DATA 1202 - Data Analysis Tools Analytics

Assignment: Python SqLite

Student Name: Mustafa Khaja Masood Khaja

2. Report:

A. Sql query and corresponding output

Trigger Code:

#Step - 1 Create Logs Table

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS Logs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
   checkout_id INTEGER,
   timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (checkout_id) REFERENCES Checkouts (checkout_id)
)
''')
# Query to check if the 'Logs' table exists in the database
cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='Logs';")
table_exists = cursor.fetchone()

if table_exists:
   print("The 'Logs' table was created successfully.")
else:
   print("The 'Logs' table was not created.")
```

Outcome:

The 'Logs' table was created successfully.

#Step 2 Trigger - Using Python + SQLite (Direct SQL Queries with cursor.execute)

```
cursor.execute('''
CREATE TRIGGER log_checkout
AFTER INSERT ON Checkouts
```

```
BEGIN
    INSERT INTO Logs (checkout_id) VALUES (NEW.checkout_id);
END;
''')
# Commit table creation and trigger
conn.commit()
```

 # Query to check if the trigger exists

```
cursor.execute("SELECT name FROM sqlite_master WHERE type='trigger' AND
name='log_checkout';")
trigger_exists = cursor.fetchone()

if trigger_exists:
    print("The trigger 'log_checkout' already exists.")
else:
    print("The trigger 'log_checkout' does not exist.")
```

 # Insert a test checkout

```
cursor.execute("INSERT INTO Checkouts (user_id, book_id, checkout_date) VALUES (?, ?, ?)", (7, 9,
'2024-11-06'))
conn.commit()
```

 # Retrieve and display logs using SQLite and Python

```
cursor.execute("SELECT * FROM Logs")
logs_sql = cursor.fetchall()
print("All records in the Logs table (SQL Query):")
for log in logs_sql:
    print(log)
```

Output:
All records in the Logs table (SQL Query):

(1, 1, '2024-11-07 01:28:18')

(2, 2, '2024-11-07 01:28:18')

(3, 3, '2024-11-07 01:28:18')

(4, 4, '2024-11-07 01:28:18')

(5, 5, '2024-11-07 02:36:32')

(6, 6, '2024-11-07 02:43:57')

Find the title, authors and the isbn of the books that 'John Smith' has checked out.

#SQL Query

```
query = '''
SELECT b.title, b.author, b.isbn
FROM Books b
JOIN Checkouts c ON b.book_id = c.book_id
JOIN Users u ON c.user_id = u.user_id
WHERE u.full_name = 'John Smith';
'''
# Execute the query
cursor.execute(query)

# Fetch and print the results
books_checked_out_by_john_smith = cursor.fetchall()
print("Books checked out by John Smith:")
print(books_checked_out_by_john_smith)
```

Output:

Books checked out by John Smith:

[('My First SQL book', 'Mary Parker', 981483029127), ('My Second SQL book', 'John Mayer', 857300923713)]

 # SQL Query to find reviewers for "My Third SQL Book"

```
query_sql = '''
SELECT u.full_name
FROM Reviews r
JOIN Users u ON r.reviewer_name = u.full_name
JOIN Books b ON r.book_id = b.book_id
WHERE b.title = 'My Third SQL book';
'''

# Execute the query
cursor = conn.cursor()
cursor.execute(query_sql)

# Fetch and display the results
reviewers_sql = cursor.fetchall()
print("Reviewers for 'My Third SQL book' (SQL Query):")
for reviewer in reviewers_sql:
    print(reviewer[0])  # Print the full name of each reviewer
```

Output:
Reviewers for 'My Third SQL Book' (SQL Query):
Jane Smith

 # Find the users that have no books checked out.

```
# SQL Query to find users with no books checked out
query_sql = '''
SELECT u.full_name
FROM Users u
LEFT JOIN Checkouts c ON u.user_id = c.user_id
WHERE c.checkout_id IS NULL;
```

'''

Execute the query

cursor = conn.cursor()

cursor.execute(query_sql)

Fetch and display the results

users_no_checkout_sql = cursor.fetchall()

print("Users with no books checked out (SQL Query):")

for user in users_no_checkout_sql:

    print(user[0]) # Print the full name of each user

Output:

Users with no books checked out (SQL Query): Harry Potter

 # Query to show all records in the Logs table

 # Using SQLite to execute SQL queries

cursor.execute("SELECT * FROM Logs")

logs = cursor.fetchall()

print("All records in the Logs table (SQL Query):")

print(logs)

Output:

All records in the Logs table (SQL Query):

All records in the Logs table (SQL Query):

[(1, 5, '2024-11-07 18:13:06')]

B. Using pandas, sending the sql query to pandas function (read_sql_query) and grnerate the same output

Trigger:

 # Load Logs table data into a Pandas DataFrame

```
logs_df = pd.read_sql_query("SELECT * FROM Logs", conn)
print("All records in the Logs table (Pandas Query):")
print(logs_df)
```

Output:

```
All records in the Logs table (Pandas Query):
  id  checkout_id          timestamp
0  1          5  2024-11-07 18:13:06
```

---

1. Find the title, authors and the isbn of the books that 'John Smith' has checked out.

#Pandas Query

```
# Execute the query and store the result in a DataFrame
books_checked_out_by_john = pd.read_sql_query(query, conn)

# Display the results
print(books_checked_out_by_john)
```

Output:

```
title      author         isbn
0   My First SQL book  Mary Parker  981483029127
1  My Second SQL book   John Mayer  857300923713
```

---

2. Find all reviewers for the book "My Third SQL book"

```
# Query to find reviewers for "My Third SQL Book" using Pandas
reviews_df = pd.read_sql_query("SELECT * FROM Reviews", conn)
users_df = pd.read_sql_query("SELECT * FROM Users", conn)  # Ensure you have Users DataFrame
books_df = pd.read_sql_query("SELECT * FROM Books", conn)  # Ensure you have Books DataFrame

# Merge DataFrames and filter for the specific book title
reviewers_pandas = reviews_df.merge(users_df, left_on='reviewer_name', right_on='full_name') \
    .merge(books_df, on='book_id')
```

```
reviewers_pandas = reviewers_pandas[reviewers_pandas['title'] == 'My Third SQL book']
print("Reviewers for 'My Third SQL Book' (Pandas Query):")
print(reviewers_pandas[['full_name']])
```

Output:

```
Reviewers for 'My Third SQL Book' (Pandas Query):
   full_name
3  Jane Smith
```

---

3. Find the users that have no books checked out

```
# Pandas Query
users_df = pd.read_sql_query("SELECT * FROM Users", conn)
checkouts_df = pd.read_sql_query("SELECT * FROM Checkouts", conn)

# Identify users with checkouts
users_with_checkout = checkouts_df['user_id'].unique()

# Filter to find users with no checkouts
users_no_checkout_pandas = users_df[~users_df['user_id'].isin(users_with_checkout)]
print("Users with no books checked out (Pandas Query):")
print(users_no_checkout_pandas[['full_name']])
```

Output:

```
Users with no books checked out (Pandas Query):
     full_name
2  Harry Potter
```

---

4. Show all the records of the logs table

#pandas query

```
logs_df = pd.read_sql_query("SELECT * FROM Logs", conn)
```

```
print("All records in the Logs table (Pandas Query):")

print(logs_df)
```

Output:

```
All records in the Logs table (Pandas Query):

   id  checkout_id          timestamp

0   1            5  2024-11-07 18:13:06
```

C. Description of the operation of programs' module

The operation of my program module involves several steps interacting with an SQLite database and using Python with the sqlite3 and pandas libraries to perform tasks related to a library database system. Here's a breakdown:

1. **SQLite Database Connection:**
   - The program starts by connecting to the SQLite database (LibraryDatabase.db) and queries the database to check the existing tables.
2. **Database Table Creation:**
   - If the tables exist, they are dropped. Then, the program creates new tables (Users, Addresses, Books, Checkouts, Reviews, Logs) if they don't already exist. Each table is defined with specific columns and relationships, using foreign keys to link the tables.
3. **Inserting Data:**
   - Data is inserted into the tables: Users, Addresses, Books, Checkouts, and Reviews. This data is manually defined in the program and inserted using executemany() for bulk inserts.
4. **Logging and Triggers:**
   - The Logs table is created to track checkout operations. A trigger (log_checkout) is created to automatically insert a record into the Logs table after each new checkout operation.
   - After a test checkout is added, the program retrieves and displays all records from the Logs table.
5. **SQL Queries:**
   - SQL queries are executed to retrieve specific data:
     - Books checked out by 'John Smith'.
     - Reviewers for a specific book ('My Third SQL book').
     - Users who have not checked out any books.
     - Records in the Logs table.
6. **Using Pandas:**
   - The program uses pandas to load the Logs table data into a DataFrame with the pd.read_sql_query() method, providing an easy-to-read output of the records in the Logs table.

Screenshots of SQL Queries:

In [17]:
```
#SQL Queries

#Step - 1 Create Logs Table
cursor.execute('''
CREATE TABLE IF NOT EXISTS Logs (
     id INTEGER PRIMARY KEY AUTOINCREMENT,
    checkout_id INTEGER,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (checkout_id) REFERENCES Checkouts (checkout_id)
)
''')
# Query to check if the 'Logs' table exists in the database
cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='Logs';")
table_exists = cursor.fetchone()

if table_exists:
    print("The 'Logs' table was created successfully.")
else:
    print("The 'Logs' table was not created.")


#Step 2 Trigger - Using Python + SQLite (Direct SQL Queries with cursor.execute)
cursor.execute('''
CREATE TRIGGER log_checkout
AFTER INSERT ON Checkouts
BEGIN
    INSERT INTO Logs (checkout_id) VALUES (NEW.checkout_id);
END;
''')
# Commit table creation and trigger
conn.commit()


# Query to check if the trigger exists
cursor.execute("SELECT name FROM sqlite_master WHERE type='trigger' AND name='log_checkout';")
trigger_exists = cursor.fetchone()

if trigger_exists:
    print("The trigger 'log_checkout' already exists.")
else:
    print("The trigger 'log_checkout' does not exist.")


# Retrieve and display logs using SQLite and Python
cursor.execute("SELECT * FROM Logs")
logs_sql = cursor.fetchall()
print("All records in the Logs table (SQL Query):")
for log in logs_sql:
    print(log)
```

```
The 'Logs' table was created successfully.
The trigger 'log_checkout' already exists.
All records in the Logs table (SQL Query):
```

Fig.1. Trigger and Logs Table Query

```
In [19]: # Insert a test checkout
cursor.execute("INSERT INTO Checkouts (user_id, book_id, checkout_date) VALUES (?, ?, ?)", (7, 9, '2024-11-06'))
conn.commit()
```

```
In [21]: # Retrieve and display logs using SQLite and Python
cursor.execute("SELECT * FROM Logs")
logs_sql = cursor.fetchall()
print("All records in the Logs table (SQL Query):")
for log in logs_sql:
    print(log)
```

```
All records in the Logs table (SQL Query):
(1, 5, '2024-11-07 18:13:06')
```

Fig.2. Test Checkout

```
In [23]: #Find the title, authors and the isbn of the books that 'John Smith' has checked out.
#SQL Query
query = '''
SELECT b.title, b.author, b.isbn
FROM Books b
JOIN Checkouts c ON b.book_id = c.book_id
JOIN Users u ON c.user_id = u.user_id
WHERE u.full_name = 'John Smith';
'''
# Execute the query
cursor.execute(query)

# Fetch and print the results
books_checked_out_by_john_smith = cursor.fetchall()
print("Books checked out by John Smith:")
print(books_checked_out_by_john_smith)
```

```
Books checked out by John Smith:
[('My First SQL book', 'Mary Parker', 981483029127), ('My Second SQL book', 'John Mayer', 857300923713)]
```

Fig.3. Books John Smith has checked out

```
In [51]: #Find all reviewers for the book "My Third SQL book" (SQL query in Python)
query_sql = '''
SELECT u.full_name
FROM Reviews r
JOIN Users u ON r.reviewer_name = u.full_name
JOIN Books b ON r.book_id = b.book_id
WHERE b.title = 'My Third SQL book';
'''

# Execute the query using the cursor
cursor.execute(query_sql)

# Fetch and display the results
reviewers_sql = cursor.fetchall()
print("Reviewers for 'My Third SQL book' (SQL Query):")
for reviewer in reviewers_sql:
    print(reviewer[0])  # Print the full name of each reviewer
```

```
Reviewers for 'My Third SQL book' (SQL Query):
Jane Smith
```

Fig.4. Reviewers for the book "My Third SQL book"

In [53]:
```
# Find the users that have no books checked out.

# SQL Query to find users with no books checked out
query_sql = '''
SELECT u.full_name
FROM Users u
LEFT JOIN Checkouts c ON u.user_id = c.user_id
WHERE c.checkout_id IS NULL;
'''

# Execute the query
cursor = conn.cursor()
cursor.execute(query_sql)

# Fetch and display the results
users_no_checkout_sql = cursor.fetchall()
print("Users with no books checked out (SQL Query):")
for user in users_no_checkout_sql:
    print(user[0])  # Print the full name of each user
```

```
Users with no books checked out (SQL Query):
Harry Potter
```

Fig.5. Users who did not check out the books

In [55]:
```
# Query to show all records in the Logs table

# Using SQLite to execute SQL queries
cursor.execute("SELECT * FROM Logs")
logs = cursor.fetchall()
print("All records in the Logs table (SQL Query):")
print(logs)
```

```
All records in the Logs table (SQL Query):
[(1, 5, '2024-11-07 18:13:06')]
```

Fig.6. Records in Log table

Screenshots of Pandas Queries:

```
#1.Find the title, authors and the isbn of the books that 'John Smith' has checked out.
#Pandas Query
# Execute the query and store the result in a DataFrame
books_checked_out_by_john = pd.read_sql_query(query, conn)

# Display the results
print(books_checked_out_by_john)
```

```
                title       author          isbn
0   My First SQL book  Mary Parker  981483029127
1  My Second SQL book   John Mayer  857300923713
```

Fig.7. Books John Smith has checked out

```
In [67]: #2Find all reviewers for the book "My Third SQL book"
         # Query to find reviewers for "My Third SQL Book" using Pandas
         reviews_df = pd.read_sql_query("SELECT * FROM Reviews", conn)
         users_df = pd.read_sql_query("SELECT * FROM Users", conn)  # Ensure you have Users DataFrame
         books_df = pd.read_sql_query("SELECT * FROM Books", conn)  # Ensure you have Books DataFrame

         # Merge DataFrames and filter for the specific book title
         reviewers_pandas = reviews_df.merge(users_df, left_on='reviewer_name', right_on='full_name') \
             .merge(books_df, on='book_id')
         reviewers_pandas = reviewers_pandas[reviewers_pandas['title'] == 'My Third SQL book']
         print("Reviewers for 'My Third SQL Book' (Pandas Query):")
         print(reviewers_pandas[['full_name']])
```

```
Reviewers for 'My Third SQL Book' (Pandas Query):
    full_name
3  Jane Smith
```

Fig.8. Reviewers for the book "My Third SQL book"

```
In [69]: #3.Find the users that have no books checked out

         # Pandas Query
         users_df = pd.read_sql_query("SELECT * FROM Users", conn)
         checkouts_df = pd.read_sql_query("SELECT * FROM Checkouts", conn)

         # Identify users with checkouts
         users_with_checkout = checkouts_df['user_id'].unique()

         # Filter to find users with no checkouts
         users_no_checkout_pandas = users_df[~users_df['user_id'].isin(users_with_checkout)]
         print("Users with no books checked out (Pandas Query):")
         print(users_no_checkout_pandas[['full_name']])
```

```
Users with no books checked out (Pandas Query):
      full_name
2  Harry Potter
```

Fig.9. Users who did not check out the books

```
In [71]: #4.Show all the records of the logs table
         #pandas query
         logs_df = pd.read_sql_query("SELECT * FROM Logs", conn)
         print("All records in the Logs table (Pandas Query):")
         print(logs_df)
```

```
All records in the Logs table (Pandas Query):
   id  checkout_id            timestamp
0   1            5  2024-11-07 18:13:06
```

Fig.10. Records in Log table

APPENDIX

```
pip install pandas

Defaulting to user installation because normal site-packages is not writeable

Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\site-packages (2.2.2)

Requirement already satisfied: numpy>=1.26.0 in c:\programdata\anaconda3\lib\site-packages (from pandas) (1.26.4)

Requirement already satisfied: python-dateutil>=2.8.2 in c:\programdata\anaconda3\lib\site-packages (from pandas) (2.9.0.post0)

Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-packages (from pandas) (2024.1)

Requirement already satisfied: tzdata>=2022.7 in c:\programdata\anaconda3\lib\site-packages (from pandas) (2023.3)

Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)

Note: you may need to restart the kernel to use updated packages.

In [1]:

import pandas as pd

print(pd.__version__)

2.2.2

In [3]:

import sqlite3


# Connect to SQLite database

conn = sqlite3.connect('LibraryDatabase.db')

cursor = conn.cursor()

In [5]:

# Query to check for all tables in the database

cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")

tables = cursor.fetchall()


if tables:

    print("Tables in the database:")

    for table in tables:

        print(table[0])  # Print the table name

else:
```

```
    print("No tables found in the database.")
```

```
Tables in the database:
Users
Books
Reviews
Addresses
Logs
sqlite_sequence
Checkouts
```

In [11]:

```
# Drop tables if they already exist
cursor.execute("DROP TABLE IF EXISTS Users")
cursor.execute("DROP TABLE IF EXISTS Addresses")
cursor.execute("DROP TABLE IF EXISTS Books")
cursor.execute("DROP TABLE IF EXISTS Checkouts")
cursor.execute("DROP TABLE IF EXISTS Reviews")
cursor.execute("DROP TABLE IF EXISTS Logs")
print("Dropped existing tables if they existed")
conn.commit()
```

```
Dropped existing tables if they existed
```

In [13]:

```
# Create Users Table
cursor.execute('''
CREATE TABLE IF NOT EXISTS Users (
    user_id INTEGER PRIMARY KEY,
    full_name TEXT NOT NULL,
    enabled TEXT NOT NULL CHECK(enabled IN ('true', 'false')),
    last_login DATETIME
)
''')
print("Users table created successfully")



# Create Addresses Table
cursor.execute('''
```

```
CREATE TABLE IF NOT EXISTS Addresses (
    address_id INTEGER PRIMARY KEY,
    user_id INTEGER UNIQUE,
    street TEXT,
    city TEXT,
    state TEXT,
    FOREIGN KEY (user_id) REFERENCES Users (user_id)
)
''')
print("Addresses table created successfully")

# Create Books Table
cursor.execute('''
CREATE TABLE IF NOT EXISTS Books (
    book_id INTEGER PRIMARY KEY,
    title TEXT NOT NULL,
    author TEXT NOT NULL,
    isbn INTEGER NOT NULL, -- Changed to INTEGER
    published_date DATE -- Added column
)
''')
print("Books table created successfully")

# Create Checkouts Table
cursor.execute('''
CREATE TABLE IF NOT EXISTS Checkouts (
    checkout_id INTEGER PRIMARY KEY,
    user_id INTEGER,
    book_id INTEGER,
    checkout_date DATETIME,
    return_date DATETIME, -- Added column
    FOREIGN KEY (user_id) REFERENCES Users (user_id),
    FOREIGN KEY (book_id) REFERENCES Books (book_id)
)
''')
```

```
print("Checkouts table created successfully")
```

```
# Create Reviews Table
cursor.execute('''
CREATE TABLE IF NOT EXISTS Reviews (
    review_id INTEGER PRIMARY KEY,
    book_id INTEGER,
    reviewer_name TEXT, -- Changed from review_text
    content TEXT, -- Added column for review content
    rating INTEGER, -- Added column for rating
    published_date DATE, -- Added column for review date
    FOREIGN KEY (book_id) REFERENCES Books (book_id)
)
''')
print("Reviews table created successfully")
conn.commit()
```

```
Users table created successfully
Addresses table created successfully
Books table created successfully
Checkouts table created successfully
Reviews table created successfully
```

In [15]:

```
# Insert data into Users table
users_data = [
  ('John Smith', 'false', '2017-10-25 10:26:10.015152'),
    ('Alice Walker', 'true', '2017-10-25 10:26:50.295461'),
    ('Harry Potter', 'true', '2017-10-25 10:26:50.295461'),
    ('Jane Smith', 'true', '2017-10-25 10:36:43.324015')
]

cursor.executemany('''INSERT INTO Users (full_name, enabled, last_login) VALUES (?, ?, ?)
    ''', users_data)

conn.commit()
```

```
print("Inserted data into Users table successfully.")


# Insert data into Addresses table
addresses_data = [
   (1, 'Market Street', 'San Fransisco', 'CA'),
    (2, 'Elm Street', 'San Fransisco', 'CA'),
    (3, 'MainStreet', 'Boston', 'CA'),
]

cursor.executemany('''INSERT INTO Addresses (user_id, street, city, state) VALUES (?, ?, ?, ?)
''', addresses_data)

conn.commit()
print("Inserted data into Addresses table successfully.")

#Insert data into Books table
books_data = [
    ('My First SQL book', 'Mary Parker', 981483029127, '2012-02-22'),
     ('My Second SQL book', 'John Mayer', 857300923713, '1972-07-03'),
    ('My Third SQL book', 'Cary Flint', 523120967812, '2015-10-18')
]
cursor.executemany('''
    INSERT INTO Books (title, author, isbn, published_date) VALUES (?, ?, ?, ?)
''', books_data)

conn.commit()
print("Inserted data into Books table successfully.")

#Insert data into Checkouts table
checkouts_data = [
    (1, 1, '2017-10-15 14:43:18.095143-07', None),
    (1, 2, '2017-10-05 16:22:44.593188-07', '2017-10-13 13:05:12.673382-05'),
     (2, 2, '2017-10-15 11:11:24.994973-07', '2017-10-22 17:47:10.407569-07'),
    (4, 3, '2017-10-15 09:27:07.215217-07', None)
```

```
]



cursor.executemany('''
    INSERT INTO Checkouts (user_id, book_id, checkout_date, return_date) VALUES (?, ?, ?, ?)
''', checkouts_data)

conn.commit()
print("Inserted data into Checkouts table successfully.")



#Insert data into Reviews table
reviews_data = [
    (1, 'John Smith', 'My First Review', 4, '2017-12-10'),
    (2, 'John Smith', 'My Second Review', 5, '2017-10-13'),
    (2, 'Alice Walker', 'Another Review', 1, '2017-10-22'),
    (3, 'Jane Smith', 'My Third SQL book', 4, '2024-11-07')
]

cursor.executemany('''
    INSERT INTO Reviews (book_id, reviewer_name, content, rating, published_date)
    VALUES (?, ?, ?, ?, ?)
''', reviews_data)

conn.commit()
print("Inserted data into Reviews table successfully.")
Inserted data into Users table successfully.
Inserted data into Addresses table successfully.
Inserted data into Books table successfully.
Inserted data into Checkouts table successfully.
Inserted data into Reviews table successfully.
In [17]:
#SQL Queries

#Step - 1 Create Logs Table
```

```
cursor.execute('''
CREATE TABLE IF NOT EXISTS Logs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
   checkout_id INTEGER,
   timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (checkout_id) REFERENCES Checkouts (checkout_id)
)
''')
# Query to check if the 'Logs' table exists in the database
cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name='Logs';")
table_exists = cursor.fetchone()

if table_exists:
   print("The 'Logs' table was created successfully.")
else:
   print("The 'Logs' table was not created.")



#Step 2 Trigger - Using Python + SQLite (Direct SQL Queries with cursor.execute)
cursor.execute('''
CREATE TRIGGER log_checkout
AFTER INSERT ON Checkouts
BEGIN
   INSERT INTO Logs (checkout_id) VALUES (NEW.checkout_id);
END;
''')
# Commit table creation and trigger
conn.commit()



# Query to check if the trigger exists
cursor.execute("SELECT name FROM sqlite_master WHERE type='trigger' AND name='log_checkout';")
trigger_exists = cursor.fetchone()

if trigger_exists:
```

```
    print("The trigger 'log_checkout' already exists.")
else:
    print("The trigger 'log_checkout' does not exist.")
```

```
# Retrieve and display logs using SQLite and Python
cursor.execute("SELECT * FROM Logs")
logs_sql = cursor.fetchall()
print("All records in the Logs table (SQL Query):")
for log in logs_sql:
    print(log)
```

The 'Logs' table was created successfully.

The trigger 'log_checkout' already exists.

All records in the Logs table (SQL Query):

In [19]:

```
# Insert a test checkout
cursor.execute("INSERT INTO Checkouts (user_id, book_id, checkout_date) VALUES (?, ?, ?)", (7, 9, '2024-11-06'))
conn.commit()
```

In [21]:

```
# Retrieve and display logs using SQLite and Python
cursor.execute("SELECT * FROM Logs")
logs_sql = cursor.fetchall()
print("All records in the Logs table (SQL Query):")
for log in logs_sql:
    print(log)
```

All records in the Logs table (SQL Query):

(1, 5, '2024-11-07 18:13:06')

In [23]:

```
#Find the title, authors and the isbn of the books that 'John Smith' has checked out.
#SQL Query
query = '''
SELECT b.title, b.author, b.isbn
FROM Books b
JOIN Checkouts c ON b.book_id = c.book_id
JOIN Users u ON c.user_id = u.user_id
```

```
WHERE u.full_name = 'John Smith';
'''
# Execute the query
cursor.execute(query)

# Fetch and print the results
books_checked_out_by_john_smith = cursor.fetchall()
print("Books checked out by John Smith:")
print(books_checked_out_by_john_smith)
```

Books checked out by John Smith:

[('My First SQL book', 'Mary Parker', 981483029127), ('My Second SQL book', 'John Mayer', 857300923713)]

In [51]:

```
#Find all reviewers for the book "My Third SQL book" (SQL query in Python)
query_sql = '''
SELECT u.full_name
FROM Reviews r
JOIN Users u ON r.reviewer_name = u.full_name
JOIN Books b ON r.book_id = b.book_id
WHERE b.title = 'My Third SQL book';
'''

# Execute the query using the cursor
cursor.execute(query_sql)

# Fetch and display the results
reviewers_sql = cursor.fetchall()
print("Reviewers for 'My Third SQL book' (SQL Query):")
for reviewer in reviewers_sql:
    print(reviewer[0])  # Print the full name of each reviewer
```

Reviewers for 'My Third SQL book' (SQL Query):

Jane Smith

In [53]:

```
# Find the users that have no books checked out.

# SQL Query to find users with no books checked out
```

```
query_sql = '''
SELECT u.full_name
FROM Users u
LEFT JOIN Checkouts c ON u.user_id = c.user_id
WHERE c.checkout_id IS NULL;
'''

# Execute the query
cursor = conn.cursor()
cursor.execute(query_sql)

# Fetch and display the results
users_no_checkout_sql = cursor.fetchall()
print("Users with no books checked out (SQL Query):")
for user in users_no_checkout_sql:
    print(user[0])  # Print the full name of each user
```

Users with no books checked out (SQL Query):

Harry Potter

In [55]:

```
# Query to show all records in the Logs table

# Using SQLite to execute SQL queries
cursor.execute("SELECT * FROM Logs")
logs = cursor.fetchall()
print("All records in the Logs table (SQL Query):")
print(logs)
```

All records in the Logs table (SQL Query):

[(1, 5, '2024-11-07 18:13:06')]

In [59]:

```
#B. Using pandas, sending the sql query to pandas function (read_sql_query) and grnerate the same output

#Trigger:

# Load Logs table data into a Pandas DataFrame
logs_df = pd.read_sql_query("SELECT * FROM Logs", conn)
print("All records in the Logs table (Pandas Query):")
```

```
print(logs_df)
```

All records in the Logs table (Pandas Query):

```
   id  checkout_id          timestamp
0   1            5  2024-11-07 18:13:06
```

In [65]:

```
#1.Find the title, authors and the isbn of the books that 'John Smith' has checked out.

#Pandas Query

# Execute the query and store the result in a DataFrame
books_checked_out_by_john = pd.read_sql_query(query, conn)


# Display the results
print(books_checked_out_by_john)
```

```
              title       author          isbn
0   My First SQL book  Mary Parker  981483029127
1  My Second SQL book   John Mayer  857300923713
```

In [67]:

```
#2Find all reviewers for the book "My Third SQL book"

# Query to find reviewers for "My Third SQL Book" using Pandas
reviews_df = pd.read_sql_query("SELECT * FROM Reviews", conn)
users_df = pd.read_sql_query("SELECT * FROM Users", conn)  # Ensure you have Users DataFrame
books_df = pd.read_sql_query("SELECT * FROM Books", conn)  # Ensure you have Books DataFrame


# Merge DataFrames and filter for the specific book title
reviewers_pandas = reviews_df.merge(users_df, left_on='reviewer_name', right_on='full_name') \
    .merge(books_df, on='book_id')
reviewers_pandas = reviewers_pandas[reviewers_pandas['title'] == 'My Third SQL book']
print("Reviewers for 'My Third SQL Book' (Pandas Query):")
print(reviewers_pandas[['full_name']])
```

```
Reviewers for 'My Third SQL Book' (Pandas Query):
   full_name
3  Jane Smith
```

In [69]:

```
#3.Find the users that have no books checked out


# Pandas Query
```

```
users_df = pd.read_sql_query("SELECT * FROM Users", conn)

checkouts_df = pd.read_sql_query("SELECT * FROM Checkouts", conn)


# Identify users with checkouts

users_with_checkout = checkouts_df['user_id'].unique()


# Filter to find users with no checkouts

users_no_checkout_pandas = users_df[~users_df['user_id'].isin(users_with_checkout)]

print("Users with no books checked out (Pandas Query):")

print(users_no_checkout_pandas[['full_name']])
```

```
Users with no books checked out (Pandas Query):

     full_name

2  Harry Potter
```

In [71]:

```
#4.Show all the records of the logs table

#pandas query

logs_df = pd.read_sql_query("SELECT * FROM Logs", conn)

print("All records in the Logs table (Pandas Query):")

print(logs_df)
```

```
All records in the Logs table (Pandas Query):

   id  checkout_id          timestamp

0   1            5  2024-11-07 18:13:06
```

In [3]:

```
pip install nbconvert
```

```
Defaulting to user installation because normal site-packages is not writeableNote: you may need to restart the kernel to use updated packages.


Requirement already satisfied: nbconvert in c:\programdata\anaconda3\lib\site-packages (7.10.0)

Requirement already satisfied: beautifulsoup4 in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (4.12.3)

Requirement already satisfied: bleach!=5.0.0 in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (4.1.0)

Requirement already satisfied: defusedxml in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (0.7.1)

Requirement already satisfied: jinja2>=3.0 in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (3.1.4)

Requirement already satisfied: jupyter-core>=4.7 in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (5.7.2)

Requirement already satisfied: jupyterlab-pygments in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (0.1.2)

Requirement already satisfied: markupsafe>=2.0 in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (2.1.3)
```

Requirement already satisfied: mistune<4,>=2.0.3 in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (2.0.4)

Requirement already satisfied: nbclient>=0.5.0 in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (0.8.0)

Requirement already satisfied: nbformat>=5.7 in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (5.9.2)

Requirement already satisfied: packaging in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (23.2)

Requirement already satisfied: pandocfilters>=1.4.1 in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (1.5.0)

Requirement already satisfied: pygments>=2.4.1 in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (2.15.1)

Requirement already satisfied: tinycss2 in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (1.2.1)

Requirement already satisfied: traitlets>=5.1 in c:\programdata\anaconda3\lib\site-packages (from nbconvert) (5.14.3)

Requirement already satisfied: six>=1.9.0 in c:\programdata\anaconda3\lib\site-packages (from bleach!=5.0.0->nbconvert) (1.16.0)

Requirement already satisfied: webencodings in c:\programdata\anaconda3\lib\site-packages (from bleach!=5.0.0->nbconvert) (0.5.1)

Requirement already satisfied: platformdirs>=2.5 in c:\programdata\anaconda3\lib\site-packages (from jupyter-core>=4.7->nbconvert) (3.10.0)

Requirement already satisfied: pywin32>=300 in c:\programdata\anaconda3\lib\site-packages (from jupyter-core>=4.7->nbconvert) (305.1)

Requirement already satisfied: jupyter-client>=6.1.12 in c:\programdata\anaconda3\lib\site-packages (from nbclient>=0.5.0->nbconvert) (8.6.0)

Requirement already satisfied: fastjsonschema in c:\programdata\anaconda3\lib\site-packages (from nbformat>=5.7->nbconvert) (2.16.2)

Requirement already satisfied: jsonschema>=2.6 in c:\programdata\anaconda3\lib\site-packages (from nbformat>=5.7->nbconvert) (4.19.2)

Requirement already satisfied: soupsieve>1.2 in c:\programdata\anaconda3\lib\site-packages (from beautifulsoup4->nbconvert) (2.5)

Requirement already satisfied: attrs>=22.2.0 in c:\programdata\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (23.1.0)

Requirement already satisfied: jsonschema-specifications>=2023.03.6 in c:\programdata\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (2023.7.1)

Requirement already satisfied: referencing>=0.28.4 in c:\programdata\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (0.30.2)

Requirement already satisfied: rpds-py>=0.7.1 in c:\programdata\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (0.10.6)

Requirement already satisfied: python-dateutil>=2.8.2 in c:\programdata\anaconda3\lib\site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (2.9.0.post0)

Requirement already satisfied: pyzmq>=23.0 in c:\programdata\anaconda3\lib\site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (25.1.2)

Requirement already satisfied: tornado>=6.2 in c:\programdata\anaconda3\lib\site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (6.4.1)

In [7]:

```
!jupyter nbconvert --to html "Assignment 2 v.final.ipynb"
```

[NbConvertApp] Converting notebook Assignment 2 v.final.ipynb to html

[NbConvertApp] Writing 327445 bytes to Assignment 2 v.final.html

In [ ]: