**TABLE OF CONTENTS**

**SL. NO. CONTENTS Page No.**

**Acknowledgment** i

**Abstract** ii

**Table of Contents** 1

**1 Introduction**

1.1 Introduction 2

1.2 Problem Statement 2

**2 Literature Survey**

2.1 Existing System 4

2.2 Proposed System 4

**3 Requirement Specification**

3.1 Introduction 5

3.2 Hardware Specification 5

3.3 Software Specification 5

**4 Design**

4.1 ER Diagram 11

4.2 Schema Diagram 13

**5 System Implementation**

5.1 Introduction 14

5.2 Code Snippet 14

5.3 Insert/Update/Delete/Stored Procedure/Trigger 17

**6 Results and Snapshots** 19

**7 Conclusion and Future Enhancement**

7.1 Conclusion 21

7.2 Future Enhancement and Scope 21

**Chapter 1**

**INTRODUCTION**

* 1. **Introduction**

Welcome to newly designed movie ticket booking System is a faster, cleaner and a tad more personal GUI, specially designed to make your booking experience better. Log on, navigate and find out for yourselves and if time permits leave your valuable feedback. Customers may view the contents of any movie show at any time and may book any movie ticket as needed. The program automatically calculates the subtotal and grand total. When a visitor decides to finally book the ticket, the order information including the buyer's name, address and billing instruction is stored in the database securely and payment has been made. The combo booking is also provided at the time of booking the ticket and there’s a wonderful facility of delivering the combos at your seat when you are watching the movie.

* 1. **Problem Statement**

**Lack of immediate retrievals**

The information is very difficult to retrieve and to find particular information like- E.g. - To find out about the ticket’s history, the user has to go through various registers. This results in inconvenience and wastage of time.

**Lack of immediate information storage: -**

The information generated by various transactions takes time and efforts to be stored at right place.

**Lack of prompt updating: -**

Various changes to information like ticket details or booking details of customer are difficult to make as paper work is involved.

**Error prone manual calculation: -**

Manual calculations are error prone and take a lot of time this may result in incorrect information.

**Preparation of accurate and prompt reports: -**

This becomes a difficult task as information is difficult to collect from various register.

**Objective:-**

1) Details of movies and ticket amount

2) Registration and recording information about the Customers.

3) Keeping record of date of the booking.

4) Keeping information of the theatres available in each destination.

**Scope of the Project:-**

1) Information about Customers is done by just writing the Customer’s name, email, password, phone number and address. Whenever the Customer comes up his information is stored freshly.

2) This System provide facilities to modify and delete ticket booking as well as client data.

3) This system provides information about current movies

4) This System maintain & control the database of ticket’s information.

5) This System displays attractive tourist places and provides attractive services that matches your priorities.

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 Existing System**

This existing system is not providing secure registration and profile management of all the users properly. This manual system gives us very less security for saving data and some data may be lost due to mismanagement. The system is giving manual information through the tourism agent.

**2.2 Proposed System**

The development of this new system contains the following activities, which try to automate the entire process keeping in the view of database integration approach. The system maintains customer’s details, movie details, ticket details and booking details. This system will provide good search capabilities. User friendliness is provided in the application with various controls provided for system rich user interfaces. Separate authentication for customer and administrator is provided in this application. Both admin and the user can access the system.

**CHAPTER 3**

**REQUIREMENT SPECIFICATION**

**3.1 Introduction**

To be used efficiently, all computer software needs certain hardware components or the other software resources to be present on a computer. These pre-requisites are known as(computer) system requirements and are often used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements: minimum and recommended. With increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time. Industry analysts suggest that this trend plays a bigger part in driving upgrades to existing computer systems than technological advancements.

**3.2 Hardware Requirements**

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatibility and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

**Hardware Requirements for present project**

PROCESSOR : Intel core i5

RAM : 4 GB

HARD DISK : 80 GB

**3.3 Software Requirements**

Software Requirements deal with defining software resource requirements and pre-requisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or pre-requisites are generally not included in the software installati on package and need to be installed separately before the software is installed.

**Software Requirements for present project**

OPERATING SYSTEM : Windows 10

FRONT END : Python (Tkinter module)

BACK END : SQLite

**Front End**

Python: Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was created in the late 1980s, and first released in 1991, by Guido van Rossum as a successor to the ABC programming language. Python 2.0, released in 2000, introduced new features, such as list comprehensions, and a garbage collection system with reference counting, and was discontinued with version 2.7 in 2020.[30] Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible and much Python 2 code does not run unmodified on Python 3. With Python 2's end-of-life, only Python 3.6.x[31] and later are supported, with older versions still supporting e.g., Windows 7 (and old installers not restricted to 64-bit Windows).

Python interpreters are supported for mainstream operating systems and available for a few more (and in the past supported many more). A global community of programmers develops and maintains CPython, a free and open-source [32] reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development

**Tkinter**

**Graphical User Interface (GUI)** is a form of user interface which allows users to interact with computers through visual indicators using items such as icons, menus, windows, etc. It has advantages over the Command Line Interface (CLI) where users interact with computers by writing commands using keyboard only and whose usage is more difficult than GUI.

Tkinter is the inbuilt python module that is used to create GUIapplications. It is one of the most commonly used modules for creating GUI applications in Python as it is simple and easy to work with. You don’t need to worry about the installation of the Tkinter module separately as it comes with Python already. It gives an object-oriented interface to the Tk GUI toolkit.

Some other Python Libraries available for creating our own GUI applications are

• Kivy

• Python Qt

• wxPython

Among all **Tkinter** is most widely used

In Python, Tkinter is a standard GUI (graphical user interface) package. Tkinter is Python's default GUI module and also the most common way that is used for GUI programming in Python. Note that Tkinter is a set of wrappers that implement the Tk widgets as Python classes.

Tkinter in Python helps in creating GUI Applications with a minimum hassle. Among various GUI Frameworks, Tkinter is the only framework that is built-in into **Python's Standard Library.**

• An important feature in favour of Tkinter is that it is cross-platform, so the same code can easily work on **Windows**, **macOS**, and **Linux.**

• Tkinter is a lightweight module.

• It is simple to use.

**Back End (SQLite) Database**

About SQLite: SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is a database, which is zero-configured, which means like other databases you do not need to configure it in your system.

SQLite engine is not a standalone process like other databases, you can link it statically or dynamically as per your requirement with your application. SQLite accesses its storage files directly

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is one of the fastest-growing database engines around, but that's growth in terms of popularity, not anything to do with its size. The source code for SQLite is in the public domain.

**Why SQLite?**

* SQLite does not require a separate server process or system to operate (serverless).
* SQLite comes with zero-configuration, which means no setup or administration needed.
* A complete SQLite database is stored in a single cross-platform disk file.
* SQLite is very small and light weight, less than 400KiB fully configured or less than 250KiB with optional features omitted.
* SQLite is self-contained, which means no external dependencies.
* SQLite transactions are fully ACID-compliant, allowing safe access from multiple processes or threads.
* SQLite supports most of the query language features found in SQL92 (SQL2) standard.
* SQLite is written in ANSI-C and provides simple and easy-to-use API.
* SQLite is available on UNIX (Linux, Mac OS-X, Android, iOS) and Windows (Win32, WinCE, WinRT).
* SQLite is very small and light weight, less than 400KiB fully configured or less than 250KiB with optional features omitted.
* SQLite is self-contained, which means no external dependencies.
* SQLite transactions are fully ACID-compliant, allowing safe access from multiple processes or threads.
* SQLite supports most of the query language features found in SQL92 (SQL2) standard.
* SQLite is written in ANSI-C and provides simple and easy-to-use API.
* SQLite is available on UNIX (Linux, Mac OS-X, Android, iOS) and Windows (Win32, WinCE, WinRT).

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. SQLite database files are a recommended storage format by the US Library of Congress. Think of SQLite not as a replacement for Oracle but as a replacement for fopen()

SQLite is a compact library. With all features enabled, the library size can be less than 600KiB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger.) There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments. Depending on how it is used, SQLite can be faster than direct filesystem I/O.

SQLite is very carefully tested prior to every release and has a reputation for being very reliable. Most of the SQLite source code is devoted purely to testing and verification. An automated test suite runs millions and millions of test cases involving hundreds of millions of individual SQL statements and achieves 100% branch test coverage. SQLite responds gracefully to memory allocation failures and disk I/O errors. Transactions are ACID even if interrupted by system crashes or power failures.

**CHAPTER 4**

**DESIGN**

**4.1 ER Diagram**

This ER (Entity Relationship) Diagram represent the model of Movie Ticket Booking System Entity. The entity-relationship diagram of Movie Ticket Booking System Show all the visual instrument of database tables and the relation between Customer, Payment, Movie, Shows etc. It used structure data and to define the relationships between structured data groups of Movie Ticket Booking System functionalities. The main entities of the Movie Ticket Booking System are Movie, Customer, Payments, seats and Shows

Movie Ticket Booking System entities and their attributes:

• Movie info Entity: Attributes of Movie are Movie\_id, Movie\_name, Director, Duration, Budget, Cast, Rating

• Customer info Entity: Attributes of Customer are Customer id Customer Name, Customer Email, Customer phono

• Ticket info Entity: Attributes of Ticket info are Ticket No, Movie\_name, Price, Seat No, Show date

• Booking Entity: Attributes of Booking are Ticket No, Customer id Show date

Description of Movie Ticket Booking System Database:

* The details of Movie are store into the Movie tables respective with all tables
* Each entity (Movie\_id, Customer id, Ticket no) contain primary key and unique keys.
* The entity Booking, Ticket has binded with Movie, Customer entity with foreign key
* There is one-to-one relationship available between Tickets, Movie info
* All the entity Movie info, Ticket info, Booking are normalized and reduce delicacy of record.
* We have implemented indexing on each tables of Movie Ticket Booking System table for fast query execution.



Fig 4.1: E R Diagram of Online Movie Ticket Booking System

**4.2 Schema Diagram**

Movie info:

|  |
| --- |
| Movie id Movie Name Director Duration Budget Cast Rating |

Customer info:

|  |
| --- |
| Customer id Customer name Customer email Customer Phone |

Ticket info:

|  |
| --- |
| Ticket no Movie id Seat no Show date Price |

Booking:

|  |
| --- |
| Customer id Ticket No Show date |

Fig 4.2: Schema Diagram for Online Movie Ticket Booking System

**CHAPTER 5**

**SYSTEM IMPLEMENTATION**

**5.1 Introduction**

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

The implementation stage involves careful planning, investigation of the existing system and it’s constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods.

**5.2 Code Snippet**

**CREATE TABLE IF NOT EXISTS** customer (

cid INTEGER PRIMARY KEY,

c\_name text,

email\_id text,

phone\_no text

);

**CREATE TABLE IF NOT EXISTS** movie\_data(

m\_id INTEGER PRIMARY KEY,

m\_name text,

release\_date text,

director text,

actors text,

budget integer,

duration int,

rating int

);

**CREATE TABLE IF NOT EXISTS** tickets(

ticket\_no INTEGER PRIMARY KEY,

m\_name text,

price int,

seat\_no int,

show\_date text

);

**CREATE TABLE IF NOT EXISTS** booking(

cid INTEGER,

ticket\_no INTEGER,

show\_date text,

FOREIGN KEY(cid) REFERENCES customer(cid),

FOREIGN KEY(ticket\_no) REFERENCES tickets(ticket\_no),

FOREIGN KEY(show\_date) REFERENCES tickets(show\_date)

PRIMARY KEY(cid,ticket\_no)

);

**Description of Customer table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Column | Column type | Length | Constraint |
| Customer | Customer\_id | NUMBER | - | P\_KEY |
|  | Customer\_name | VARCHAR | 30 |  |
|  | Email\_id | VARCHAR | 30 |  |
|  | Phone\_no | NUMBER | 10 |  |

**Description of Movie table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Column | Column type | Length | Constraint |
| Movie data | Movie\_id | NUMBER | - | P\_KEY |
|  | Movie\_name | VARCHAR | 20 |  |
|  | Release\_date | DATE | - |  |
|  | Director | VARCHAR | 20 |  |
|  | Actor | VARCHAR | 20 |  |
|  | Budget | NUMBER | - |  |
|  | Duration | NUMBER | - |  |
|  | Rating | NUMBER | - |  |

**Description of Movie table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Column | Column type | Length | Constraint |
| Tickets | Tiket\_no | NUMBER | - | P\_KEY |
|  | Movie\_name | VARCHAR | 20 |  |
|  | Price | NUMBER | - |  |
|  | Seat\_no | NUMBER | - |  |
|  | Show\_date | DATE | - |  |

**Description of Booking table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Column | Column type | Length | Constraint |
| Booking | Custometr\_id | NUMBER | - | P\_KEY |
|  | Tiket\_no | NUMBER | - |  |
|  | Show\_date | DATE | - |  |

**5.3 Insert, Update, Delete, Stored Procedure**

**Code snippet for Insert**

def addMovie(self,m\_id,m\_name,release\_date,director,actors,budget,duration,rating):

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

c = conn.cursor()

c.execute('INSERT INTO movie\_data VALUES(?,?,?,?,?,?,?,?)',(m\_id,m\_name,release\_date,director,actors,budget,duration,rating))

conn.commit()

conn.close()

**Code snippet for Delete**

def DeleteMovieRec(self,m\_id):

conn=sqlite3.connect("movie.db")

c=conn.cursor()

c.execute("DELETE FROM movie\_data WHERE m\_id=?", (m\_id,))

conn.commit()

conn.close()

**Code snippet for Retrieve the Stored data Procedure**

def ViewMovieData(self):

conn=sqlite3.connect("movie.db")

c=conn.cursor()

c.execute("SELECT \* FROM movie\_data")

rows=c.fetchall()

conn.commit()

conn.close()

return rows

def ViewTicketsData(self):

conn=sqlite3.connect("movie.db")

c=conn.cursor()

c.execute("SELECT \* FROM tickets")

rows=c.fetchall()

conn.commit()

conn.close()

return rows

def ViewBookingRec(self):

conn=sqlite3.connect("movie.db")

c=conn.cursor()

c.execute("""SELECT c.cid,t.ticket\_no,t.show\_date

FROM customer c,tickets t

""")

rows = c.fetchall()

conn.commit()

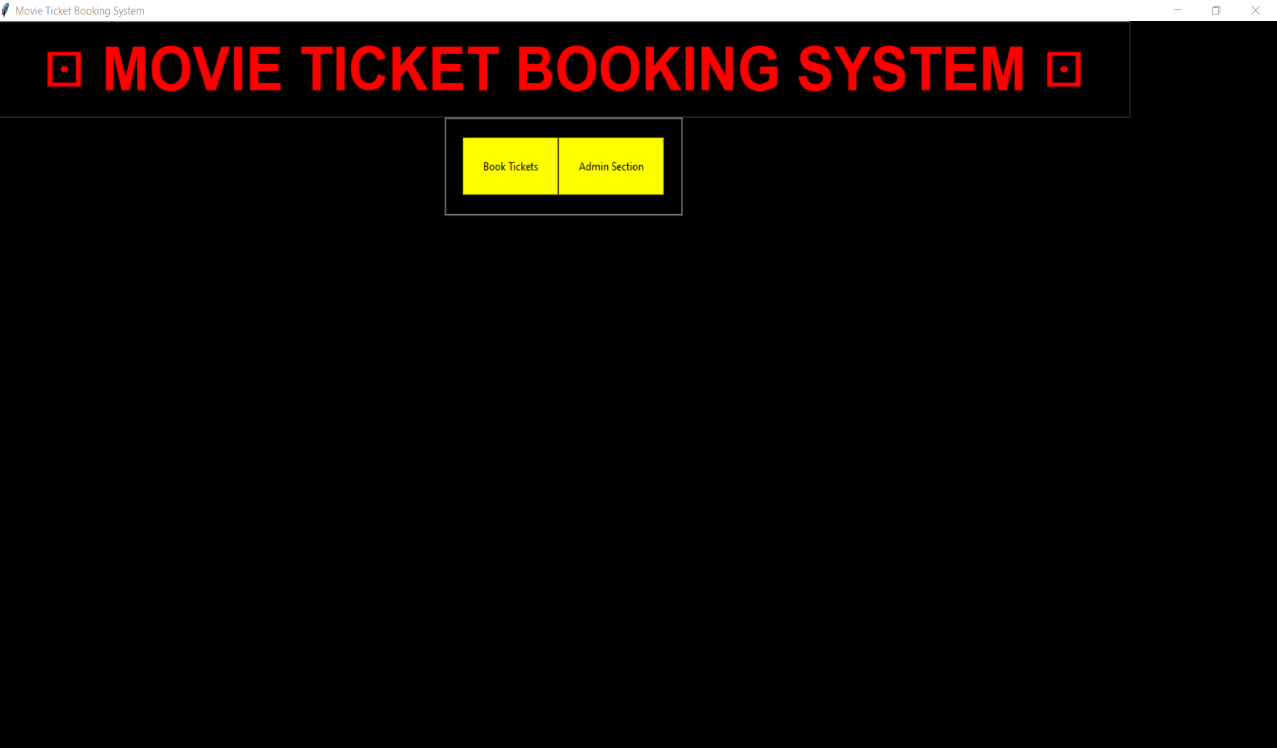
conn.close()

return rows

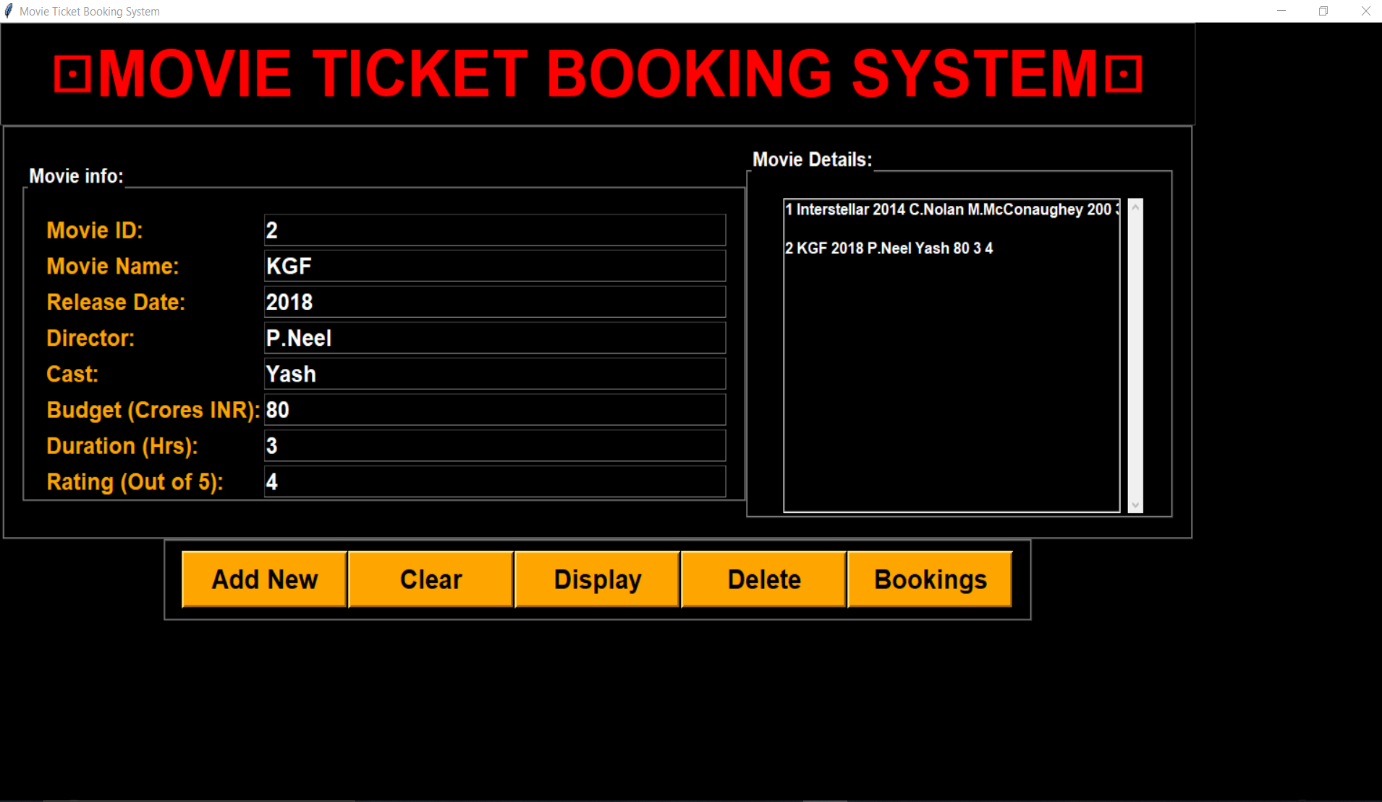
**CHAPTER 6**

**RESULTS AND SNAPSHOTS**

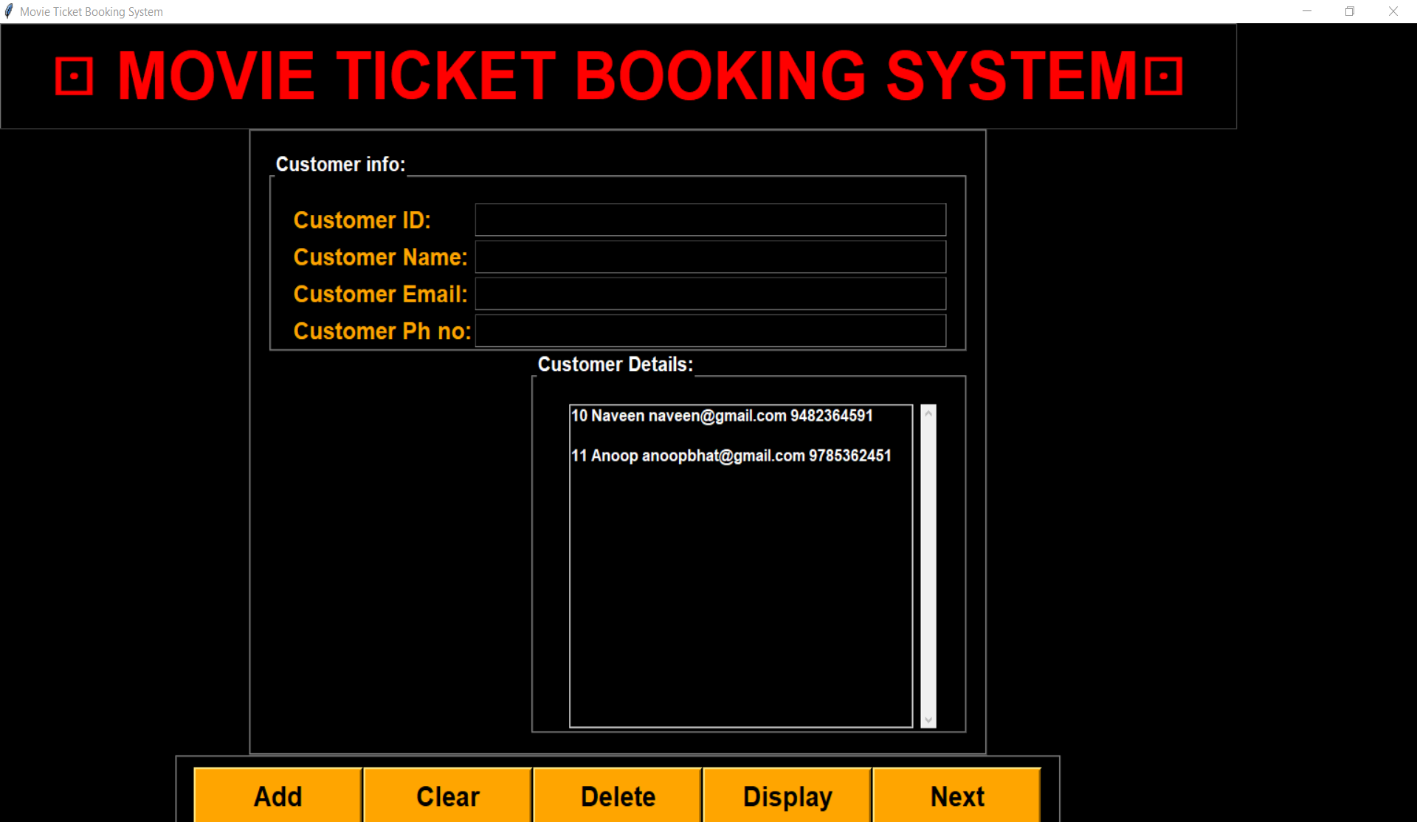
**Snapshot 1: This snapshot shows the main home page.**

****

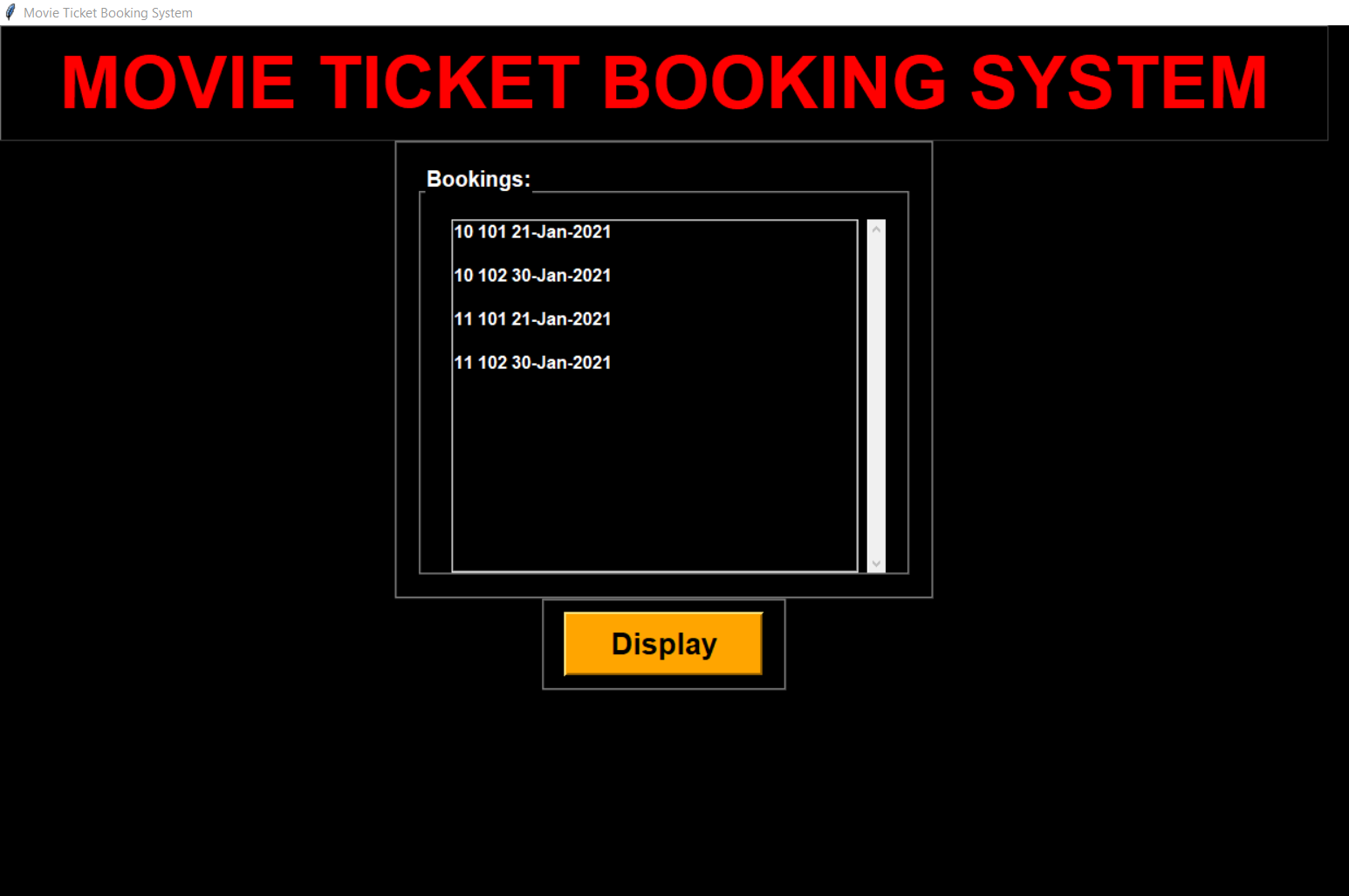
**Snapshot 2:This snapshot shows the Admin to** **perform insert,update,delete Movies.**

****

**Snapshot 3: This snapshot shows the Customer Registration .**



**Snapshot 4: This snapshot shows the whoever Booking.**



**CHAPTER 7**

**CONCLUSION AND FUTURE ENHANCEMENT**

**7.1 Conclusion**

This project is developed successfully and the performance is found to be satisfactory. This project is designed to meet the requirements of assigning jobs. It has been developed in Python and the database has been built in SQLite keeping in mind the specifications of the system. The user will be able to book the ticket using this GUI. The relationship between company manager, employee, and customer satisfy a good communication to complete ticketing process.

We have designed the project to provide the user with easy retrieval of data, details of theatre and necessary feedback as much as possible. In this project, the user is provided with a GUI that can be used to book movie tickets online. To implement this as a GUI application we used Python as the technology. SQLite has advantages such as enhanced performance, scalability, built- in security and simplicity. To build any GUI application using Python we need a programming language such as Python and so on. SQLite was used as back-end database since it is one of the most popular opensource databases, and it provides fast data access, easy installation and simplicity. For front end we used Python and Tkinter.

**7.2 Future Enhancement and Scope**

In the future enhancements of this project entitled “Online movie ticket booking”, we would like to include facilities like,

1. Providing theatres arrangements.

2. Making it as online system.

3. Providing Booking all over the world.