

Final Project Report

Group Members : Amanda Barger, Ryan Feinberg, Jordan Jolliff

Overview:

Our project will be based on a movie theater database system called “Jar Movies”. The database stores information relevant to movie theaters and the relationships associated with its many subentities. The movie theater system has different privileges for Members and Staff, creating a different view for each type of user. To ensure database security, we implemented invalid character detection, using a pre-generated list of characters that could cause a SQL injection attack to check against any user input.

Design:

- Database design and implementation is done in postgresql.
- Interface design and implementation is done in python.

Psycopg2 is the library used to communicate between our python code and the postgresql database.

```
import psycopg2
```

.connect() allows us to connect to our database, and .cursor() allows us to perform queries on the database.

```
# connect to database server
conn = psycopg2.connect(host="localhost", dbname="jarMovies", user="postgres",
password="database_design", port=5433)

# create cursor to database
cur = conn.cursor()
```

Table creation and value insertion is handled by running Tables.sql and Insert.sql respectively. Using the commands shown below.

```
# initiate tables in database if they don't exist
cur.execute(open("/Users/ryan/Desktop/Database
Design/FinalProjectCollection/MovieTheaterDB/Tables.sql", "r").read())
conn.commit()
```

```
# insert into or update values in database
cur.execute(open("/Users/ryan/Desktop/Database
Design/FinalProjectCollection/MovieTheaterDB/Inserts.sql", "r").read())
conn.commit()
```

Snippets of table and insertion sql files (files included in submission).

```
10
11 CREATE TABLE IF NOT EXISTS Theaters (
12     TheaterCode VARCHAR(12) PRIMARY KEY,
13     Address VARCHAR(100),
14     Sponsor VARCHAR(50)
15 );
16
17 CREATE TABLE IF NOT EXISTS Rooms (
18     RoomID INTEGER PRIMARY KEY,
19     TheaterCode VARCHAR(12) REFERENCES Theaters ON DELETE CASCADE,
20     isXD BOOLEAN,
21     Capacity INTEGER
22 );
23
24 CREATE TABLE IF NOT EXISTS At (
25     MovieID VARCHAR(12) REFERENCES Movies NOT NULL,
26     TheaterCode VARCHAR(12) REFERENCES Theaters NOT NULL
27 );
28
29 CREATE TABLE IF NOT EXISTS Showing (
30     ShowingID VARCHAR(10) PRIMARY KEY,
31     MovieID VARCHAR(12) REFERENCES Movies NOT NULL,
32     MovieName VARCHAR(50),
33     Date DATE,
34     StartTime TIME,
35     TheaterCode VARCHAR(12) REFERENCES Theaters NOT NULL,
36     RoomID INTEGER REFERENCES Rooms NOT NULL
37 );
38
39 CREATE TABLE IF NOT EXISTS Customers (
40     MemberID CHAR(10) PRIMARY KEY,
41     Password VARCHAR(50) NOT NULL,
42     Points INTEGER,
43     Name VARCHAR(50),
```

```
8
9 INSERT INTO Theaters(TheaterCode, Address, Sponsor)
10 VALUES
11 (1111, '123 Movie Ln', 'AMC'),
12 (2222, '987 Theater Blvd', 'Regal Theaters'),
13 (3333, '567 Business Dr', 'Local Theater');
14
15 INSERT INTO Rooms(RoomID, TheaterCode, isXD, Capacity)
16 VALUES
17 (1, 1111, 'false', 80),
18 (2, 1111, 'false', 75),
19 (3, 1111, 'true', 100),
20 (4, 1111, 'false', 85),
21 (5, 1111, 'true', 120);
22
23
24 INSERT INTO At(MovieID, TheaterCode)
25 VALUES
26 (123456789101, 1111),
27 (111111111111, 2222),
28 (222222222222, 3333),
29 (333333333333, 1111);
30
31 INSERT INTO Showing(ShowingID, MovieID, MovieName, Date, StartTime, TheaterCode, RoomID)
32 VALUES
33 (123456789, 123456789101, 'The Hunger Games', '01-01-2023', '17:00:00', 1111, 1),
34 (234567890, 111111111111, 'Wish', '01-01-2023', '13:30:00', 1111, 2),
35 (564789123, 222222222222, 'Tiger 3', '01-01-2023', '16:00:00', 1111, 3),
36 (456789412, 333333333333, 'Made Up Movie', '01-01-2023', '20:30:00', 1111, 5),
37 (456879122, 123456789101, 'The Hunger Games', '01-01-2023', '18:00:00', 2222, 1),
38 (546897123, 111111111111, 'Wish', '01-01-2023', '14:00:00', 2222, 3),
39 (984156448, 222222222222, 'Tiger 3', '01-01-2023', '15:30:00', 2222, 2),
40 (456897412, 333333333333, 'Made Up Movie', '01-01-2023', '15:00:00', 3333, 4),
41 (456789121, 123456789101, 'The Hunger Games', '01-01-2023', '17:30:00', 3333, 1)
```

The design of our database we submitted for Milestone 1 had two changes in the final design. In the original Showing table the primary key was set to movieID, and when it came to creating the tables in SQL we realized that we needed to make a new primary key attribute since a movie can have different showings, such as different rooms and different times. So we created the ShowingID as the primary key for the Showing table. Another change we made was adding a primary key attribute to the ConcessionPurchase table. Our original ConcessionPurchase table had two foreign keys set as primary keys. When a customer purchases something they get a number on their receipt. So we added a Receipt attribute as the only primary key for the ConcessionPurchase table.

Our Extended Relation Diagram generated by our database (pgAdmin 4) below (load Tables.sql into pgAdmin 4 to see).

Below is the code snippet for the main page function, which gives a good example of the properties described above.

```
def main_page():
    userInput = -1
    while(userInput!=0):
        print("\n\n#####")
        print("Welcome to JAR, where movies are.")
        print("#####")
        print("Navigations: ")
        print("1. Member Login")
        print("2. Admin Page")
        print("3. About JAR")
        print("0. Exit")
        # check user input
        userInput = int(input("Please select by entering one of digits 0-3: "))
        while(userInput < 0 or userInput > 3):
            print("---> Invalid selection <---")
            userInput = int(input("Please select by entering one of digits 0-3: "))

        # page transitions:
        if userInput == 1:
            member_login()
        if userInput == 2:
            admin_page()
        if userInput == 3:
            about_page()
        if userInput == 0:
            print("Exiting program... Goodbye :)")
```

Adding a user to the account

```
if userInput == 1: # signup <-----
```

```

# get username and password (iterates until user confirms their username and password
is entered correctly)
userInput = -1
while(userInput!=1):
    name = input("Enter your full name: ")
    email = input("Enter an email address: ")
    password = input("Enter a password: ")
    MemberID = str(random.randint(999999999, 999999999))
    userInput = int(input("Is name: {}, email: {}, and password: {} correct? (1=yes,
0=no): ".format(name, email, password)))
# Injection attack check
if userInput:
    for CHAR in INJECTIONCHARS:
        if ((CHAR in name) or (CHAR in email) or (CHAR in password)):
            print(f"Invalid character = {CHAR}\nReturning to Main Page...")
            return
# Account creation valid
print("Creating Account...")
print("Account created with Member ID: {}".format(MemberID))
cur.execute(''' INSERT INTO Customers(MemberID, Password, Points, Name, Email)
VALUES({}, '{}', 0, '{}', '{}') '''.format(MemberID, password, name, email))
conn.commit()

```

Notice MemberID is generated at this point. Also notice that we are checking for SQL injection attacks after the user inputs their information. This SQL injection technique is repeated throughout the program for each query that takes in user input(s).

```
INJECTIONCHARS = [' ', '\'', '-', '#', ';']
```

No user input query example (to see what each customer has purchased)

```

cur.execute(''' SELECT Customers.Name, ConcessionStand.Name
FROM Customers, ConcessionPurchase, ConcessionStand
WHERE Customers.MemberID = ConcessionPurchase.MemberID
AND ConcessionPurchase.ItemID = ConcessionStand.ItemID; ''')

```

One of the results:

Customer: Genia Bendix ---> Food: Small Popcorn

Creating views:

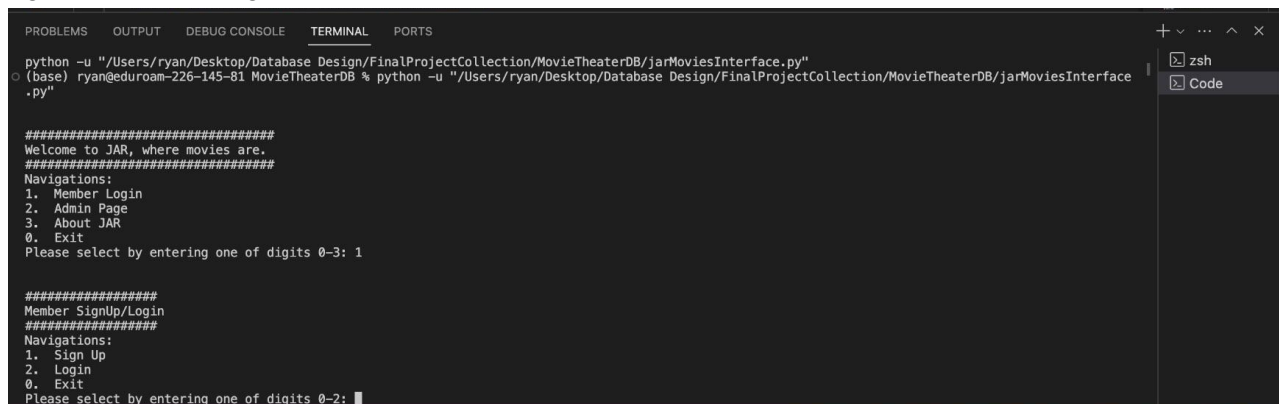
```
CREATE VIEW AdminViewConcessionStand
    AS SELECT ItemID, Name, Price, Quantity
    FROM ConcessionStand
    GROUP BY ItemID;
CREATE VIEW CustomerViewConcessionStand
    AS SELECT Name, Price
    FROM ConcessionStand;
CREATE VIEW StaffViewConcessionStand
    AS SELECT ItemID, Name, Price, Quantity
    FROM ConcessionStand
    GROUP BY ItemID;
```

```
# initializing customer view for movie
cur.execute(''' CREATE USER u{}; '''.format(MemberID, password))
conn.commit()
cur.execute(''' GRANT SELECT ON TABLE CustomerViewMovie TO u{};'''.format(MemberID))
conn.commit()
print("Account Created... Redirecting to login page...")
```

Creating different views ensures that there are different levels of privilege in the system, for instance, a customer can only SELECT specific tables, such as Movies, Showings, and Tickets. On the other hand, staff members can SELECT or UPDATE information in all tables connected to staff areas. At the highest tier, there is an administrator view that can SELECT, UPDATE, or DELETE information in all tables.

Main Functionality:

Our program uses a command-line interface. Upon loading up the program the user will see the main page, which consists of navigations to our other functionalities. Most important of them is the Member Login option (option 1.), which gives the user the option to make an account or sign into an existing account.



```
python -u "/Users/ryan/Desktop/Database Design/FinalProjectCollection/MovieTheaterDB/jarMoviesInterface.py"
(base) ryan@duroam-226-145-81 MovieTheaterDB % python -u "/Users/ryan/Desktop/Database Design/FinalProjectCollection/MovieTheaterDB/jarMoviesInterface.py"

#####
Welcome to JAR, where movies are.
#####
Navigations:
1. Member Login
2. Admin Page
3. About JAR
0. Exit
Please select by entering one of digits 0-3: 1

#####
Member SignUp/Login
#####
Navigations:
1. Sign Up
2. Login
0. Exit
Please select by entering one of digits 0-2: █
```

Example user sign up sequence:

```
#####
Member SignUp/Login
#####
Navigations:
1. Sign Up
2. Login
0. Exit
Please select by entering one of digits 0-2: 1
#####
Enter your full name: Ryan Feinberg
Enter an email address: ryan@usf.edu
Enter a password: pass123
Is name: Ryan Feinberg, email: ryan@usf.edu, and password: pass123 correct? (1=yes, 0=no): 1
Creating Account...
Account created with Member ID: 5906805299
```

Subsequent login sequence:

```
#####
Member SignUp/Login
#####
Navigations:
1. Sign Up
2. Login
0. Exit
Please select by entering one of digits 0-2: 2
#####
#####
Login:
Enter an email address: ryan@usf.edu
Enter a password: pass123

#####
Greetings JAR Member!!!
#####
Navigations:
1. See Profile
2. Change Email or Password
3. See Points
```

Member Portal functions:

```
#####
Greetings JAR Member!!!
#####
Navigations:
1. See Profile
2. Change Email or Password
3. See Points
4. See Purchased Tickets
5. Browse Movies
0. Logout
Please select by entering one of digits 0-5: █
```

1. Check Users name, email, and MemberID
2. Change email and/or password with UPDATE function
3. Check user points
4. See all tickets purchased by the current user
5. Browse all movies being shown in theaters. Also allows user to purchase a ticket any of the showing movies
0. Return to main page

Admin page contains a interface for sudo direct access to the database and all of its functionalities

```
#####
Welcome to JAR, where movies are.
#####
Navigations:
1. Member Login
2. Admin Page
3. About JAR
0. Exit
Please select by entering one of digits 0-3: 2

#####
Administrative page. All funtionality available
Navigations:
1. INSERT
2. SELECT
3. DELETE
4. Advanced Queries
0. Back
Selection: █
```

Select the desired query and then simply type the SQL statement into the command-line.
For example:

```
#####
Administrative page. All funtionality available
Navigations:
1. INSERT
2. SELECT
3. DELETE
4. Advanced Queries
0. Back
Selection: 2
postgresSQL SELECT query: SELECT * FROM Movies;
('123456789101', 'The Hunger Games', datetime.time(2, 38), 'Francis Lawrence', True, 7, False)
('1111111111111', 'Wish', datetime.time(1, 35), 'Fawn Veerasunthorn', False, -1, False)
('2222222222222', 'Tiger 3', datetime.time(2, 37), 'Maneesh Sharma', True, 8, True)
('3333333333333', 'Made Up Movie', datetime.time(0, 59), 'Ryan Feinberg', True, 10, True)
```