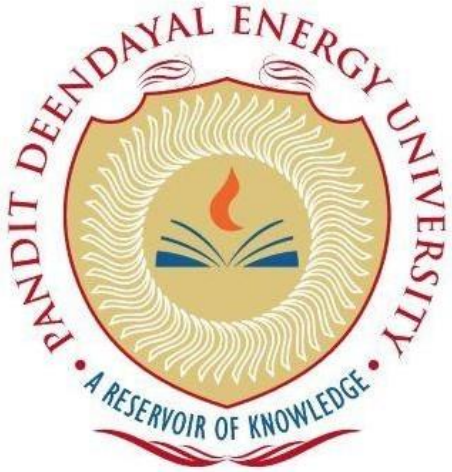
PANDIT DEENDAYAL ENERGY UNIVERSITYSCHOOL OF TECHNOLOGY



Course: Database Management SystemsCourse Code: 20CP208P

Project

B.Tech. (Computer Science and Engineering)

Semester 4

Submitted To:

Dr. Hargeet Kaur

Submitted By:

Jainil Doshi

Rashmi Panchal

Jugal Chhatriwala

G7 Batch

Table of Contents

Acknowledgement

It is with great pleasure that we express our deepest gratitude to Dr. Hargget Kaur, the coordinator of our DBMS project, for her invaluable guidance and support throughout the project. Her invaluable guidance and support throughout the development of our movie ticket database management system. Her expertise and insightful feedback have been instrumental in the successful completion of this project. We could not have done it without her mentorship.

Our project aimed to create a database management system that would allow movie theaters to efficiently manage their ticket bookings. We started with an Entity Relationship Diagram and a Data Dictionary to design the database, followed by the creation of the database and data insertion. We then focused on database management, including security, backup and recovery, performance tuning, and maintenance.

Throughout this process, Dr. Kaur provided us with valuable advice and guidance. Her expertise in database management allowed us to create a system that met our objectives and could benefit movie theaters. We are grateful for her unwavering support and mentorship.

We would also like to extend our thanks to the team members who contributed their time and effort to this project. Their hard work and dedication have been crucial in achieving our objectives. Their contributions have been essential to the success of this project.

We hope that this system will be useful to those who seek to manage movie bookings efficiently. Our goal was to create a user-friendly, efficient, and secure system that could be easily adopted by movie theaters. We look forward to continuing to improve and refine this system in the future.

In conclusion, we are grateful for the support and guidance we received during the development of this movie ticket database management system. We hope that it will be useful to those in the movie theater industry and look forward to continuing to improve and refine it in the future.

Abstract

This report discusses the development of a movie ticket database management system that aims to provide a user-friendly, efficient, and secure solution for movie theaters to manage their ticket bookings. The system was designed using an Entity Relationship Diagram and Data Dictionary and implemented using various database management techniques. The report provides an overview of the project's objectives, methodology, and achievements.

The Movie Ticket Booking DBMS project is an innovative system designed to streamline the process of booking and managing movie tickets. The project aims to provide a user-friendly interface for customers to book their movie tickets online, while also providing cinema owners with an efficient system for managing ticket sales, scheduling movie screenings, and tracking revenue.

The system is designed using the principles of database management systems and employs modern web technologies to provide a seamless user experience. Customers can search for movies by genre, language, or location, and easily book tickets through a secure online payment system. Cinema owners can manage their movie schedules, assign seats, and track ticket sales and revenue through an intuitive dashboard.

The project provides a robust and scalable solution for movie ticket booking, which can be easily customized and integrated into existing cinema management systems. Overall, the Movie Ticket Booking DBMS project is a significant step forward in the automation of cinema operations and represents a major contribution to the field of database management systems.

Introduction

In today's fast-paced world, people are increasingly relying on online platforms for convenience and ease of access. The movie industry is no exception, with online ticket booking becoming the preferred mode of ticket purchase for moviegoers. However, with the rise in demand for online ticket booking, there is a need for a reliable and efficient system to manage the booking process.

The Movie Ticket Booking DBMS project is a web-based application that provides a solution to this need. The project aims to provide a user-friendly interface for customers to book their movie tickets online, while also providing cinema owners with an efficient system for managing ticket sales, scheduling movie screenings, and tracking revenue.

The system's design is discussed in detail, including the database schema and various tables used to store data related to movies, theaters, shows, and bookings. The report also discusses the system's implementation, including the use of SQL queries to retrieve and manipulate data.

The system is designed using the principles of database management systems and modern web technologies, making it highly scalable, customizable, and easy to integrate with existing cinema management systems. The project offers a centralized platform for cinema owners to manage all aspects of their cinema operations, from movie scheduling to ticket sales and revenue tracking.

The system is designed to provide a seamless user experience for customers, with an intuitive and easy-to-use interface. Customers can search for movies by genre, language, or location, and easily book tickets through a secure online payment system. The system provides real-time updates on movie schedules, ticket availability, and seat assignments, ensuring a hassle-free experience for moviegoers.

Cinema owners can also benefit from the system's advanced features, including the ability to manage movie schedules, assign seats, and track ticket sales and revenue. The system provides an intuitive dashboard for cinema owners to monitor their cinema operations, enabling them to make informed decisions and optimize their revenue streams.

Overall, the Movie Ticket Booking DBMS project represents a significant step forward in the automation of cinema operations, providing a robust and scalable solution for movie ticket booking. The project is a major contribution to the field of database management systems, offering a comprehensive solution for managing cinema operations in the digital age.

Project Description

The Movie Ticket Booking DBMS project is a web-based application designed to provide a user-friendly platform for customers to book movie tickets online, while also offering cinema owners a reliable and efficient system for managing their cinema operations.

The project is built using the principles of database management systems, employing modern web technologies to provide a robust and scalable solution for movie ticket booking. The system is designed to cater to the needs of both customers and cinema owners, offering a centralized platform for managing all aspects of cinema operations.

The system provides a user-friendly interface for customers to search for movies, view movie schedules, and book tickets online. Customers can search for movies based on their preferred genre, language, or location, making it easy to find the movie they want to watch. The system provides real-time updates on movie schedules and ticket availability, ensuring that customers have access to the latest information.

The system also offers a secure online payment system, enabling customers to book their movie tickets from the comfort of their own homes. The system provides a seamless user experience, making the ticket booking process quick and easy.

For cinema owners, the system offers a range of advanced features, including the ability to manage movie schedules, assign seats, and track ticket sales and revenue. The system provides an intuitive dashboard for cinema owners to monitor their cinema operations, enabling them to make informed decisions and optimize their revenue streams.

The system also provides an efficient system for managing movie screenings, enabling cinema owners to schedule movies and assign seats with ease. The system is designed to be highly scalable, making it easy to integrate with existing cinema management systems.

In summary, the Movie Ticket Booking DBMS project is a comprehensive solution for managing cinema operations in the digital age. The system provides a user-friendly platform for customers to book movie tickets online, while also offering cinema owners a reliable and efficient system for managing their cinema operations. The project is a major contribution to the field of database management systems, providing a robust and scalable solution for movie ticket booking.

List Of Entity & Attributes

1) **Theatre:**

- Theatre D VARCHAR(5)
- Name of Theatre VARCHAR(30)
- No of Screens INT
- Are VARCHAR(30)

2) **Screen:**

- Screen ID VARCHAR(5)
- No of Seats Gold INT
- No of Seats Silver INT
- Theatre ID VARCHAR(S)

3) **Movie :**

- Movie ID VARCHAR(S)
- Name VARCHAR(30)
- Language VARCHAR(10)
- Genre VARCHAR(20)
- Target Audience VARCHAR(S)

4) **User:**

- User ID VARCHAR(5)
- First Name VARCHAR(15)
- Last Name VARCHAR(20)
- Booking ID VARCHAR(10)
- Email ID VARCHAR(30)
- Age INT
- Phone Number VARCHAR(10)

5) Shows:

- Show ID VARCHAR(10)
- Show Time TIME
- Show Date DATE
- Seats Remaining Gold INT
- Seats Remaining Silver INTH
- Chat Cost Gold INT
- Cas Cost Siver INT
- Screen ID VARCHAR(5)
- Movie_ID VARCHAR(S)

6) Booking:

- Booking ID VARCHAR(10)
- No of Tickets INT
- Total Cost INT
- KO Card Number VARCHAR(29)
- Name on card VARCHAR(21)
- User_ID VARCHAR(S)
- Show ID VARCHAR(10)

7) Ticket:

- Ticket ID VARCHAR(20)
- Class VARCHAR(3)
- Price INT

Entity – Relationship Model

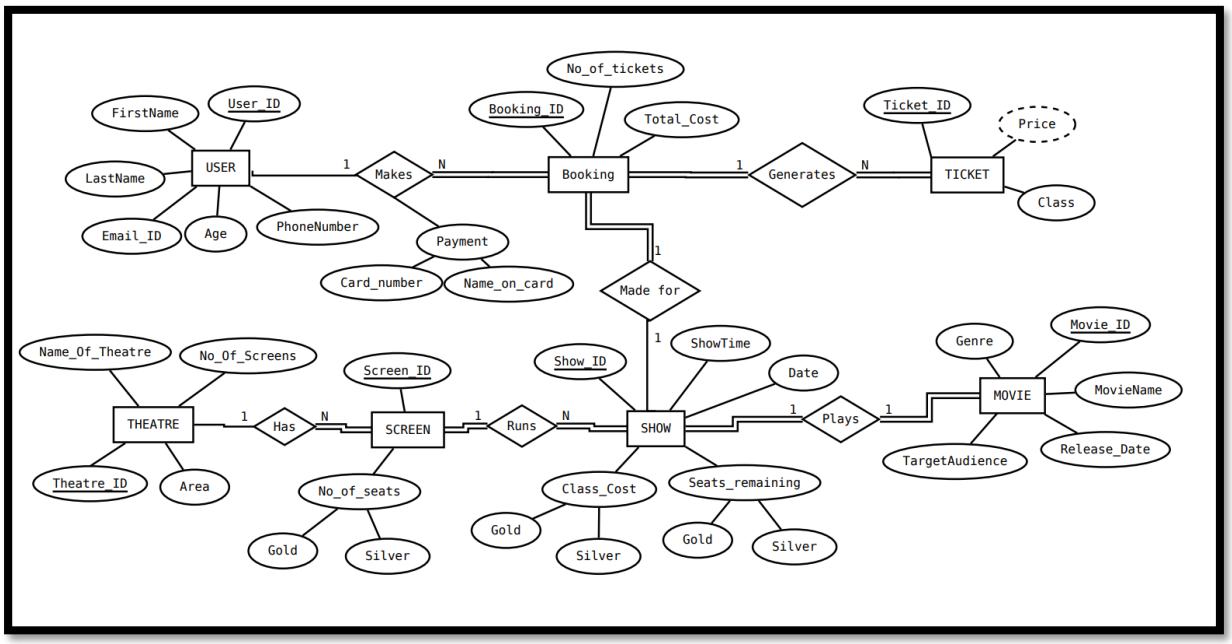
An Entity-Relationship (ER) model is important in a Movie Booking Database because it helps in understanding the relationships between the different entities involved in the database, such as users, movies, theaters, screens, shows, and bookings.

An ER model provides a visual representation of the database schema, showing the entities, attributes, and relationships among them. It helps in identifying the key entities and the relationships between them. The ER model can also help in detecting any potential issues with the database design before implementation.

In the context of a Movie Booking Database, an ER model helps in understanding the relationships between movies, theaters, screens, shows, and bookings. For example, it can help in identifying the theaters where a particular movie is being screened, the screens in each theater, the shows scheduled for each screen, and the bookings made for each show.

In summary, an ER model helps in organizing and visualizing the complex relationships between entities in a Movie Booking Database, making it easier to understand and maintain the database.

ER- Diagram :



Cardinality constraints:

- Theatre: One theatre can have multiple screens (One-to-Many)
- Screen: One screen belongs to one theatre (Many-to-One), one screen can have multiple shows (One-to-Many)
- Movie: One movie can have multiple shows (One-to-Many)
- Shows: One show can have multiple bookings (One-to-Many), one show is associated with one screen (Many-to-One), one show is associated with one movie (Many-to-One)
- Booking: One booking is associated with one show (Many-to-One), one booking is associated with one user (Many-to-One), one booking can have multiple tickets (One-to-Many)
- Ticket: One ticket is associated with one booking (Many-to-One)
- User: One user can have multiple bookings (One-to-Many)

Key constraints:

- Theatre: D (Primary key)
- Screen: Screen_ID (Primary key), Theatre_ID (Foreign key referencing Theatre)
- Movie: Movie_ID (Primary key)
- Shows: Show_ID (Primary key), Screen_ID (Foreign key referencing Screen), Movie_ID (Foreign key referencing Movie)
- Booking: Booking_ID (Primary key), User_ID (Foreign key referencing User), Show_ID (Foreign key referencing Shows)
- Ticket: Ticket_ID (Primary key)
- User: User_ID (Primary key)

Schema Diagram

A schema is important in a Movie Booking Database because it defines the structure and relationships between different entities (tables) in the database. It provides a clear understanding of how the data is organized and how the different tables are related to each other.

In the context of a Movie Booking Database, a well-designed schema ensures that the data is consistent, accurate, and easy to manage. It allows for efficient querying of the data and facilitates the maintenance and updating of the database. Additionally, a well-designed schema helps to prevent data duplication and ensures that the database remains scalable.

Furthermore, a well-defined schema helps developers to understand the requirements of the system and build the application accordingly. It also helps to maintain the integrity and security of the data by defining constraints, such as primary keys, foreign keys, and other data validation rules.

In addition to the above benefits, a well-designed schema also helps to reduce errors and inconsistencies in the data. By defining the relationships between different entities, a schema helps to ensure that the data is properly normalized, which makes it easier to update and maintain the database.

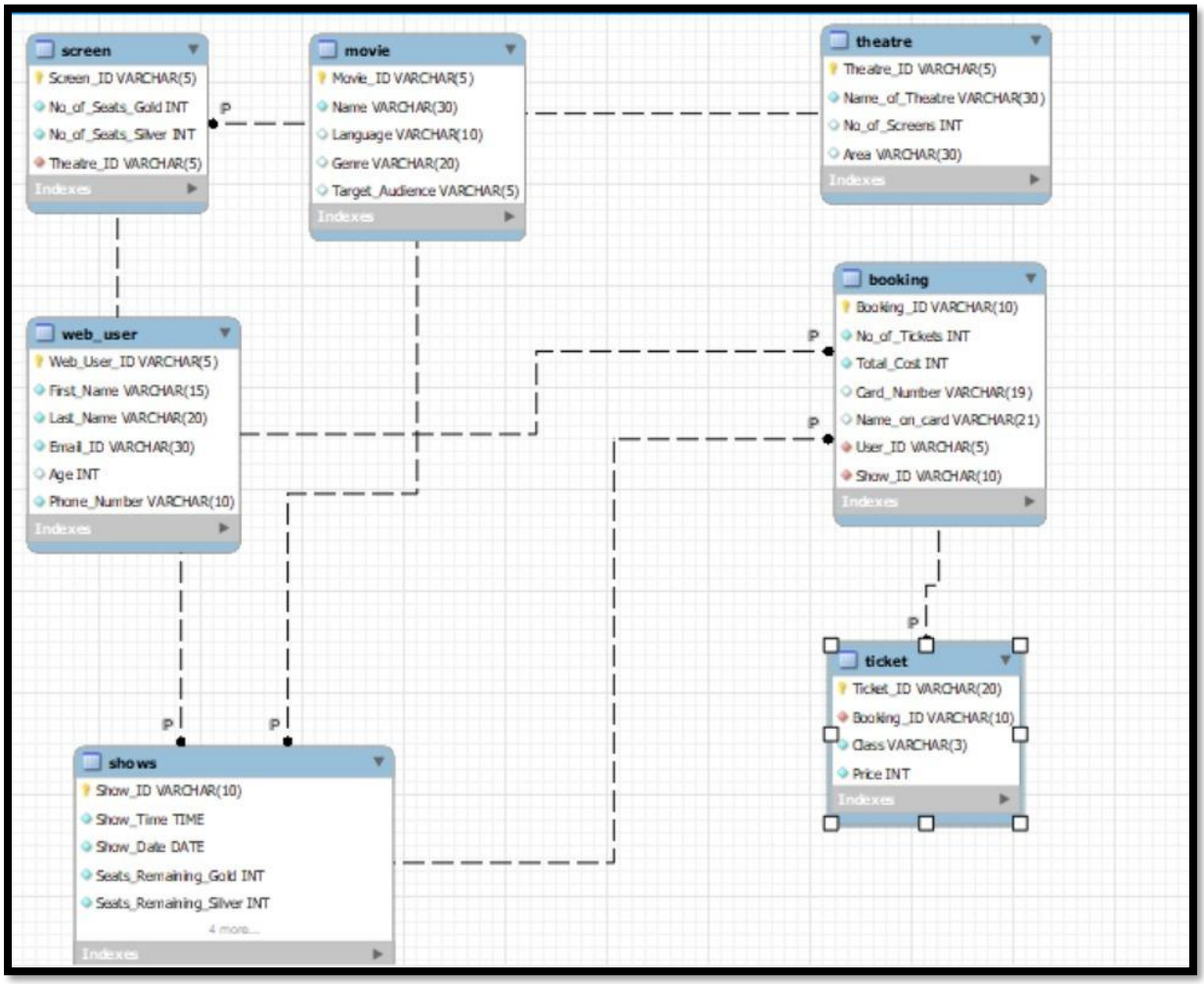
Moreover, a schema allows developers to easily understand and modify the database structure as needed. It also provides a clear understanding of the business logic and data flow, which makes it easier to develop new features and functionalities.

A well-defined schema also facilitates integration with other systems and applications. For example, if the movie booking system needs to integrate with a payment gateway or an inventory management system, having a clear schema will make it easier to map the data and ensure that the integration is seamless.

Overall, having a well-designed schema is critical to the success of a movie booking database as it helps to ensure the accuracy, consistency, and integrity of the data. It also facilitates the development, maintenance, and integration of the database and ensures compliance with industry standards and regulations.

In summary, a schema is essential in a Movie Booking Database to ensure that the data is well-organized, consistent, and easy to manage. It also helps to maintain the integrity and security of the data and facilitates the development and maintenance of the application.

Diagram:


screen
Screen_ID VARCHAR(5)
No_of_Seats_Gold INT
No_of_Seats_Silver INT
Theatre_ID VARCHAR(5)
Indexes
movie
Movie_ID VARCHAR(5)
Name VARCHAR(30)
Language VARCHAR(10)
Genre VARCHAR(20)
Target_Audience VARCHAR(5)
Indexes
theatre
Theatre_ID VARCHAR(5)
Name_of_Theatre VARCHAR(30)
No_of_Screens INT
Area VARCHAR(30)
Indexes
booking
Booking_ID VARCHAR(10)
No_of_Tickets INT
Total_Cost INT
Card_Number VARCHAR(19)
Name_on_card VARCHAR(21)
User_ID VARCHAR(5)
Show_ID VARCHAR(10)
Indexes
web_user
Web_User_ID VARCHAR(5)
First_Name VARCHAR(15)
Last_Name VARCHAR(20)
Email_ID VARCHAR(30)
Age INT
Phone_Number VARCHAR(10)
Indexes
ticket
Ticket_ID VARCHAR(20)
Booking_ID VARCHAR(10)
Class VARCHAR(3)
Price INT
Indexes
shows
Show_ID VARCHAR(10)
Show_Time TIME
Show_Date DATE
Seats_Remaining_Gold INT
Seats_Remaining_Silver INT
4 more...
Indexes

Relational - Model

The Relational Model is important in a Movie Booking Database because it allows for the efficient organization and retrieval of data related to the various entities and their relationships in the system.

In a movie booking database, there are multiple entities such as theatre, screen, movie, show, booking, ticket, and user, each with their own attributes and relationships with other entities. By using the relational model, we can create tables for each entity and define the relationships between them using foreign keys. This ensures data consistency and integrity, as the database management system can enforce referential integrity constraints to prevent invalid data entries or updates.

One of the key benefits of the relational model is that it allows for data to be stored in a normalized form, which reduces data redundancy and inconsistencies. This ensures that data is accurate and up-to-date, which is crucial for movie booking systems that require real-time updates of showtimes, theaters, and customer information.

The relational model also allows for the creation of complex queries that can retrieve data from multiple tables, which is important for generating reports and analyzing trends. It provides a flexible and scalable framework for managing large amounts of data, which is essential for movie booking systems that handle thousands of transactions every day.

In addition, the relational model supports the use of constraints and rules that ensure data integrity and consistency. This helps to prevent errors and inconsistencies that can arise from data entry errors or software bugs.

The relational model also facilitates data sharing and collaboration among different users and departments within an organization. It allows for the creation of user roles and permissions, which ensures that data is only accessible and modifiable by authorized users.

The relational model is an essential component of any movie booking system that requires efficient and effective management of complex data structures. It provides a flexible, scalable, and secure framework for managing data, and supports efficient querying, analysis, and reporting of data.

In conclusion, without the relational model, it would be much more difficult to efficiently manage and analyze data in a movie booking system. Therefore, it is imperative to use this model for the successful development and management of movie booking systems.

Relational Model:

Theatre (Theatre_ID VARCHAR(5), Name VARCHAR(30), No_of_Screens INT, Area VARCHAR(30))

Screen (Screen_ID VARCHAR(5), No_of_Seats_Gold INT, No_of_Seats_Silver INT, Theatre_ID VARCHAR(5), FOREIGN KEY (Theatre_ID) REFERENCES Theatre(Theatre_ID))

Movie (Movie_ID VARCHAR(5), Name VARCHAR(30), Language VARCHAR(10), Genre VARCHAR(20), Target_Audience VARCHAR(30))

Show (Show_ID VARCHAR(10), Show_Time TIME, Show_Date DATE, Seats_Remaining_Gold INT, Seats_Remaining_Silver INT, Chat_Cost_Gold INT, Cas_Cost_Silver INT, Screen_ID VARCHAR(5), Movie_ID VARCHAR(5), FOREIGN KEY (Screen_ID) REFERENCES Screen(Screen_ID), FOREIGN KEY (Movie_ID) REFERENCES Movie(Movie_ID))

Booking (Booking_ID VARCHAR(10), No_of_Tickets INT, Total_Cost INT, KO_Card_Number VARCHAR(29), Name_on_card VARCHAR(21), User_ID VARCHAR(5), Show_ID VARCHAR(10), FOREIGN KEY (User_ID) REFERENCES User(User_ID), FOREIGN KEY (Show_ID) REFERENCES Show(Show_ID))

Ticket (Ticket_ID VARCHAR(20), Class VARCHAR(3), Price INT)

User (User_ID VARCHAR(5), First_Name VARCHAR(15), Last_Name VARCHAR(20), Email_ID VARCHAR(30), Age INT, Phone_Number VARCHAR(10), Booking_ID VARCHAR(10), FOREIGN KEY (Booking_ID) REFERENCES Booking(Booking_ID))

Normalization

Normalization is the process of organizing data in a database to eliminate redundancy and dependency issues, resulting in a more efficient and flexible database. Normalization helps to ensure that the data in the database is consistent and accurate, reduces the chances of data inconsistencies, and makes it easier to maintain the database over time.

In the context of a movie booking ticket system, normalization is crucial to ensure that the data is organized efficiently and that there are no data integrity issues. Below are the different types of normalization and why they are needed in a movie booking ticket system.

First Normal Form (1NF): The first normal form requires that each table in the database has a primary key and that every column in the table is atomic, meaning it contains only one piece of information. In a movie booking ticket system, this means that each table in the database, such as the booking table or the user table, should have a unique primary key and that each column should contain only one piece of information.

Second Normal Form (2NF): The second normal form requires that each non-key column in a table is functionally dependent on the primary key. In other words, any column that is not part of the primary key should depend on the primary key. In a movie booking ticket system, this means that the columns in the booking table that are not part of the primary key, such as the number of tickets or the total cost, should depend on the booking ID, which is the primary key.

Third Normal Form (3NF): The third normal form requires that each non-key column in a table is not transitively dependent on the primary key. This means that any column that is not part of the primary key should not depend on any other non-key column in the table. In a movie booking ticket system, this means that the columns in the show table that are not part of the primary key, such as the number of seats remaining or the chat cost, should not depend on any other non-key column in the table.

Normalization is crucial in a movie booking ticket system to ensure that data is organized efficiently and that there are no data inconsistencies. By applying the different normalization rules, we can create a more efficient and flexible database that is easier to maintain and manage over time.

- ***Here is Our Schema's Normalization steps Given***

1. To normalize the given tables into 1NF, we need to eliminate any repeating groups or multivalued attributes.

The initial schema is:

theatre (Theatre ID, Name of Theatre, No of Screens, Area)

screen (Screen ID, No of Seats Gold, No of Seats Silver, Theatre ID)

movie (Movie ID, Name, Language, Genre, Target Audience)

shows (Show ID, Show Time, Show Date, Seats Remaining Gold, Seats Remaining Silver, Chat Cost Gold, Cas Cost Silver, Screen ID, Movie_ID)

booking (Booking ID, No of Tickets, Total Cost, KO Card Number, Name on card, User_ID, Show ID)

ticket (Ticket ID, Class, Price)

user (User ID, First Name, Last Name, Booking ID, Email ID, Age, Phone Number)

In the theatre table, "No of Screens" is a multivalued attribute, which violates the first normal form (1NF). We need to remove it and create a new table for screens.

The new schema in 1NF is:

theatre (Theatre ID, Name of Theatre, Area)

screen (Screen ID, No of Seats Gold, No of Seats Silver, Theatre ID)

movie (Movie ID, Name, Language, Genre, Target Audience)

shows (Show ID, Show Time, Show Date, Seats Remaining Gold, Seats Remaining Silver, Chat Cost Gold, Cas Cost Silver, Screen ID, Movie_ID)

booking (Booking ID, No of Tickets, Total Cost, KO Card Number, Name on card, User_ID, Show ID)

ticket (Ticket ID, Class, Price)

user (User ID, First Name, Last Name, Booking ID, Email ID, Age, Phone Number)

Now we have a separate table for screens, which includes the number of seats for each screen. This eliminates the multivalued attribute in the theatre table, and the schema is now in the first normal form.

2. To convert the schema into second normal form (2NF), we need to ensure that every non-key attribute depends on the entire primary key. In other words, there should not be any partial dependencies.

The primary key for each table is:

theatre (Theatre ID)

screen (Screen ID)

movie (Movie ID)

shows (Show ID)

booking (Booking ID)

ticket (Ticket ID)

user (User ID)

Now let's check for partial dependencies in each table:

theatre (Theatre ID, Name of Theatre, Area)

There are no partial dependencies in this table.

screen (Screen ID, No of Seats Gold, No of Seats Silver, Theatre ID)

No of Seats Gold and No of Seats Silver depend only on the Screen ID, and not on the entire primary key (Screen ID, Theatre ID). Therefore, we need to split this table into two tables:

screen (Screen ID, Theatre ID)

This table will have a composite primary key consisting of Screen ID and Theatre ID.

seat_info (Screen ID, No of Seats Gold, No of Seats Silver)

This table will also have a composite primary key consisting of Screen ID and Theatre ID.

The new schema in 2NF is:

theatre (Theatre ID, Name of Theatre, Area)

screen (Screen ID, Theatre ID)

seat_info (Screen ID, No of Seats Gold, No of Seats Silver)

movie (Movie ID, Name, Language, Genre, Target Audience)

shows (Show ID, Show Time, Show Date, Seats Remaining Gold, Seats Remaining Silver, Chat Cost Gold, Cas Cost Silver, Screen ID, Movie_ID)

booking (Booking ID, No of Tickets, Total Cost, KO Card Number, Name on card, User_ID, Show ID)

ticket (Ticket ID, Class, Price)

user (User ID, First Name, Last Name, Booking ID, Email ID, Age, Phone Number)

shows (Show ID, Show Time, Show Date, Seats Remaining Gold, Seats Remaining Silver, Chat Cost Gold, Cas Cost Silver, Screen ID, Movie_ID)

There are no partial dependencies in this table.

booking (Booking ID, No of Tickets, Total Cost, KO Card Number, Name on card, User_ID, Show ID)

No of Tickets, Total Cost, KO Card Number, and Name on card depend only on the Booking ID, and not on the entire primary key (Booking ID, User_ID, Show ID). Therefore, we need to split this table into two tables:

booking (Booking ID, User_ID, Show ID)

This table will have a composite primary key consisting of Booking ID, User_ID, and Show ID.

booking_details (Booking ID, No of Tickets, Total Cost, KO Card Number, Name on card)

This table will have Booking ID as its primary key.

The final schema in 2NF is:

theatre (Theatre ID, Name of Theatre, Area)
screen (Screen ID, Theatre ID)
seat_info (Screen ID, No of Seats Gold, No of Seats Silver)
movie (Movie ID, Name, Language, Genre, Target Audience)
shows (Show ID, Show Time, Show Date, Seats Remaining Gold, Seats Remaining Silver, Chat Cost Gold, Cas Cost Silver, Screen ID, Movie_ID)
booking (Booking ID, User_ID, Show ID)
booking_details (Booking ID, No of Tickets, Total Cost, KO Card Number, Name on card)
ticket (Ticket ID, Class, Price)
user (User ID, First Name, Last Name, Email ID, Age, Phone Number)

3. To convert the schema into third normal form (3NF), we need to ensure that there are no transitive dependencies in the tables. In other words, every non-key attribute should depend only on the primary key, and not on any other non-key attribute.

The primary key for each table is:

theatre (Theatre ID)

screen (Screen ID, Theatre ID)

seat_info (Screen ID)

movie (Movie ID)

shows (Show ID)

booking (Booking ID)

booking_details (Booking ID)

ticket (Ticket ID)

user (User ID)

Now let's check for transitive dependencies in each table:

theatre (Theatre ID, Name of Theatre, Area)

There are no transitive dependencies in this table.

screen (Screen ID, Theatre ID)

There are no transitive dependencies in this table.

seat_info (Screen ID, No of Seats Gold, No of Seats Silver)

No of Seats Gold and No of Seats Silver depend only on the Screen ID, which is not the primary key. Therefore, we need to split this table into two tables:

screen (Screen ID, Theatre ID)

This table will have a composite primary key consisting of Screen ID and Theatre ID.

seat_info (Screen ID, Class, No of Seats)

This table will have a composite primary key consisting of Screen ID and Class.

The new schema in 3NF is:

theatre (Theatre ID, Name of Theatre, Area)

screen (Screen ID, Theatre ID)

seat_info (Screen ID, Class, No of Seats)

movie (Movie ID, Name, Language, Genre, Target Audience)

shows (Show ID, Show Time, Show Date, Seats Remaining Gold, Seats Remaining Silver, Chat Cost Gold, Cas Cost Silver, Screen ID, Movie_ID)

booking (Booking ID, User_ID, Show ID)

booking_details (Booking ID, No of Tickets, Total Cost, KO Card Number, Name on card)

ticket (Ticket ID, Class, Price)

user (User ID, First Name, Last Name, Email ID, Age, Phone Number)

movie (Movie ID, Name, Language, Genre, Target Audience)

There are no transitive dependencies in this table.

shows (Show ID, Show Time, Show Date, Seats Remaining Gold, Seats Remaining Silver, Chat Cost Gold, Cas Cost Silver, Screen ID, Movie_ID)

There are no transitive dependencies in this table.

booking (Booking ID, User_ID, Show ID)

There are no transitive dependencies in this table.

booking_details (Booking ID, No of Tickets, Total Cost, KO Card Number, Name on card)

There are no transitive dependencies in this table.

ticket (Ticket ID, Class, Price)

There are no transitive dependencies in this table.

user (User ID, First Name, Last Name, Email ID, Age, Phone Number)

There are no transitive dependencies in this table.

The final schema in 3NF is:

theatre (Theatre ID, Name of Theatre, Area)

screen (Screen ID, Theatre ID)

seat_info (Screen ID, Class, No of Seats)

movie (Movie ID, Name, Language, Genre, Target Audience)

shows (Show ID, Show Time, Show Date, Seats Remaining Gold, Seats Remaining Silver, Chat Cost Gold, Cas Cost Silver, Screen ID, Movie_ID)

booking (Booking ID, User_ID, Show ID)

booking_details (Booking ID, No of Tickets, Total Cost, KO Card Number, Name on card)

ticket (Ticket ID, Class, Price)

user (User ID, First Name, Last Name, Email ID, Age, Phone Number)

4. To convert the schema into Boyce-Codd normal form (BCNF), we need to ensure that every determinant is a candidate key. In other words, for every functional dependency $X \rightarrow Y$, X must be a superkey.

The primary key for each table is:

theatre (Theatre ID)

screen (Screen ID, Theatre ID)

seat_info (Screen ID, Class)

movie (Movie ID)

shows (Show ID)

booking (Booking ID)

booking_details (Booking ID)

ticket (Ticket ID)

user (User ID)

Now let's check for functional dependencies in each table:

theatre (Theatre ID, Name of Theatre, Area)

There are no non-trivial functional dependencies in this table.

screen (Screen ID, Theatre ID)

There are no non-trivial functional dependencies in this table.

seat_info (Screen ID, Class, No of Seats)

The determinant for No of Seats is (Screen ID, Class). Therefore, we need to split this table into two tables:

screen (Screen ID, Theatre ID)

This table will have a composite primary key consisting of Screen ID and Theatre ID.

seat_info (Screen ID, Class, No of Seats)

This table will have a composite primary key consisting of Screen ID and Class.

The new schema in BCNF is:

theatre (Theatre ID, Name of Theatre, Area)

screen (Screen ID, Theatre ID)

seat_info (Screen ID, Class, No of Seats)

movie (Movie ID, Name, Language, Genre, Target Audience)

shows (Show ID, Show Time, Show Date, Seats Remaining Gold, Seats Remaining Silver, Chat Cost Gold, Cas Cost Silver, Screen ID, Movie_ID)

booking (Booking ID, User_ID, Show ID)

booking_details (Booking ID, No of Tickets, Total Cost, KO Card Number, Name on card)

ticket (Ticket ID, Class, Price)

user (User ID, First Name, Last Name, Email ID, Age, Phone Number)

movie (Movie ID, Name, Language, Genre, Target Audience)

There are no non-trivial functional dependencies in this table.

shows (Show ID, Show Time, Show Date, Seats Remaining Gold, Seats Remaining Silver, Chat Cost Gold, Cas Cost Silver, Screen ID, Movie_ID)

There are no non-trivial functional dependencies in this table.

booking (Booking ID, User_ID, Show ID)

There are no non-trivial functional dependencies in this table.

booking_details (Booking ID, No of Tickets, Total Cost, KO Card Number, Name on card)

There are no non-trivial functional dependencies in this table.

ticket (Ticket ID, Class, Price)

There are no non-trivial functional dependencies in this table.

user (User ID, First Name, Last Name, Email ID, Age, Phone Number)

There are no non-trivial functional dependencies in this table.

Therefore, the final schema in BCNF is:

theatre (Theatre ID, Name of Theatre, Area)

screen (Screen ID, Theatre ID)

seat_info (Screen ID, Class, No of Seats)

movie (Movie ID, Name, Language, Genre, Target Audience)

shows (Show ID, Show Time, Show Date, Seats Remaining Gold, Seats Remaining Silver, Chat Cost Gold, Cas Cost Silver, Screen ID, Movie_ID)

booking (Booking ID, User_ID, Show ID)

booking_details (Booking ID, No of Tickets, Total Cost, KO Card Number, Name on card)

ticket (Ticket ID, Class, Price)

user (User ID, First Name, Last Name, Email ID, Age, Phone Number)

The schema is already in the highest normal form (BCNF). All the tables have been normalized such that every determinant is a candidate key, and there are no non-trivial functional dependencies. Therefore, the schema cannot be further decomposed while preserving the integrity and functionality of the original data.

So Final Schema is:

User (User_ID, First_Name, Last_Name, Email_ID, Age)

User_Phone_Number (Phone_Number, User_ID)

Theatre (Theatre_ID, Name_of_Theatre, No_of_Screens, Area)

Screen_Details (Screen_ID, No_of_Seats_Gold, No_of_Seats_Silver, Theatre_ID)

Movie (Movie_ID, Name)

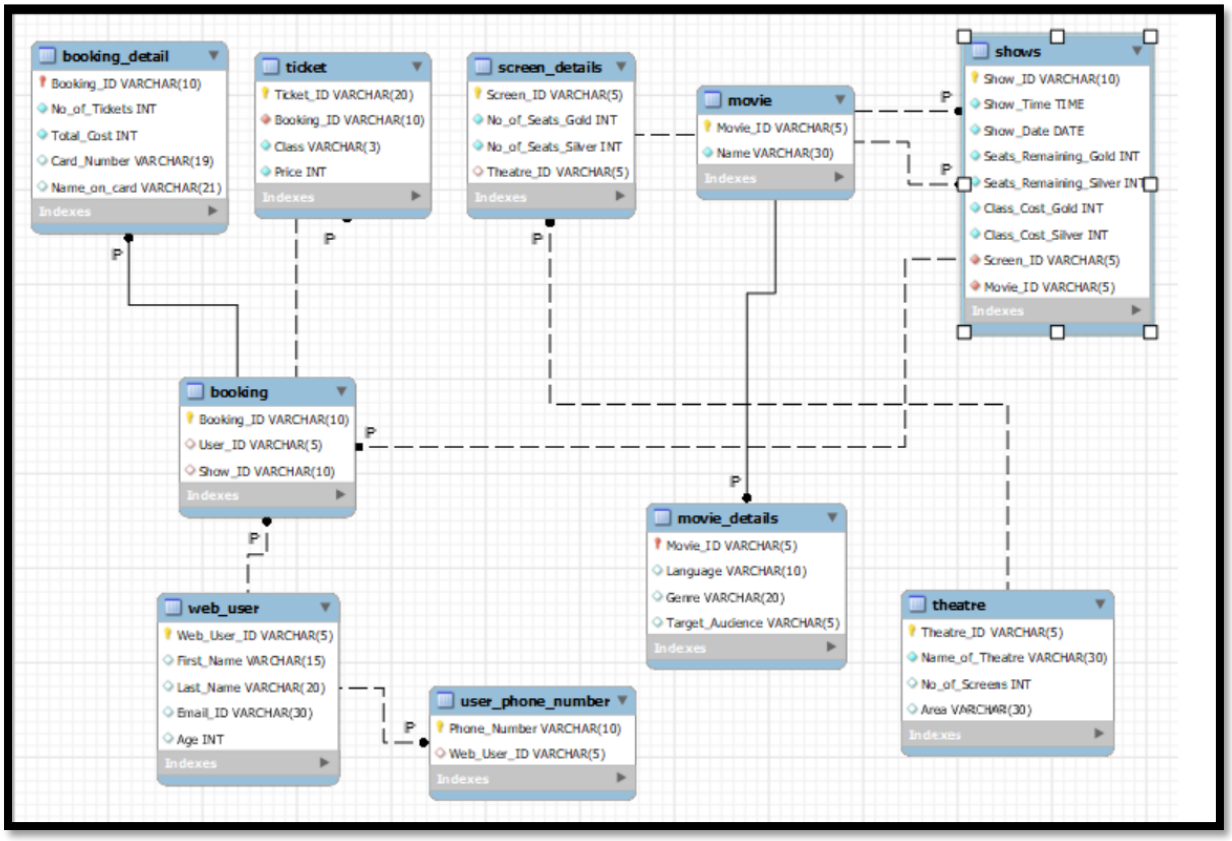
Movie_Details (Movie_ID, Language, Genre, Target_Audience)

Show (Show_ID, Show_Time, Show_Date, Seats_Remaining_Gold, Seats_Remaining_Silver, Class_Cost_Gold, Class_Cost_Silver, Screen_ID, Movie_ID)

Booking (Booking_ID, No_of_Tickets, Total_Cost, Card_Number, Name_on_card, User_ID, Show_ID)

Ticket (Ticket_ID, Booking_ID, Class, Price)

Final Schema Diagram :



Function Dependencies :

Functional Dependencies (FDs) are constraints that define how the values of one or more attributes in a relation determine the values of one or more other attributes. Based on the given database schema, the following FDs can be identified:

1. Web_User:

- Web_User_ID → First_Name, Last_Name, Email_ID, Age

2. User_Phone_Number:

- Phone_Number → Web_User_ID

3. Theatre:

- Theatre_ID → Name_of_Theatre, No_of_Screens, Area

4. Screen_Details:

- Screen_ID → No_of_Seats_Gold, No_of_Seats_Silver, Theatre_ID

5. Movie:

- Movie_ID → Name

6. Movie_Details:

- Movie_ID → Language, Genre, Target_Audience

7. Shows:

- Show_ID → Show_Time, Show_Date

SQL Query

Create Table of Final Normalize Schema :

```
CREATE TABLE Web_User (
   Web_User_ID VARCHAR(5) PRIMARY KEY,
   First_Name VARCHAR(15),
   Last_Name VARCHAR(20),
   Email_ID VARCHAR(30),
   Age INT
);

CREATE TABLE User_Phone_Number (
   Phone_Number VARCHAR(10) PRIMARY KEY,
   Web_User_ID VARCHAR(5),
   FOREIGN KEY (Web_User_ID) REFERENCES Web_User(Web_User_ID) ON DELETE
CASCADE ON UPDATE CASCADE
);

CREATE TABLE Theatre (
   Theatre_ID VARCHAR(5) PRIMARY KEY,
   Name_of_Theatre VARCHAR(30) NOT NULL,
   No_of_Screens INT,
   Area VARCHAR(30)
);

CREATE TABLE Screen_Details (
   Screen_ID VARCHAR(5) PRIMARY KEY,
   No_of_Seats_Gold INT NOT NULL,
   No_of_Seats_Silver INT NOT NULL,
```

```
    Theatre_ID VARCHAR(5),
    FOREIGN KEY (Theatre_ID) REFERENCES Theatre(Theatre_ID) ON DELETE CASCADE ON
UPDATE CASCADE
);

CREATE TABLE Movie (
    Movie_ID VARCHAR(5) PRIMARY KEY,
    Name VARCHAR(30) NOT NULL
);

CREATE TABLE Movie_Details (
    Movie_ID VARCHAR(5) PRIMARY KEY,
    Language VARCHAR(10),
    Genre VARCHAR(20),
    Target_Audience VARCHAR(5),
    FOREIGN KEY (Movie_ID) REFERENCES Movie(Movie_ID) ON DELETE CASCADE ON
UPDATE CASCADE
);

CREATE TABLE Shows (
    Show_ID VARCHAR(10) PRIMARY KEY,
    Show_Time TIME NOT NULL,
    Show_Date DATE NOT NULL,
    Seats_Remaining_Gold INT NOT NULL CHECK (Seats_Remaining_Gold >= 0),
    Seats_Remaining_Silver INT NOT NULL CHECK (Seats_Remaining_Silver >= 0),
    Class_Cost_Gold INT NOT NULL,
    Class_Cost_Silver INT NOT NULL,
    Screen_ID VARCHAR(5) NOT NULL,
    Movie_ID VARCHAR(5) NOT NULL,
    FOREIGN KEY (Screen_ID) REFERENCES Screen_Details(Screen_ID) ,
    FOREIGN KEY (Movie_ID) REFERENCES Movie(Movie_ID)
);
CREATE TABLE Booking (
```

```
    Booking_ID VARCHAR(10) PRIMARY KEY,
    User_ID VARCHAR(5),
    Show_ID VARCHAR(10),
    FOREIGN KEY (User_ID) REFERENCES Web_User(Web_User_ID) ,
    FOREIGN KEY (Show_ID) REFERENCES Shows(Show_ID)
);
CREATE TABLE Booking_Detail (
    Booking_ID VARCHAR(10) PRIMARY KEY,
    No_of_Tickets INT NOT NULL,
    Total_Cost INT NOT NULL,
    Card_Number VARCHAR(19),
    Name_on_card VARCHAR(21),
        FOREIGN KEY (Booking_ID) REFERENCES Booking(Booking_ID)
);


CREATE TABLE Ticket (
    Ticket_ID VARCHAR(20) PRIMARY KEY,
    Booking_ID VARCHAR(10) NOT NULL,
    Class VARCHAR(3) NOT NULL,
    Price INT NOT NULL,
    FOREIGN KEY (Booking_ID) REFERENCES Booking(Booking_ID)
);
```

Insert Value in the table (Populate Data):

```
-- Web_User table
INSERT INTO Web_User (Web_User_ID, First_Name, Last_Name, Email_ID, Age)
VALUES
   ('WU001', 'Rashmi', 'Panchal', 'rashmi.panchal@example.com', 25),
   ('WU002', 'Jugal', 'Chattriwala', 'jugal.chattriwala@example.com', 30),
   ('WU003', 'Jainil', Doshi', 'jainil.doshi@example.com', 27),
   ('WU004', 'Amit', 'Patel', 'amit.patel@example.com', 28),
   ('WU005', 'Sara', 'Shah', 'sara.shah@example.com', 22);

-- User_Phone_Number table
INSERT INTO User_Phone_Number (Phone_Number, Web_User_ID)
VALUES
   ('9949303201', 'WU001'),
   ('9898123434', 'WU002'),
   ('1290238934', 'WU003'),
   ('9012233445', 'WU004'),
   ('8797076757', 'WU005');

-- Theatre table
INSERT INTO Theatre (Theatre_ID, Name_of_Theatre, No_of_Screens, Area)
VALUES
   ('T001', 'PVR Phoenix', 6, 'Ahemdabad'),
   ('T002', 'Conplex', 8, 'Vadodra'),
   ('T003', 'Cinepolis ', 4, 'Bhopal'),
   ('T004', 'Rajhans', 5, 'Sciencecity'),
   ('T005', 'INOX Forum', 6, 'Rajkot');
```

```
-- Screen_Details table
INSERT INTO Screen_Details (Screen_ID, No_of_Seats_Gold, No_of_Seats_Silver, Theatre_ID)
VALUES
   ('S001', 100, 200, 'T001'),
   ('S002', 150, 250, 'T002'),
   ('S003', 120, 180, 'T003'),
   ('S004', 80, 150, 'T004'),
   ('S005', 90, 170, 'T005');

-- Movie table
INSERT INTO Movie (Movie_ID, Name)
VALUES
   ('M001', 'Pathan'),
   ('M002', 'KGF CAPTER 2'),
   ('M003', 'Bahubali: The Conclusion'),
   ('M004', 'DDLJ'),
   ('M005', 'Mummy Returns');

-- Movie_Details table
INSERT INTO Movie_Details (Movie_ID, Language, Genre, Target_Audience)
VALUES
   ('M001', 'English', 'Action', 'PG-13'),
   ('M002', 'English', 'Drama', 'R'),
   ('M003', 'Hindi', 'Epic', 'U/A'),
   ('M004', 'Hindi', 'Romance', 'U'),
   ('M005', 'English', 'Animation', 'U');
--Show values
INSERT INTO Show (Show_ID, Show_Time, Show_Date, Seats_Remaining_Gold,
Seats_Remaining_Silver, Class_Cost_Gold, Class_Cost_Silver, Screen_ID, Movie_ID)
VALUES ('S001', '13:00:00', '2023-05-01', 20, 50, 300, 200, 'SC001', 'M001');
```

```
INSERT INTO Show (Show_ID, Show_Time, Show_Date, Seats_Remaining_Gold,
Seats_Remaining_Silver, Class_Cost_Gold, Class_Cost_Silver, Screen_ID, Movie_ID)
VALUES ('S002', '17:30:00', '2023-05-02', 25, 60, 300, 200, 'SC002', 'M002');

INSERT INTO Show (Show_ID, Show_Time, Show_Date, Seats_Remaining_Gold,
Seats_Remaining_Silver, Class_Cost_Gold, Class_Cost_Silver, Screen_ID, Movie_ID)
VALUES ('S003', '10:45:00', '2023-05-03', 15, 40, 250, 150, 'SC003', 'M003');

INSERT INTO Show (Show_ID, Show_Time, Show_Date, Seats_Remaining_Gold,
Seats_Remaining_Silver, Class_Cost_Gold, Class_Cost_Silver, Screen_ID, Movie_ID)
VALUES ('S004', '20:15:00', '2023-05-04', 10, 30, 350, 250, 'SC004', 'M004');

INSERT INTO Show (Show_ID, Show_Time, Show_Date, Seats_Remaining_Gold,
Seats_Remaining_Silver, Class_Cost_Gold, Class_Cost_Silver, Screen_ID, Movie_ID)
VALUES ('S005', '12:30:00', '2023-05-05', 18, 45, 300, 200, 'SC005', 'M005');

--Booking Table
INSERT INTO Booking (Booking_ID, Web_User_ID, Show_ID)
VALUES
('B001', 'WU001', 'S001'),
('B002', 'WU002', 'S002'),
('B003', 'WU003', 'S003'),
('B004', 'WU001', 'S004'),
('B005', 'WU002', 'S005');

--Booking Detail Table

INSERT INTO Booking_Detail (Booking_ID, No_of_Tickets, Total_Cost, Card_Number, Name_on_card)
VALUES
('B001', 2, 600, '1234567890123456', 'Rashmi Panchal'),
('B002', 1, 200, '2345678901234567', 'Jugal Chhatrivala'),
('B003', 3, 900, '3456789012345678', 'Jainil Doshi'),
```

```
('B004', 4, 1200, '4567890123456789', 'Amit Patel'),
('B005', 2, 600, '5678901234567890', 'Sara Shah');


--Ticket Table
INSERT INTO Ticket (Ticket_ID, Booking_ID, Class, Price)
VALUES
('T001', 'B001', 'G', 300),
('T002', 'B001', 'G', 300),
('T003', 'B002', 'S', 200),
('T004', 'B003', 'G', 300),
('T005', 'B003', 'G', 300),
('T006', 'B003', 'G', 300),
('T007', 'B004', 'G', 300),
('T008', 'B004', 'G', 300),
('T009', 'B004', 'G', 300),
('T010', 'B004', 'S', 300),
('T011', 'B005', 'G', 300),
('T012', 'B005', 'S', 300);
```

Output Table

- **Theatre:**

Theatre_ID	Name_of_Theatre	No_of_Screens	Area
T001	PVR Phoenix	6	Ahemdabad
T002	Conplex	8	Vadodra
T003	Cinepolis	4	Bhopal
T004	Rajhans	5	Sciencecity
T005	INOX Forum	6	Rajkot

- **Screen:**

Screen_ID	No_of_Seats_Gold	No_of_Seats_Silver	Theatre_ID
S001	100	200	T001
S002	150	250	T002
S003	120	180	T003
S004	80	150	T004
S005	90	170	T005

- **Movie:**

Movie_ID	Name
M001	Pathan
M002	KGF CAPTER 2
M003	Bahubali: The Conclusion
M004	DDLJ
M005	Mummy Returns

- **Movie Detail:**

Movie_ID	Language	Genre	Target_Audience
M001	English	Action	PG-13
M002	English	Drama	R
M003	Hindi	Epic	U/A
M004	Hindi	Romance	U
M005	English	Animation	U

- **Show:**

Show_ID	Show_Time	Show_Date	Seats_Remaining_Gold	Seats_Remaining_Silver	Class_Cost_Gold	Class_Cost_Silver	Screen_ID	Movie_ID
S001	13:00:00	2023-05-01	20	50	300	200	SC001	M001
S002	17:30:00	2023-05-02	25	60	300	200	SC002	M002
S003	10:45:00	2023-05-03	15	40	250	150	SC003	M003
S004	20:15:00	2023-05-04	10	30	350	250	SC004	M004
S005	12:30:00	2023-05-05	18	45	300	200	SC005	M005

- **User:**

Web_User_ID	First_Name	Last_Name	Email_ID	Age
WU001	Rashmi	Panchal	rashmi.panchal@example.com	25
WU002	Jugal	Chattriwala	jugal.chattriwala@example.com	30
WU003	Jainil	Doshi	jainil.doshi@example.com	27
WU004	Amit	Patel	amit.patel@example.com	28
WU005	Sara	Shah	sara.shah@example.com	22

- **Booking:**

Result Grid Filter Rows:

Booking_ID	Web_User_ID	Show_ID
B001	WU001	S001
B002	WU002	S002
B003	WU003	S003
B004	WU001	S004
B005	WU002	S005
NULL	NULL	NULL

- **Booking Detail:**

Booking_ID	No_of_Tickets	Total_Cost	Card_Number	Name_on_card
B001	2	600	1234567890123456	Rashmi Panchal
B002	1	200	2345678901234567	Jugal Chhatrivala
B003	3	900	3456789012345678	Jainil Doshi
B004	4	1200	4567890123456789	Amit Patel
B005	2	600	5678901234567890	Sara Shah
NULL	NULL	NULL	NULL	NULL

- **Ticket:**

	Ticket_ID	Booking_ID	Class	Price
▶	T001	B001	G	300
	T002	B001	G	300
	T003	B002	S	200
	T004	B003	G	300
	T005	B005	S	300

How this Report Help

This SQL code creates several tables that could be used in a movie theater booking system. Here are some benefits of this database design:

1. Primary keys: Each table has a primary key which uniquely identifies each record. This helps in maintaining data integrity and ensures that duplicate data is not added to the tables.

2. Foreign keys: The use of foreign keys ensures that data is consistent across multiple tables. For example, the Web_User_ID field in the User_Phone_Number table references the Web_User table's primary key, which ensures that only valid user IDs are entered in the User_Phone_Number table.

3. Referential integrity: The use of foreign keys also helps to maintain referential integrity. If a record is deleted from the Web_User table, all related records in the User_Phone_Number table are automatically deleted due to the ON DELETE CASCADE option.

4. Data validation: The use of CHECK constraints ensures that data entered into the tables meets specific criteria. For example, the Seats_Remaining_Gold and Seats_Remaining_Silver fields in the Shows table must have a value greater than or equal to zero.

5. Security: By using primary keys, foreign keys, and referential integrity, the database is more secure as it reduces the chances of data being tampered with or lost.

Overall, this database design ensures data consistency, accuracy, and security, making it suitable for use in a movie Ticket booking system.

Conclusion

In conclusion, the Movie Ticket Booking DBMS project was a challenging but rewarding experience for our team. We worked together to design and implement a database system that can efficiently manage the different entities and relationships in the movie booking process, including theatres, screens, movies, shows, bookings, tickets, and users.

Through this project, we learned how to apply various database concepts such as the relational model, normalization, and SQL queries to create a functional and efficient system. We also developed skills in project management, collaboration, and communication as we worked together to meet project milestones and deliverables.

As a team, we are proud of the final result of our project and the effort we put into it. We believe that the database system we developed can be a valuable tool for movie theaters and users alike, making the movie booking process more efficient and user-friendly.

Overall, this project was a valuable learning experience for us as we developed technical skills and gained experience working as a team. We are grateful for the opportunity to work on this project together and look forward to applying our skills in future projects.

The End