l.status
47         FROM Loans l
48         JOIN Books b ON l.book_id = b.book_id
49         JOIN Members m ON l.member_id = m.member_id
50         WHERE m.email = ?
51         ORDER BY l.loan_date DESC
52     ''', (email,))
53     return cur.fetchall()
54
```

```

task3.py  X
ASSIGNMENT_16.4 > task3.py > ...
88     def main():
89         if not os.path.exists(db_to_use):
90             print('Primary DB not found at', db_to_use)
91             db_to_use = FALLBACK_DB
92
93
94     conn = connect(db_to_use)
95     try:
96         members = list_members(conn)
97         print('\nMembers in DB:')
98         print_rows(members)
99
100    if not members:
101        # Try fallback DB if different
102        if db_to_use != FALLBACK_DB and os.path.exists(FALLBACK_DB):
103            print('\nNo members found in primary DB, trying fallback DB:', FALLBACK_DB)
104            conn.close()
105            conn = connect(FALLBACK_DB)
106            members = list_members(conn)
107            print('\nMembers in fallback DB:')
108            print_rows(members)
109            if not members:
110                print('No members to query in fallback either.')
111                return
112        else:
113            print('No members to query.')
114            return
115
116    # choose first member for demo
117    first = members[0]
118    member_id = first['member_id']
119    email = first['email']
120
121    print(f"\nLoans for member_id={member_id} ({first['name']}): ALL LOANS:")
122    rows = loans_by_member_id(conn, member_id)
123    print_rows(rows)
124
125    print(f"\nCurrent loans for member_id={member_id}:")
126    rows = current_loans_by_member(conn, member_id)
127    print_rows(rows)
128
129    print(f"\nOverdue loans for member_id={member_id}:")
130    rows = overdue_loans_by_member(conn, member_id)
131    print_rows(rows)
132
133    print(f"\nLoans looked up by email={email}:")
134    rows = loans_by_email(conn, email)
135    print_rows(rows)
136
137    finally:
138        conn.close()
139
140
141    if __name__ == '__main__':
142        main()
143

```

OUTPUT :

```

PS C:\Users\khaja\Downloads\ai_retake\ASSIGNMENT_16.4> & C:/Users/khaja/anaconda3/python.exe "c:/Users/khaja/Downloads/ai_retake/ASSIGNMENT_16.4/ASSIGNMENT_16.4/task3.py"
Members in DB:
  (no rows)
No members to query.
PS C:\Users\khaja\Downloads\ai_retake\ASSIGNMENT_16.4>

```

OBSERVATION :

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

TASK4

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 :

```
task4.py < /dev/null > task4.py > ...
```

```
1  #!/usr/bin/env python3
2
3 """
4 task4.py - Demonstrate update (borrow) and safe delete member operations
5
6 This script connects to `library.db` (falls back to `library_fixed.db` if needed)
7 and demonstrates:
8 - borrow_book(member_id, book_id): checks availability, decrements available_copies, inserts a Loan
9 - return_loan(loan_id): sets return_date and increments available_copies
10 - delete_member_safe(member_id): only deletes a member if they have no active loans
11
12 It prints state before and after each operation.
13 """
14
15 import sqlite3
16 import os
17 from datetime import date, timedelta
18
19 BASE = os.path.dirname(__file__)
20 DB_PRIMARY = os.path.join(BASE, 'library.db')
21 DB_FALLBACK = os.path.join(BASE, 'library_fixed.db')
22
23
24 def connect(db_path):
25     conn = sqlite3.connect(db_path)
26     conn.row_factory = sqlite3.Row
27     return conn
28
29
30 def choose_db():
31     if os.path.exists(DB_PRIMARY):
32         return DB_PRIMARY
33     if os.path.exists(DB_FALLBACK):
34         return DB_FALLBACK
35     raise FileNotFoundError('No database found')
36
37
38 def print_state(conn):
39     cur = conn.cursor()
40     # detect schema variants
41     cur.execute("PRAGMA table_info(Books)")
42     books_cols = [r[1] for r in cur.fetchall()]
43     has_available_copies = 'available_copies' in books_cols
44     cur.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='BookCopies'")
45     has_bookcopies = cur.fetchone() is not None
46
47     print('\nBooks:')
48     if has_available_copies:
49         for r in cur.execute("SELECT book_id, title, total_copies, available_copies FROM Books"):
50             print(' ', dict(r))
51     elif has_bookcopies:
52         for r in cur.execute('''
53             SELECT b.book_id, b.title, COUNT(c.copy_id) as total_copies,
54             SUM(CASE WHEN c.available = 1 THEN 1 ELSE 0 END) as available_copies
55             FROM Books b
56             '''):
57             print(' ', dict(r))
58
59
60 def borrow_book(member_id, book_id):
61     conn = choose_db()
62     conn.row_factory = sqlite3.Row
63     cur = conn.cursor()
64     cur.execute("SELECT available_copies FROM Books WHERE book_id = ? AND available_copies > 0", (book_id,))
65     row = cur.fetchone()
66     if row is None:
67         print(f"Error: Book {book_id} not found or unavailable")
68         return
69     available_copies = row['available_copies']
70     cur.execute("UPDATE Books SET available_copies = ? WHERE book_id = ?", (available_copies - 1, book_id))
71     conn.commit()
72
73     loan_id = generate_loan_id()
74     cur.execute("INSERT INTO Loans (loan_id, member_id, book_id, return_date) VALUES (?, ?, ?, ?)", (loan_id, member_id, book_id, None))
75     conn.commit()
76
77     print(f"Loan {loan_id} issued for book {book_id} by member {member_id}. Available copies: {available_copies - 1}")
78
79
80 def return_loan(loan_id):
81     conn = choose_db()
82     conn.row_factory = sqlite3.Row
83     cur = conn.cursor()
84     cur.execute("SELECT available_copies FROM Books WHERE book_id = ? AND available_copies > 0", (loan['book_id'],))
85     row = cur.fetchone()
86     if row is None:
87         print(f"Error: Book {loan['book_id']} not found or unavailable")
88         return
89     available_copies = row['available_copies']
90     cur.execute("UPDATE Books SET available_copies = ? WHERE book_id = ?", (available_copies + 1, loan['book_id']))
91     conn.commit()
92
93     cur.execute("UPDATE Loans SET return_date = ? WHERE loan_id = ? AND member_id = ? AND book_id = ?",
94                (date.today(), loan_id, loan['member_id'], loan['book_id']))
95     conn.commit()
96
97     print(f"Loan {loan_id} returned for book {loan['book_id']} by member {loan['member_id']}. Available copies: {available_copies + 1}")
98
99
100 def delete_member(member_id):
101     conn = choose_db()
102     conn.row_factory = sqlite3.Row
103     cur = conn.cursor()
104     cur.execute("SELECT count(*) AS num_loans FROM Loans WHERE member_id = ?", (member_id,))
105     row = cur.fetchone()
106     if row['num_loans'] > 0:
107         print(f"Member {member_id} has {row['num_loans']} active loans and cannot be deleted")
108         return
109
110     cur.execute("DELETE FROM Members WHERE member_id = ?", (member_id,))
111     conn.commit()
112
113     print(f"Member {member_id} deleted successfully")
```

```

task4.py > 
ASSIGNMENT_16.4 > task4.py > ...
233 def demo():
234     print('Using DB: ', db)
235     conn = connect(db)
236     try:
237         print_state(conn)
238
239         # Find a member and a book that is available
240         cur = conn.cursor()
241         cur.execute('SELECT member_id FROM Members ORDER BY member_id LIMIT 1')
242         m = cur.fetchone()
243         if not m:
244             print('No members to demo with')
245             return
246         member_id = m['member_id']
247         # choose a book with available_copies > 0
248         cur.execute('SELECT book_id FROM Books WHERE available_copies > 0 ORDER BY book_id LIMIT 1')
249         b = cur.fetchone()
250         if not b:
251             print('No available books to borrow for demo')
252         else:
253             book_id = b['book_id']
254             loan_id = borrow_book(conn, member_id, book_id)
255             print_state(conn)
256             # return it
257             if loan_id:
258                 return_loan(conn, loan_id)
259             print_state(conn)
260
261         # Demonstrate delete failure: try to delete a member with active loan (if exists)
262         # find any member with active loan
263         cur.execute('SELECT DISTINCT member_id FROM Loans WHERE return_date IS NULL LIMIT 1')
264         r = cur.fetchone()
265         if r:
266             member_with_active = r['member_id']
267             delete_member_safe(conn, member_with_active)
268             # Now return all their loans and try again
269             cur.execute('UPDATE Loans SET return_date = ? WHERE member_id = ? AND return_date IS NULL', (date.today().isoformat(), member_with_active))
270             cur.execute('''
271                 UPDATE Books SET available_copies = available_copies + 1
272                 WHERE book_id IN (SELECT book_id FROM Loans WHERE member_id = ?)
273                 ''', (member_with_active,))
274             conn.commit()
275             print('All active loans for member returned; attempting delete again...')
276             delete_member_safe(conn, member_with_active)
277             print_state(conn)
278         else:
279             print('No members currently have active loans to demonstrate delete-safety.')
280
281     finally:
282         conn.close()
283
284
285     if __name__ == '__main__':
286         demo()
287
288

```

OUTPUT :

```

● PS C:\Users\khaja\Downloads\ai_retake\ASSIGNMENT_16.4> & C:/Users/khaja/anaconda3/python.exe "c:/Users/khaja/Downloads/ai_retake/ASSIGNMENT_16.4/ASSIGNMENT_16.4/task4.py"
Using DB: c:\Users\khaja\Downloads\ai_retake\ASSIGNMENT_16.4\ASSIGNMENT_16.4\library.db
Books:
Members:
Loans:
No members to demo with
○ PS C:\Users\khaja\Downloads\ai_retake\ASSIGNMENT_16.4>

```

OBSERVATION :

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