****

**A DCDSL PROJECT REPORT**

**ON**

**“Data-Driven Insights for BookMyShow”**

Submitted By

|  |  |
| --- | --- |
| Kashish Chelwani | 24070126082 |
| Kavish Nag | 24070126085 |
| Kunal Jhindal | 24070126098 |

UNDER THE GUIDANCE OF

Dr. Pooja Kamat

Assistant Professor

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE

A.Y. 2025-26

****

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING**

**CERTIFICATE**

This is to certify that the DCDSL Project work entitled “**Data-Driven Insights for BookMyShow**” is carried out by the **Students - Kashish Chelwani(24070126082), Kavish Nag(24070126085), Kunal Jhindal(24070126098)** in **Artificial Intelligence & Machine Learning**, Symbiosis International (Deemed University), Pune during the academic year 2024-2025.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Name and Signature of the Guide |  | Dr. Shilpa Bade-Gite |
|  |  | Head, Department of AI&ML |
|  |  |  |

## TABLE OF CONTENTS

|  |  |
| --- | --- |
|  | Page Number |
| Certificate |  |
| Introduction | 1 |
| Problem Statement | 2 |
| System architecture and Modules | 3 |
| Functional requirements | 4 |
| Entities, relationships and attributes | 5-6 |
| Relational schema | 7 |
| Implementation   1. Database Setup: Describe the installation and configuration steps. 2. Table Creation Scripts: Provide SQL scripts used to create tables and relationships. 3. Stored Procedures, Views, and Indexes: Include any stored procedures, views, or indexes created. 4. Sample Data: Present sample data | 8-16 |
| User Interface   1. Screenshots of the UI 2. Usage Instructions: Provide guidance on how users can interact with the database. | 17-22 |
| Conclusion | 23 |
| References | 24 |

## Introduction

**BookMyShow** is a leading online movie ticketing platform that manages millions of users, theaters, movies, and bookings daily. Managing such a large-scale ecosystem demands a robust and efficient data management system capable of handling real-time transactions and generating analytical insights. This project focuses on designing and implementing a **relational database management system (RDBMS)** for BookMyShow, integrating **SQL** and **Python** to enhance operational efficiency, data accessibility, and analytical performance.

The main objectives of the project include:

* Designing a **normalized relational database structure** to store and retrieve information about users, theaters, movies, shows, and bookings efficiently.
* Implementing **SQL-based functions, procedures, triggers, and views** to automate repetitive operations and maintain data integrity.
* Conducting **in-depth analytical operations** such as revenue tracking, user engagement analysis, movie rating evaluation, and performance reporting.
* Integrating **SQL with Python** for data visualization and advanced analytics to support data-driven decision-making.
* Improving **data accessibility, reporting accuracy, and overall system transparency** across all operational entities.

Together, these efforts aim to create a scalable and intelligent database solution that strengthens BookMyShow’s ability to analyze performance, optimize operations, and deliver a seamless user experience.

## Problem Statement

BookMyShow faces several analytical challenges despite handling vast data volumes:

* Difficulty in understanding **user engagement** and **retention trends**.
* Lack of metrics to measure **movie success** and **theater performance**.
* Inefficiency in **seat utilization** and **dynamic pricing**.
* Incomplete tracking of **revenue generation** and **payment reliability**.
* Underutilization of **customer reviews and sentiments** for content improvement.

Hence, this project leverages **SQL-based analytical insights** to uncover patterns, highlight inefficiencies, and support data-driven strategies to boost performance, revenue, and customer satisfaction.

## System Architecture and Modules

**Architecture Overview**

The system follows a **modular, multi-layer architecture**:

1. **Database Layer (MySQL):** Manages all relational data including users, movies, theaters, and transactions.
2. **Logic Layer (Stored Procedures & Functions):** Implements core business logic like booking creation, cancellations, revenue computation, etc.
3. **Application Layer (Python Integration):** Connects with MySQL using mysql.connector for automation, verification, and data visualization.
4. **User Interface Layer:** Provides user interaction for querying reports, executing bookings, and viewing analytics.

**System Modules**

| **Module** | **Description** |
| --- | --- |
| **User Management** | Handles user registration, details, and age calculation. |
| **Movie & Theater Management** | Stores and links movies, screens, and theaters. |
| **Booking System** | Manages ticket bookings, cancellations, and payments. |
| **Revenue Analysis** | Computes revenue per theater, date, and movie. |
| **Review Management** | Captures and analyzes customer reviews and ratings. |
| **Analytics & Reporting** | Generates summaries through SQL Views and Python visualizations. |

## Functional Requirements

 The system should store and manage data related to users, movies, theaters, shows, and bookings.

 It must support **booking creation, cancellation, and seat availability updates**.

 The system should generate **reports** on:

* Top-rated movies
* Theater performance
* Revenue trends
* Frequent customers

 Users should be able to **add reviews** and view average movie ratings.

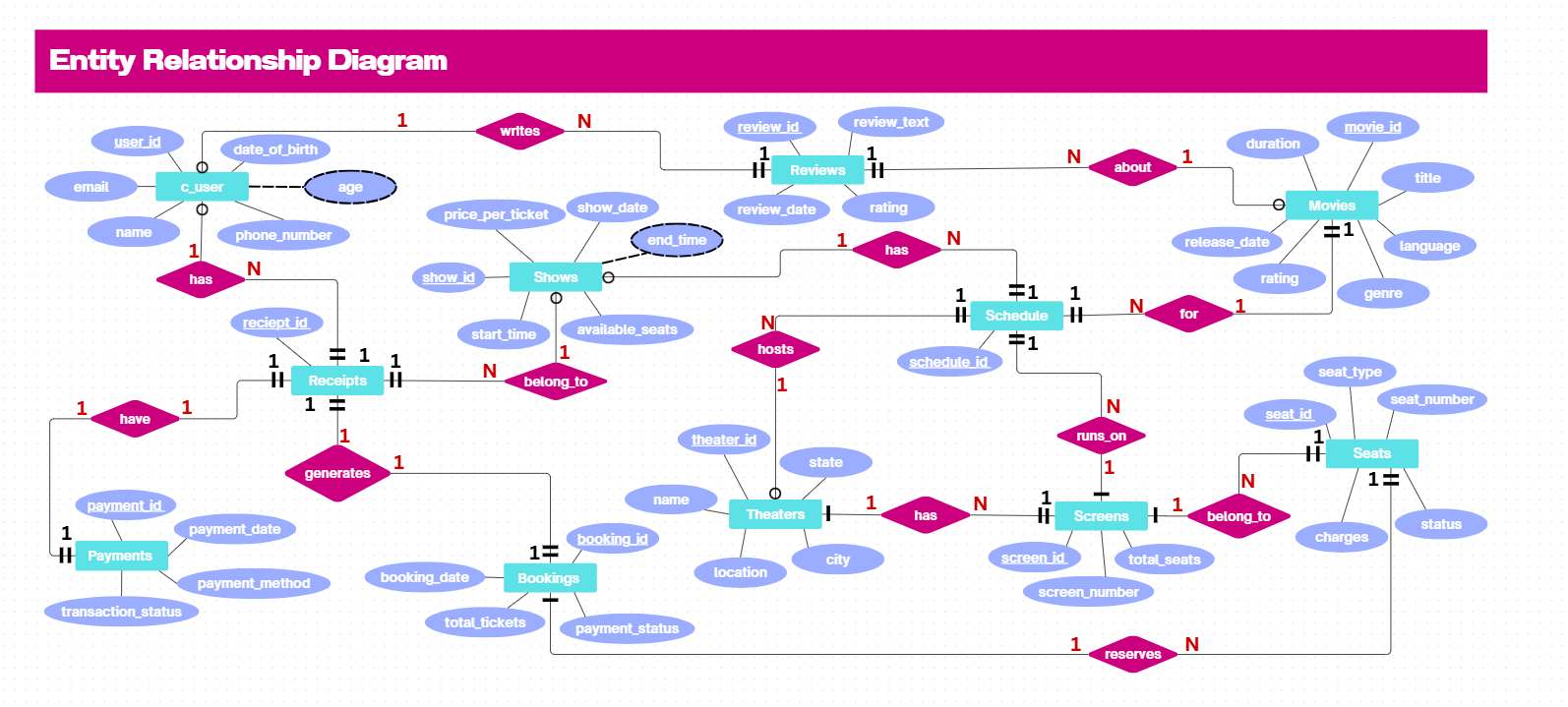
 Admins should be able to view **daily, monthly, and overall revenue summaries**.

 The platform should maintain **data consistency** and ensure **secure transactions**

## Entities, Relationships, and Attributes

**Main Entities**

| Entity | Attributes |
| --- | --- |
| User | user\_id, name, email, phone\_number, DOB |
| Movies | movie\_id, title, genre, language, duration, rating, release\_date |
| Theaters | theater\_id, name, city, state |
| Screens | screen\_id, screen\_number, total\_seats |
| Shows | show\_id, show\_date, start\_time, price\_per\_ticket, available\_seats |
| Bookings | booking\_id, booking\_date, total\_tickets, payment\_status |
| Payments | payment\_id, payment\_method, payment\_date, transaction\_status |
| Receipts | receipt\_id, booking\_id, user\_id, show\_id, payment\_id |
| Reviews | review\_id, user\_id, movie\_id, rating, review\_text, review\_date |
| Seats | seat\_id, screen\_id, seat\_number, seat\_type, status, charges |

****

## Relational Schema

• **c\_user**(user\_id, c\_name, email, phone\_number, DOB)  
• **movies**(movie\_id, title, genre, m\_lang, duration, rating, release\_date)  
• **theaters**(theater\_id, name, location, city, state)  
• **screens**(screen\_id, screen\_number, total\_seats)  
• **shows**(show\_id, show\_date, start\_time, price\_per\_ticket, available\_seats,)  
• **schedule**(schedule\_id, show\_id, movie\_id, theater\_id, screen\_id)  
• **bookings**(booking\_id, booking\_date, total\_tickets, payment\_status)  
• **seats**(seat\_id, booking\_id, screen\_id, seat\_number, seat\_type, charger, status)  
• **payments**(payment\_id, payment\_method, payment\_date, transaction\_status)  
• **receipts**(receipt\_id, booking\_id, user\_id, show\_id, payment\_id)  
• **reviews**(review\_id, user\_id, movie\_id, rating, review\_text, review\_date)

## Implementation

1. **Database Setup**

• Installed and configured MySQL Server 8.0.

• Created a new database named BookMyShow

• Designed and normalized tables based on the ER Diagram.

• Inserted sample data for testing project functionalities.

• Established connectivity with Python (Streamlit) using the mysql.connector library.

• Verified database operations using SQL queries and result visualization in the web app.

1. **Table Creation Scripts**

CREATE TABLE c\_user (

user\_id VARCHAR(20) PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100) UNIQUE,

phone\_number BIGINT,

date\_of\_birth DATE

);

CREATE TABLE Movies (

movie\_id VARCHAR(20) PRIMARY KEY,

title VARCHAR(200),

genre VARCHAR(50),

language VARCHAR(50),

duration INT,

rating DECIMAL(3,1),

release\_date DATE

);

CREATE TABLE Theaters (

theater\_id VARCHAR(20) PRIMARY KEY,

name VARCHAR(100),

location VARCHAR(100),

city VARCHAR(100),

state VARCHAR(100)

);

CREATE TABLE Screens (

screen\_id VARCHAR(20) PRIMARY KEY,

screen\_number INT,

total\_seats INT

);

CREATE TABLE Shows (

show\_id VARCHAR(20) PRIMARY KEY,

show\_date DATE,

start\_time TIME,

price\_per\_ticket DECIMAL(8,2),

available\_seats INT

);

CREATE TABLE Schedule (

schedule\_id VARCHAR(20) PRIMARY KEY,

show\_id VARCHAR(20),

movie\_id VARCHAR(20),

theater\_id VARCHAR(20),

screen\_id VARCHAR(20),

FOREIGN KEY (show\_id) REFERENCES Shows(show\_id),

FOREIGN KEY (movie\_id) REFERENCES Movies(movie\_id),

FOREIGN KEY (theater\_id) REFERENCES Theaters(theater\_id),

FOREIGN KEY (screen\_id) REFERENCES Screens(screen\_id)

);

CREATE TABLE Bookings (

booking\_id VARCHAR(20) PRIMARY KEY,

booking\_date DATETIME,

total\_tickets INT,

payment\_status VARCHAR(50)

);

CREATE TABLE Seats (

seat\_id VARCHAR(20) PRIMARY KEY,

booking\_id VARCHAR(20),

screen\_id VARCHAR(20),

seat\_number VARCHAR(10),

seat\_type VARCHAR(20),

charges DECIMAL(8,2),

status VARCHAR(20),

FOREIGN KEY (booking\_id) REFERENCES Bookings(booking\_id),

FOREIGN KEY (screen\_id) REFERENCES Screens(screen\_id)

);

CREATE TABLE Payments (

payment\_id VARCHAR(20) PRIMARY KEY,

payment\_method VARCHAR(50),

payment\_date DATETIME,

transaction\_status VARCHAR(50)

);

CREATE TABLE Reciepts (

reciept\_id VARCHAR(20) PRIMARY KEY,

booking\_id VARCHAR(20),

user\_id VARCHAR(20),

show\_id VARCHAR(20),

payment\_id VARCHAR(20),

FOREIGN KEY (booking\_id) REFERENCES Bookings(booking\_id),

FOREIGN KEY (user\_id) REFERENCES c\_user(user\_id),

FOREIGN KEY (show\_id) REFERENCES Shows(show\_id),

FOREIGN KEY (payment\_id) REFERENCES Payments(payment\_id)

);

CREATE TABLE Reviews (

review\_id VARCHAR(20) PRIMARY KEY,

user\_id VARCHAR(20),

movie\_id VARCHAR(20),

rating DECIMAL(3,1),

review\_text TEXT,

review\_date DATETIME,

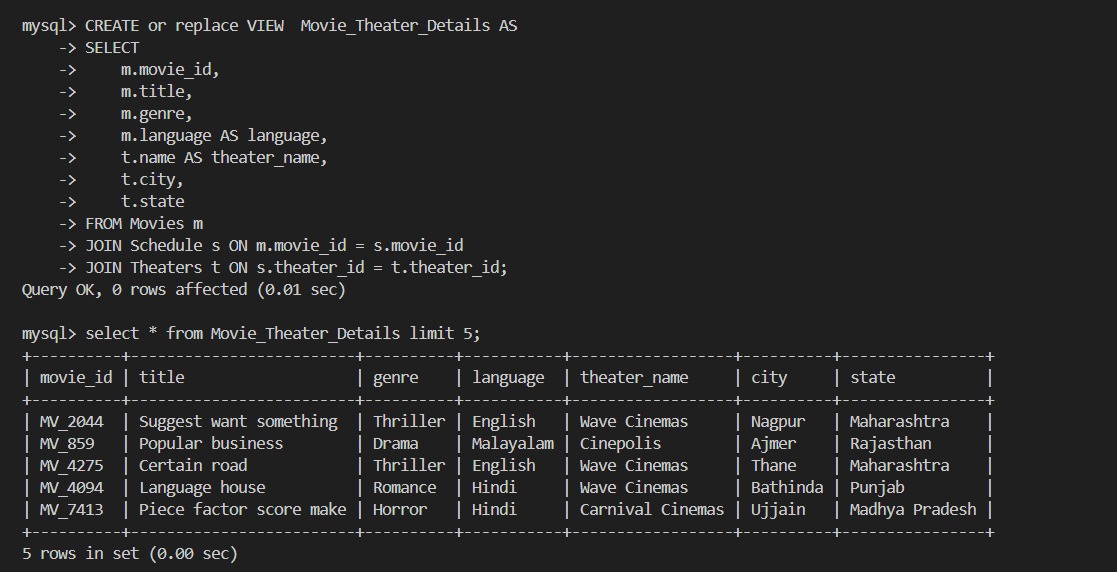
FOREIGN KEY (user\_id) REFERENCES c\_user(user\_id),

FOREIGN KEY (movie\_id) REFERENCES Movies(movie\_id)

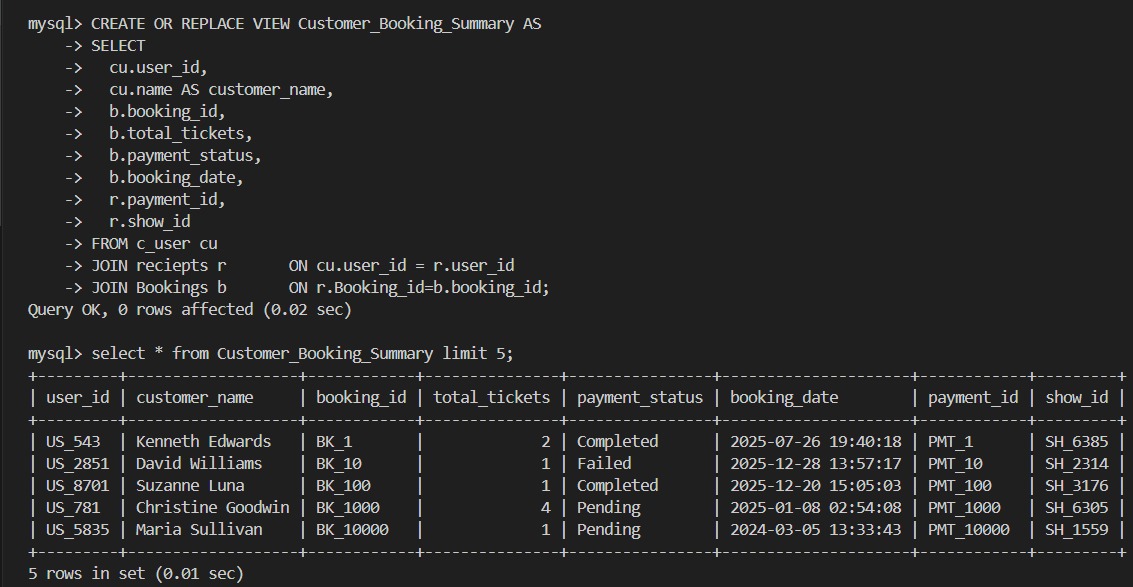
);

1. **Stored Procedures, Views, and Indexes**

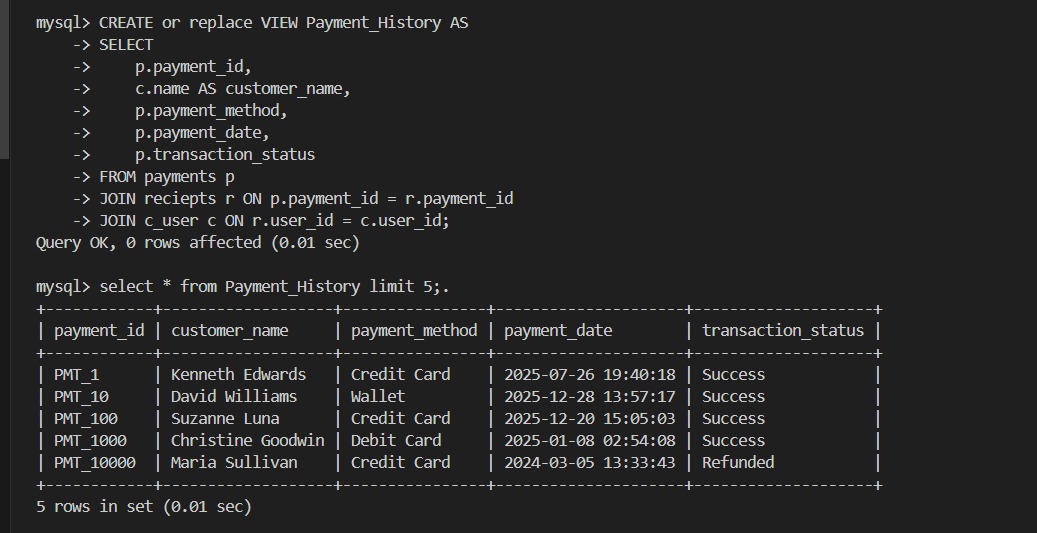
**VIEW** - This view displays all movies along with their genres, languages, and the theaters where they are being shown, including the theater’s city and state details.



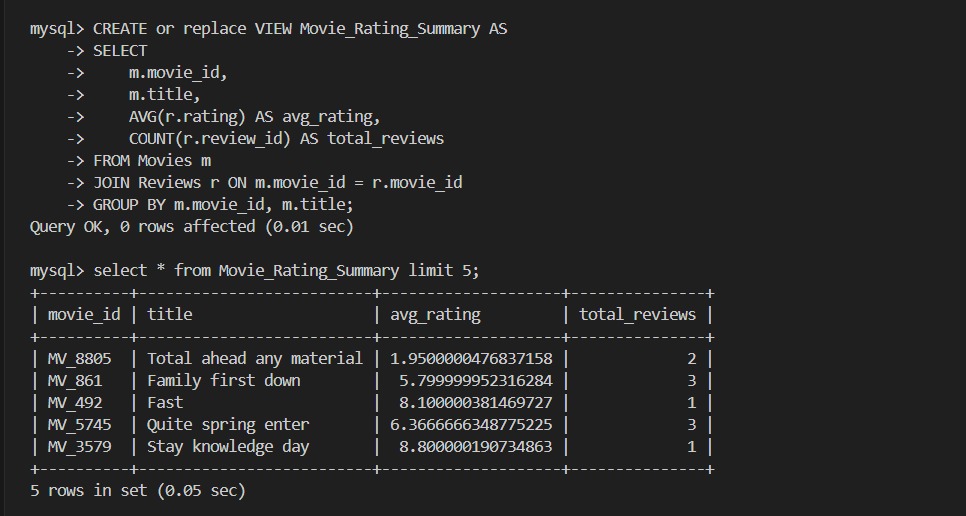
**VIEW** - This view displays all customer booking details along with user information, total tickets booked, payment status, booking date, payment ID, and show ID.



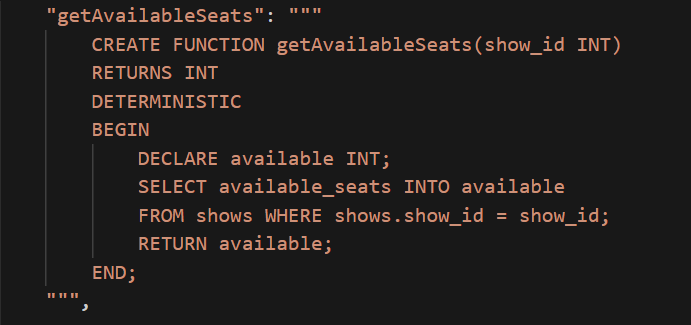
**VIEW** - This view displays all payment records along with customer names, payment methods, payment dates, and transaction statuses.

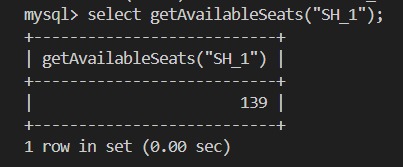


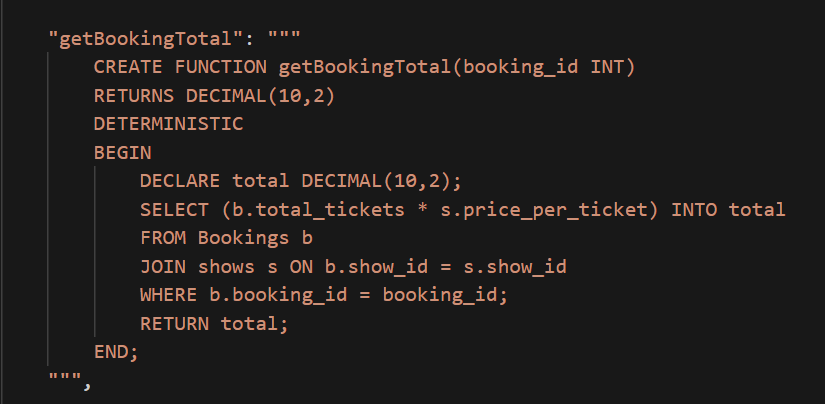
**VIEW** - This view displays all movies along with their average ratings and the total number of reviews each movie has received.

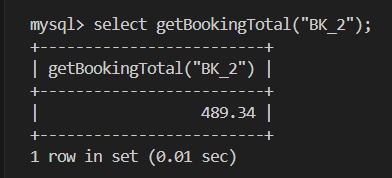


**FUNCTION** - This function takes a show ID (show\_id INT) as input. Its purpose is to query the shows table to retrieve and return the number of available seats (available\_seats) for the specified show.





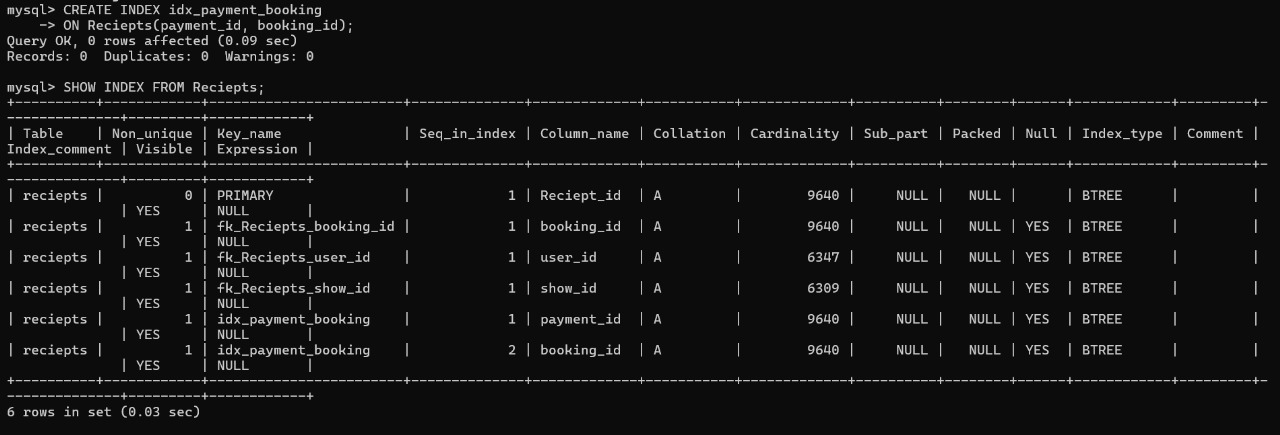
**FUNCTION** - This function accepts a booking ID (booking\_id INT). It calculates and returns the total cost of the booking by multiplying the number of tickets by the show's price per ticket.



**PROCEDURE** - This stored procedure ShowPerformanceSummary calculates the occupancy rate for a given show by comparing total seats and available seats from related tables. It returns the percentage of seats booked, helping analyze show performance and theater utilization.

****

**INDEX**- This index is a combined index on the Receipts table that makes searches by payment\_id and booking\_id much faster.



**TRIGGER**- This trigger automatically updates the payment status of a booking to “Paid” after a successful payment is inserted into the payments table. It retrieves the corresponding booking ID from the receipts table and ensures that only successful transactions update the booking status.

DELIMITER //

CREATE TRIGGER update\_booking\_status\_after\_payment

AFTER INSERT ON payments

FOR EACH ROW

BEGIN

DECLARE book\_id VARCHAR(20);

-- Get corresponding booking\_id from reciepts

SELECT booking\_id INTO book\_id

FROM reciepts

WHERE payment\_id = NEW.payment\_id

LIMIT 1;

-- If payment successful and booking found, mark booking as paid

IF NEW.transaction\_status = 'Success' AND book\_id IS NOT NULL THEN

UPDATE bookings

SET payment\_status = 'Paid'

WHERE booking\_id = book\_id;

END IF;

END //

DELIMITER ;

1. **Sample Data**

To test and validate the functionality of the BookMyShow database, **sample data** was inserted into each table.  
This data represents realistic scenarios involving users, movies, theaters, bookings, and payments to ensure that the system behaves as expected during query execution and report generation.

The sample records were designed to:

* Simulate different user profiles, theater locations, and movie genres.
* Represent real-world cases such as multiple bookings, various payment methods, and different review ratings.
* Enable verification of analytical SQL functions, stored procedures, and views such as Daily Revenue, Top Rated Movies, and Frequent Customers.

Below are examples of some inserted data:

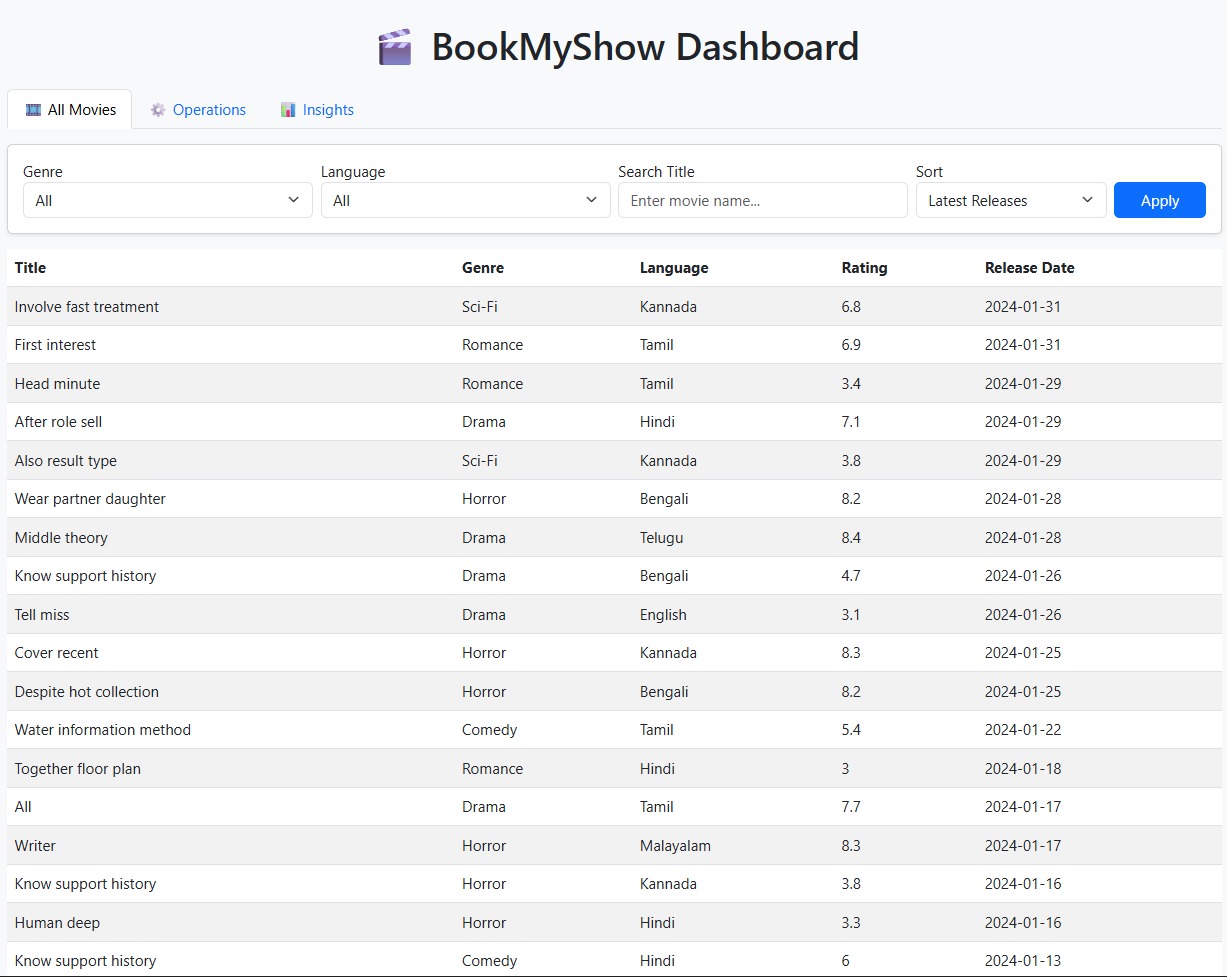
| **Table** | **Sample Data Example** |
| --- | --- |
| **c\_user** | (‘US\_1’, ‘Allison Hill, ‘[allisonhill@gmail.com](mailto:allisonhill@gmail.com)’, 8143513964, ‘1998-03-15’) |
| **movies** | (‘MV\_1’, ‘Address clear, ‘Action’, ‘Kannada, 118, 4.3, ‘2023-01-25’) |
| **theaters** | (‘TH\_1’, ‘PVR Cinemas’, ‘Andheri’, ‘Mumbai’, ‘Maharashtra’) |
| **shows** | (‘SH\_1’, ‘2025-03-10’, ‘18:00:00’, 250.00, 100) |
| **screens** | (‘SC\_1’, 2, 266) |
| **bookings** | (‘BK\_1’, ‘2025-03-05 14:32:00’, 2, ‘Completed’) |
| **payments** | (‘PMT\_1’, ‘UPI’, ‘2025-03-05 14:35:00’, ‘Success’) |
| **reviews** | (‘REV\_1’, ‘US\_8392, ‘MV\_8805, 4.5, ‘Great movie!’, ‘2025-03-06 10:00:00’) |
| **seats** | (‘ST\_1’, ‘BK\_9384’, ‘SC\_666’, ‘B18’, ‘VIP’, 50, ‘Available’) |
| **schedule** | (‘S\_1’, ‘SH\_1’, ‘MV\_2044’, ‘TH\_489’, ‘SC\_148’) |
| **receipts** | (‘R\_1’, ‘BK\_1’, ‘US\_543, ‘SH\_6385’, ‘PMT\_1’) |

All datasets were verified to ensure:

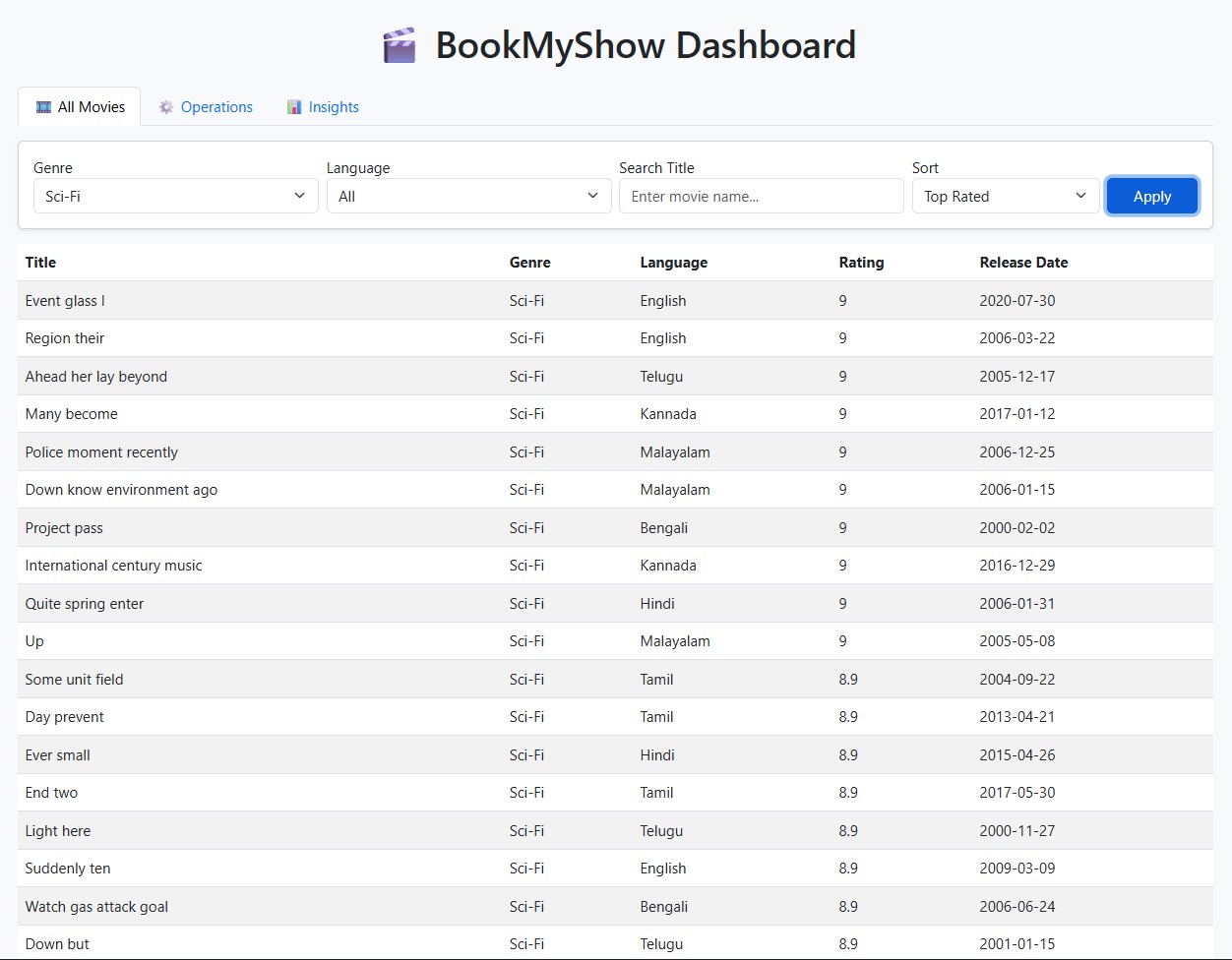
* Referential integrity between parent and child tables.
* Correct outputs for all SQL functions and views.

## ****User Interface****

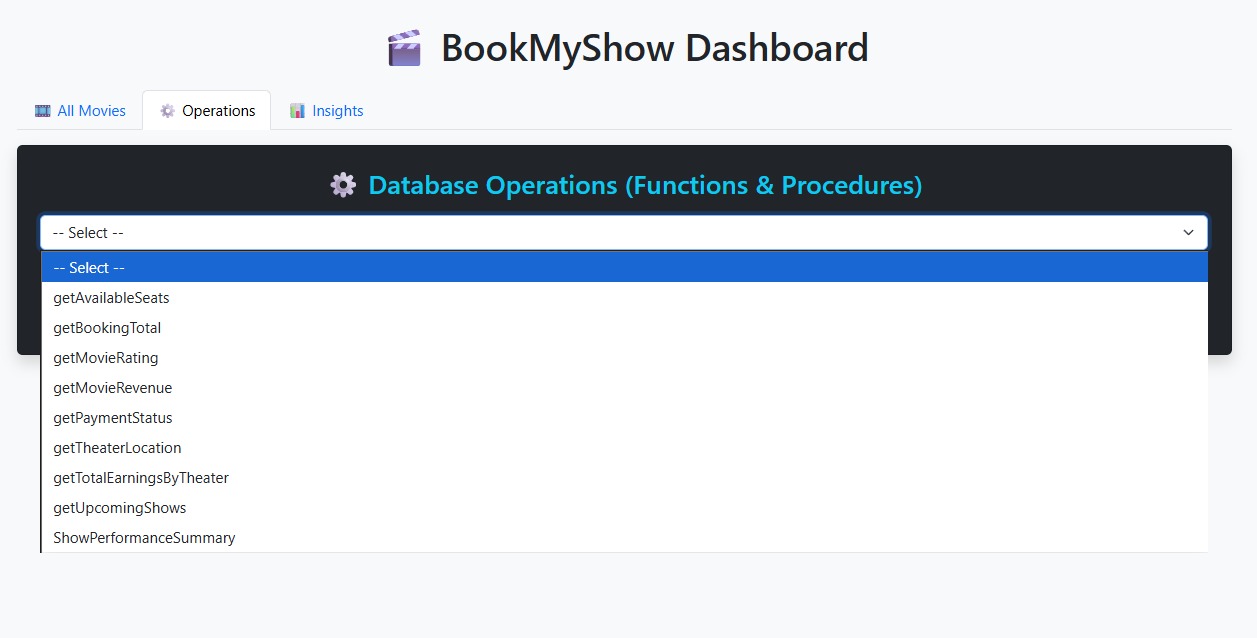
### ****Screenshots of the UI****



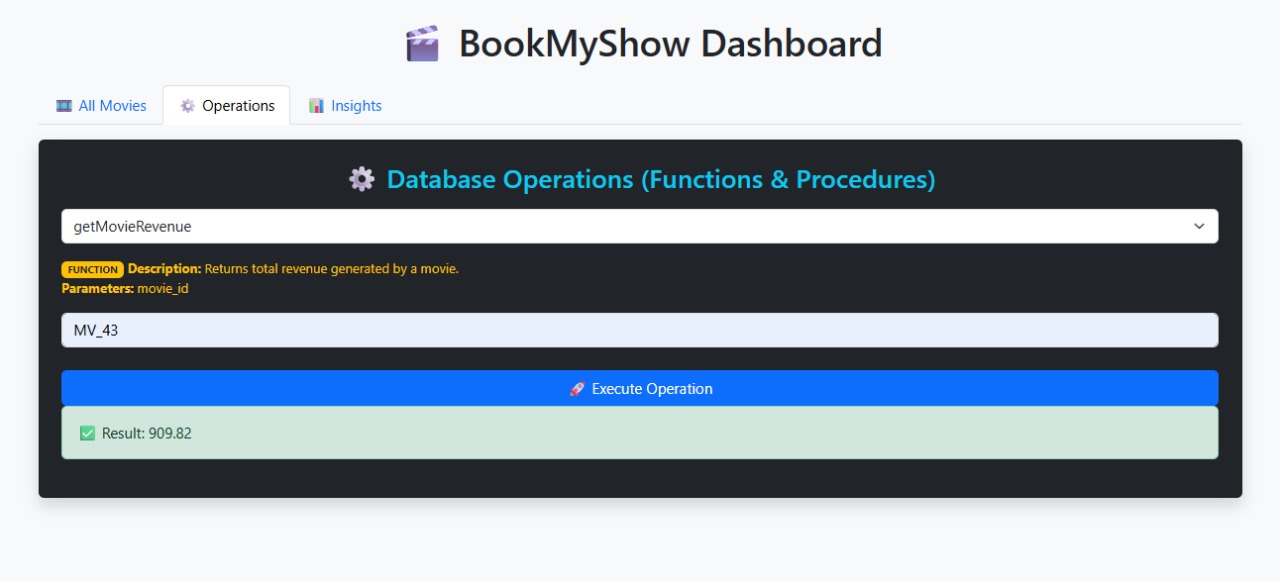
A dynamic web-based dashboard that displays all movies from the BookMyShow database without any filters applied, built using Flask and MySQL.



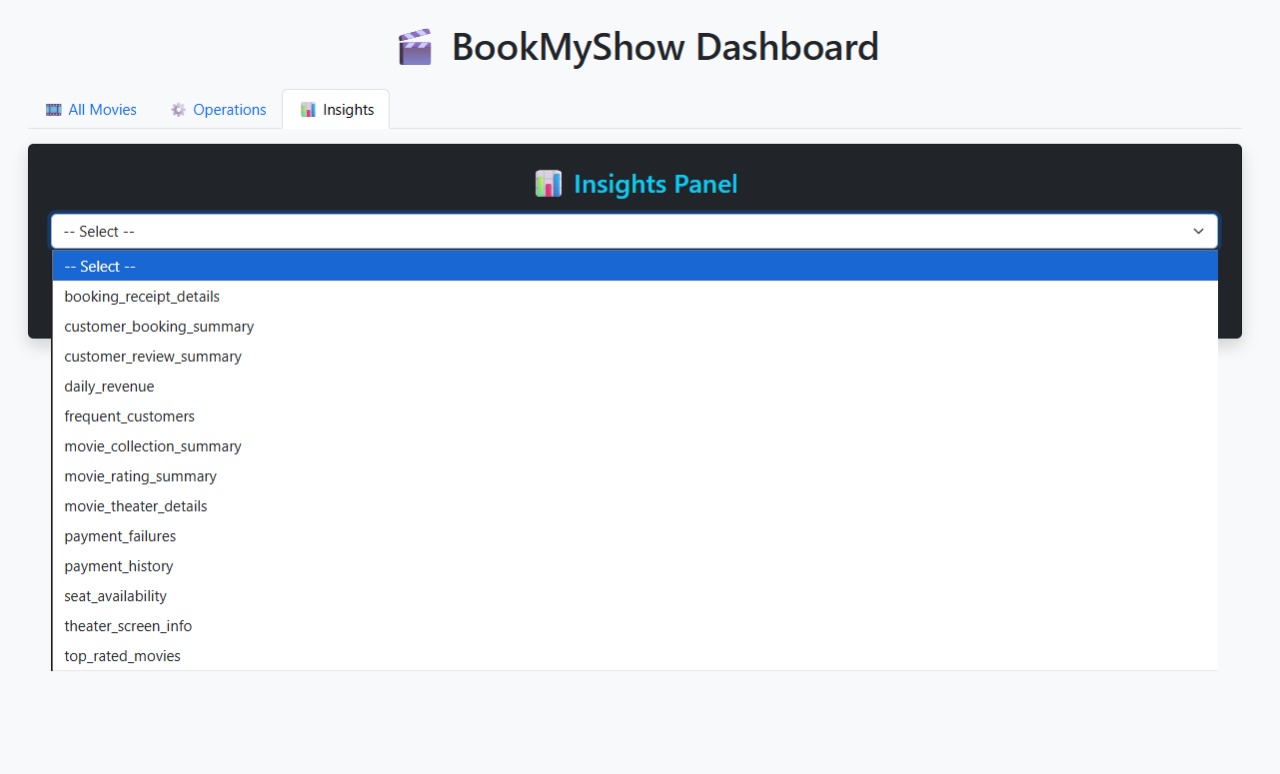
A dynamic BookMyShow dashboard view displaying only Sci-Fi movies from the database, filtered by genre and sorted by top-rated titles using Flask and MySQL.



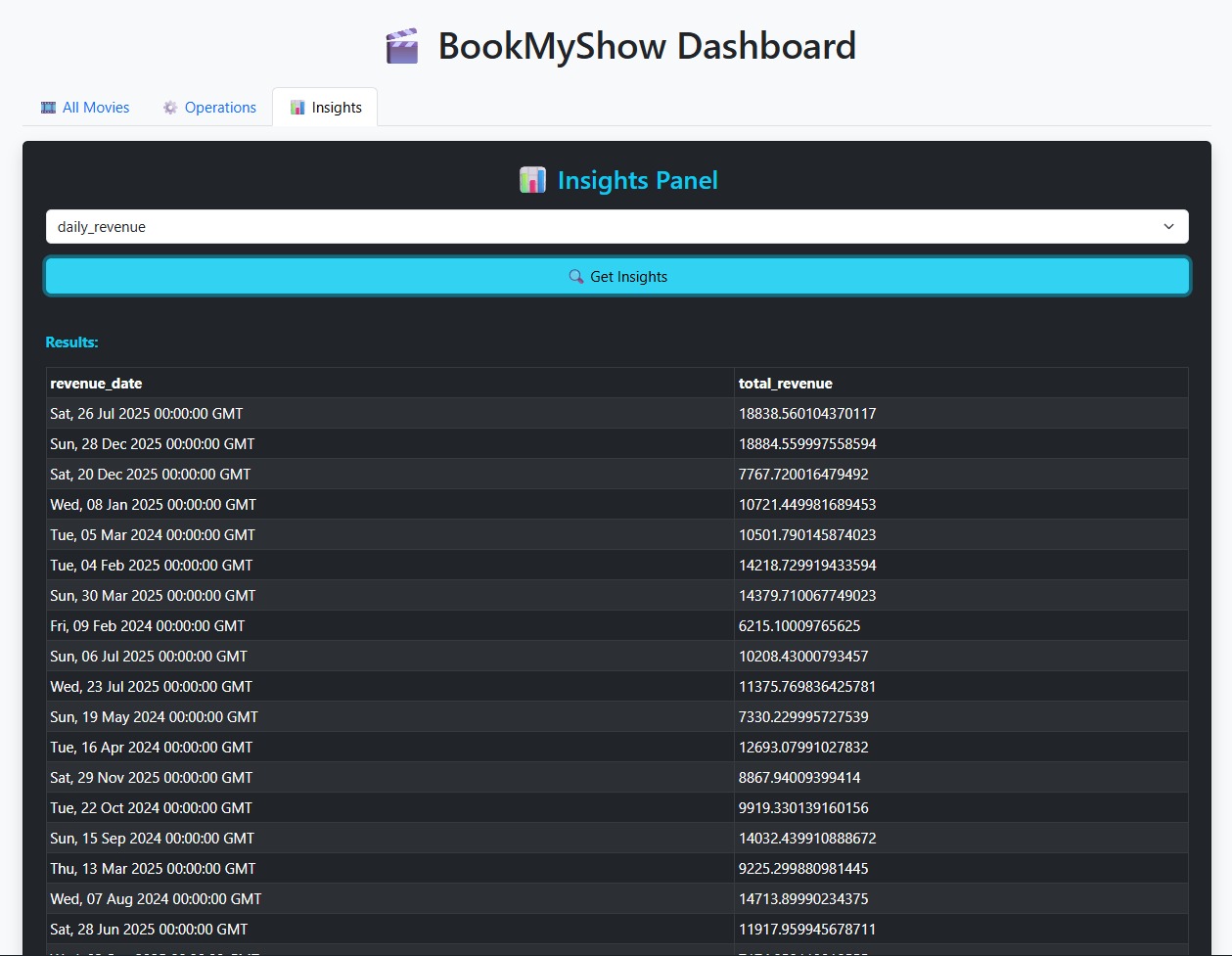
A dashboard view displaying the database operations panel, where users can select and execute various MySQL functions and procedures from the BookMyShow database.



The dashboard executes the getMovieRevenue MySQL function with a given movie ID to display the total revenue generated by that specific movie.



The Insights Panel of the BookMyShow Dashboard allows users to select and view analytical reports from various MySQL database views such as revenue, ratings, and customer summaries.



The Insights Panel displays data from the daily\_revenue view, showing total revenue generated on each date retrieved directly from the BookMyShow database.

1. **Usage Instructions**

## Overview

The BookMyShow Dashboard is a web-based application developed using Flask, MySQL, HTML, CSS, and Bootstrap. It provides a single interface for managing and analyzing data related to movies, bookings, theaters, and payments. Users can view movies, execute MySQL functions and stored procedures, and generate analytical insights through database views.

## System Requirements

Software Requirements:  
- Python 3.8 or above  
- MySQL 8.0 or above  
- Flask Framework  
- MySQL Connector for Python  
- Web browser (Google Chrome recommended)  
  
Python Libraries Used:  
pip install flask mysql-connector-python

## Database Configuration

1. Open MySQL and create a new database:  
 CREATE DATABASE bookmyshow;  
 USE bookmyshow;  
2. Create all necessary tables (e.g., movies, bookings, users, etc.).  
3. Define all required functions, stored procedures, and views as per project specifications.  
4. Update database credentials in the Flask file app.py if needed.

## Running the Application

1. Open the project folder in the terminal or command prompt.  
2. Run the Flask application:  
 python app.py  
3. Open a web browser and visit:  
 http://127.0.0.1:5000/  
4. The homepage will load the dashboard once the MySQL connection is successfully established.

## Application Modules

a) Movies Section:  
- Displays all movies from the database.  
- Users can filter results by genre, language, or title.  
- Sorting options include: Latest Releases, Top Rated, Alphabetical Order.  
- Pagination allows navigation through large datasets.  
  
b) Functions Module:  
- Allows users to execute MySQL functions dynamically.  
 Steps:  
 1. Select a function from the dropdown.  
 2. Enter the required parameters.  
 3. Click 'Get Results' to execute and view output.  
  
c) Stored Procedures Module:  
- Executes pre-defined stored procedures to generate analytical reports.  
 Steps:  
 1. Select a stored procedure.  
 2. Provide the required parameters.  
 3. Click 'Execute Procedure'.  
 4. Results are displayed in a formatted table.  
  
d) Views Module:  
- Fetches and displays data from database views.  
 Steps:  
 1. Select a view name.  
 2. Click 'Get Insights'.  
 3. Output table is displayed.

## Web Interface

The web interface uses Bootstrap 5 for responsive design. style.css adds custom Amazon-inspired themes, gradient effects, and hover animations. Data tables and buttons are dynamically updated using JavaScript and Fetch API calls to Flask backend endpoints.

## API Endpoints

/api/movies (POST): Fetches movies based on filters  
/api/function\_info/<name> (GET): Retrieves metadata for a MySQL function  
/api/execute\_function (POST): Executes a MySQL function  
/api/procedure\_info/<name> (GET): Retrieves stored procedure parameters  
/api/execute\_procedure (POST): Executes a stored procedure  
/api/execute\_view (POST): Executes a database view  
/health (GET): Checks server health status

## Troubleshooting

Issue: Database connection failed → Verify MySQL credentials in app.py  
Issue: Function or procedure not found → Ensure it exists in the bookmyshow database  
Issue: CSS not applied → Move style.css to the static directory and link properly  
Issue: Blank page → Ensure all .html files are located inside the templates folder

## Output

After successful execution, the dashboard provides:  
- A list of movies with filters and search features.  
- Function and procedure execution results displayed instantly.  
- Interactive tables showing query outputs from MySQL views.

## Conclusion

The project successfully implemented a **BookMyShow Database System** that transforms raw transactional data into meaningful insights.  
By leveraging **SQL functions, stored procedures, and views**, the system enhances:

* User engagement understanding
* Theater and movie performance tracking
* Revenue optimization
* Customer satisfaction analysis

The project demonstrates how SQL and Python integration can effectively support **data-driven decision-making** for an online entertainment platform.

## References

* + Course material: Database Concepts for Data Science
  + MySQL Official Documentation — <https://dev.mysql.com/doc/>
  + Python MySQL Connector - <https://pypi.org/project/mysql-connector-python/>
  + BookMyShow Official Platform - <https://www.bookmyshow.com/>
  + Reference freelance platforms — Upwork, Fiverr