

MOVIE TICKET BOOKING SYSTEM WITH SOCIAL DISTANCING

BY

Mounica Bachu (Id:0152625202)
Shiva Abhishek Varma Penmetsa (Id: 015212171)
Swarupa Gowri Adiseshachalam (Id: 014568671)

Movie Ticket Booking System with Social Distancing
Term project report

Table of Contents

PROJECT DESCRIPTION	N
DATABASE INITIAL STUDY	N
É ÁBÁI Ó B	N
AĒ AĒJÍ Ó T HBAÍ ÓBÉ BĒÍ	N
CÚRNÓRKOHNÍUÓNR NÓRÚÉ	N
É RREĆCÚRNÓRKOHNÍUÓNR NÓRÚÉ	N
GHE AĒBÉ AEBÓ DĒ E	N
BLÓMÓ UUÚNR ÁE	N
GTRTRUNÓ UUÚNR ÁE	N
ÁE EÍ HADÍ	N
I ÁE GB	N
ÅE I ÉAEHÓ	N
DATABASE DESIGN – CONCEPT DESIGN	O
ÅÍ I Ó BI HÍ ÉBI	O
AIIÍ É G ÓÉI	O
BÉI ÓBI	L·E
HBÉAI Ó E I DÓ	L·E
DATABASE DESIGN – LOGICAL DESIGN	L·M
ÉE Ç ÓAÉ HBÉAI Ó E I DÓ	L·M
DATABASE DESIGN – PHYSICAL DESIGN	L·N
GDUJ ÓAÉ I Ó E HAÇ B BI Ó E AÍ BI	L·N
I BÁI HÓU AĒ AÄÅBI I ÅE EÍ HÉ	L·N
DBMS SELECTION	L·N
AÄÅE I AĒ AĒJÍ Ó	L·N
AÄÅE I I BÉBÁI Ó E	L·N
DAHÆI AHB AĒ AÉ I ÉAÍ I AHB HBÁI ÓBÉ BĒÍ	L·O
IMPLEMENTATION & LOADING	M
AÄÅE I Ó I AEEAÍ Ó E	M
I ÁDBÉ A ÁHBÁI Ó E	M
AÉAÍ A ÉE AEEC Ó T ÅE EÍ BHÍ Ó E	NN
TESTING AND EVALUATION	NN
I ÁHBÉ I DÉÍ	NN
I AÉ Ó Ó BÉ I DÉÍ	NN
ÅE E EÍ DÉ I EÅJÍ I AÍ I	NN
I GAEÍ I BH	NN
I GAEÍ BAÍ E GHÁB	NO
ÅEE ÅEE AØ BHÉ AÍ BÉ HÉ I	NL
ÁHBÁI BBÉ GEE I JBB	NL
ÁHBÁI BÁI I Ó E E BH	NN
ÁHBÁI BÉ E Ó DÉ	NN
ÁHBÁI BÉ E Ó DÉ	NN
I DBAÍ HBÉ HBÉ E Ó AÉ	NN
I ÓI E ÅE E EBAE I BAÍ	NN
E AEBÉ GAIJÉ BÉ I	NN

Movie Ticket Booking System with Social Distancing Term project report

I BAÌ ËÅÉ É ÊÏŒ Ç	ÑÑ
QUERY DESCRIPTION	ÑM
BÍ BĒÌ	ÑM
Ãœ I KRÖÖNËI ÖRÚ	ÑL
Ì HBÖÇBHÌ	ÑM
Ãœ ÅÈ É ËÉI DÉ Í ËÅJI ÌAÌ Í I	ÑL
Ãœ Í TÑÉ ÜLNF	ÑL
Ãœ Í TÑÉ ÜNKÜËTTÖN	ÑL
Ì Í ÉHBÆ GHE ÁBAE HBI	ÑN
Ãœ ÅÉÉ ÁÉÉ AËB BHÉ AÌ BËHÉ Í I FG	ÑM
Ãœ NÄNKÚÑBR TÖVNM	ÑN
Ãœ NÄNKÚÑÁÜLURR NF	ÑN
ÃœG NÄNKÚÑÉ RÜÖN	ÑN
ÜG NÄNKÚÑÉ RÜÖN ÖRÚ	ÑV
ÜÃœ Ë AËBËÇAIJÉ BËÌ	ÑN
ÜÃœ Í ÕNUËÅRRPÖNNËÜNKÚ	ÑN
ÜÃœ Ì DBAÌ HBËHBË ËÍ AË	ÑN
ÃœG I BAÌ ËÅÉ É ÊÏŒ Ç	ÑL'
SUMMARY & CONCLUSION	ÑÑ
FUTURE WORK	ÑÑ

Project Description

A movie ticket booking system provides its customers with the ability to purchase theatre seats online. E-ticketing systems allow the customers to browse through movies currently being played in the theatres and to book seats, anywhere anytime.

With the ongoing situation of pandemic, to ensure the reduction of the spread of the covid virus, people have been asked to maintain social distance in all public places. Since movie theatres also come under the category of public exposure, social distancing has to be followed both in the theatres/cinemas and inside the screens where each movie will be casted.

So here in this project module, we ensure that the movie tickets are booked with the protocols of social distancing. By doing so, the customers can watch the movie peacefully, without any commotion of pandemic. It also helps the movie theatre management to run its business and provide entertainment for customers.

Database initial study

Objective

The database of the movie ticket booking system with social distancing is aimed to display overall information of the new release of movies and their corresponding screening schedule to the users, based on which the customers can reserve a ticket to watch movies. Admin maintains the movie information of the theatres associated, by updating the details of the movie screenings which will in turn update the database and are up for display to the users. Automating the booking process can decrease the complexity of managing the key data in the database. We can elevate the customer's experience by considering the seat preference by the first come first serve basis based on booking a movie ticket in a few easy steps. This database can be deployed as a backend database design for a website which handles movie booking for a range of theatres across many cities and their screenings.

Analysis & Requirements

The main purpose of the theatre/multiplex organization is to provide screening of new movies without any chaos amidst the pandemic situation. In order to attain this organization is planning to implement the following operations.

- Provide options for customers to book tickets for movies without any physical contact.
- Ensure social distancing is followed not only in the theatre/multiplex but also within the movie screens.
- Sanitize the screens after each movie screening.
- Check the fitness of the employees on a daily basis.

The overall management of the above-mentioned operations will be managed by the theatre/multiplex manager. Each department in the theatre/multiplex will have supervisors who will be reporting to the manager. The supervisor will be responsible for the work in their corresponding department. The employees in each department will be reporting to their supervisor. They will be following a work schedule organized by the supervisor every day.

Our ticket booking service should meet the following requirements:

Functional Requirements:

- Once the user selects a movie, the service should display the cinemas running that movie and its available shows.
- The user should be able to choose a show at a particular cinema and book their tickets.
- The service should be able to show the user the seating arrangement of the cinema hall. The user should be able to select multiple seats up to 10 seats according to their preference.
- The user should be able to distinguish available seats from the booked ones.
- Users should be able to put a hold on the seats for five minutes before they make a payment to finalize the booking.
- The user should be able to wait if there is a chance that the seats might become available – e.g., when held by other users expire.

Non-Functional Requirements:

- The system would need to be highly concurrent. There will be multiple booking requests for the same seat at any particular point in time. The service should handle this gracefully and fairly.
- The core thing of the service is ticket booking which means financial transactions. This means that the system should be secure and the database ACID compliant.

Problem Definition

Since the current pandemic situation is new for everyone, the existing system does not support the new guidelines to handle the pandemic. A study of the existing system is as follows.

Existing system:

In terms of ticket booking, the current system facilitates the customers to book movie tickets online without any social distancing norms. It also guides the customers to choose movies by providing the rating of the movie. The administrator will be able to add the movie schedule as per the plan and maintain the system. The following diagram depicts the process of the existing system.

Movie Ticket Booking System with Social Distancing Term project report

Customers ± Select movie ± Choose seats of their choice ± Pay via payment gateway ± Book tickets

System ± Releases the seats chosen by the customer if the payment fails or not paid by the customer ± updates the available seats after the booking/cancellation

Proposed system:

A module that enhances the existing system by implementing the social distancing norms is developed here. Every alternate row will be mandatorily blocked from booking so that the pandemic guidelines are followed within the screens in each theatre. Once the customer chooses seats from the remaining available seats, two seats from either side of their booking will be blocked automatically. E.g.: If a customer books two seats in the same row, then 6 seats will be blocked for their booking. Two seats are booked by the customer, two seats on the left and 2 seats on the right are automatically blocked to maintain the distance with the next booking. The customer will be charged only for the number of seats he/she has booked. This assures that the distance is maintained between different customers. The following diagram depicts the process of the proposed system.

Customers ± Select movie ± Choose seats from the given choice ± Block additional four seats on either side of the given booking ± Pay via payment gateway ± Book tickets

System ± Releases both the seats chosen by the customer and the four seats that are blocked either side of the customer chosen seats if the payment fails or not paid by the customer ± updates the available seats after the booking/cancellation.

Constraints

- There are few constraints to be considered in the working of the proposed system.
- System will not handle partial ticket orders. Either the user gets all the tickets they want, or they get nothing.
- System should be consistent and show accurate results.

Movie Ticket Booking System with Social Distancing Term project report

- To stop system abuse, we can restrict users not to book more than ten seats at a time.
- We can assume that traffic would spike on popular/much-awaited movie releases, and the seats fill up pretty fast. The system should be scalable, highly available to cope up with the surge in traffic.
- System security must be considered since the payment transactions will be taking place for every booking.

Scope

- This system is developed keeping in view of the current multiplex working pattern. Schedules for many screens can be programmed in this application.
- Customers can buy tickets 24×7.
- Tickets availability is visible to only verified users.
- Movie Rating should be given by only users who booked tickets.

Boundaries

- Payment should be completed within 5 minutes after reaching the payment page and the seat gets blocked.
- Verified users should be able to book up to 10 tickets per screening.
- Booked tickets cannot be cancelled before 2 hours of the screening
- If the payment is failed or if the ticket is cancelled, the blocked ticket should be open for booking immediately.

Database Design – Concept Design

In this data model, an abstract database structure is created to represent the real-world objects. The entities and the relationships between them are identified and a conceptual ERD is developed in this stage. This ERD will provide a clear picture of the business and its functional areas. The business rules and the assumptions to be considered for the proposed movie ticket booking with a social distancing system are as follows.

Business rules

- A user can book multiple bookings.
- Each booking may have a maximum of 10 seats reserved.
- One seat can be occupied by one person (kids below 1 year does not require tickets).
- One movie can be screened in many theatres.
- One theatre has one or more screens.
- One theatre screen can screen one movie at a time.
- One booking will have one payment reference number.
- Theatre runs multiple movie shows in a day.
- Theatre has many employees.
- Each screen contains 60 seats.
- Block seats after first booking for each screen.
- Employee updates sanitizing checks after the end of screening of each show.

Assumptions

- One or more theatres can be present in a multiplex.
- Each theatre has 60 seats.
- Each seat can be sold only once per screening.
- Seats are numbered with row number and seat position within a row.
- Database has a track of each reservation into the system.
- A movie can have more than one screening at different times or can be screened simultaneously in a different auditorium.

Entities

From the mentioned business rules, the entities that were identified are as follows.

- Employee
- Theatre
- Screen
- Seat
- Movie_Show
- Show_Seat
- Movie
- Customer
- Payment
- Booking

Relationship

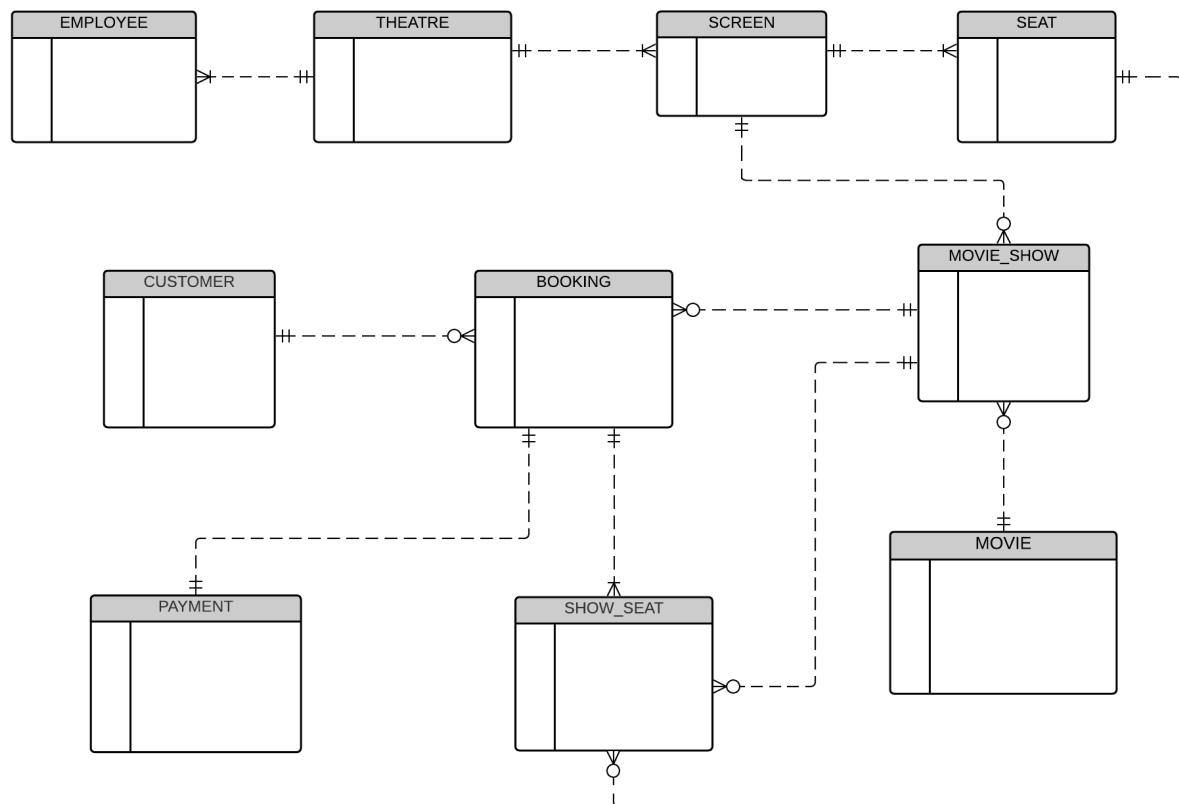
The basic relationship between the identified entities are as follows.

- An employee works for a theatre.
- A theatre has one or many screens.
- Each screen has one or more seats.
- Each screen presents a movie show.
- Only one movie is casted in a show at a time.
- An admin adds/deletes a movie to the system.
- A customer books a movie show.
- Customer makes a payment for each booking.
- Each booking allocates seats in the screen for a show.

A high level ERD is designed in this stage which represents a functional model of the identified entities and the relationships among them. The figure 3.1 illustrates the conceptual ERD of the movie ticket booking system with social distancing.

Movie Ticket Booking System with Social Distancing
Term project report

Figure: 3.1: Conceptual ER-Diagram



Database Design – Logical Design

Here, the devised entities and relationships in the conceptual design are converted to logical and in a more meaningful manner. The relationships are described further to depict a clear picture of the relationship between the entities. The attributes and constraints for each entity are defined. All the tables are normalized until 3NF.

Logical Relationships

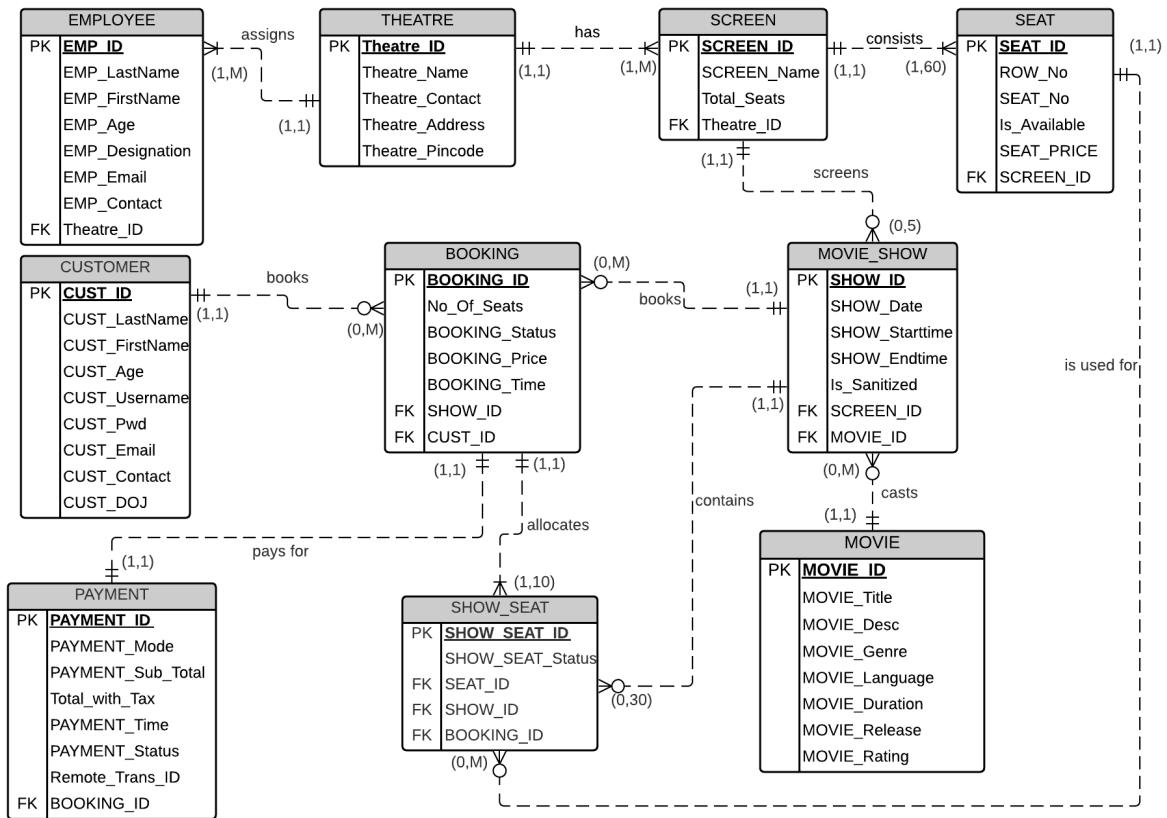
The specification of the relationship between the mentioned entities are as follows.

- A theatre has one to many relationships with employee
- Each employee is associated with one theatre
- A theatre has one to many relationships with screen
- Each screen belongs to one theatre
- A screen has zero to many relationships with movie show
- Movie show has one to one relationship with Screen
- Each movie can be screened in many movie shows.
- Movie show has one to one relationship with Movie
- Movie show has zero to many relationships with Booking
- Each Booking has one to one relationship by customer
- A customer has zero to many relationships with Booking
- A Booking has one to many relationships with show seat
- Each show seat has one to one relationship with Booking
- Each Booking has one to one relationship with payment

Figure 4.1 illustrates the logical ERD with attributes, relationships, cardinality and constraints between the entities. The normalization process for each entity is defined in the figure 4.2.

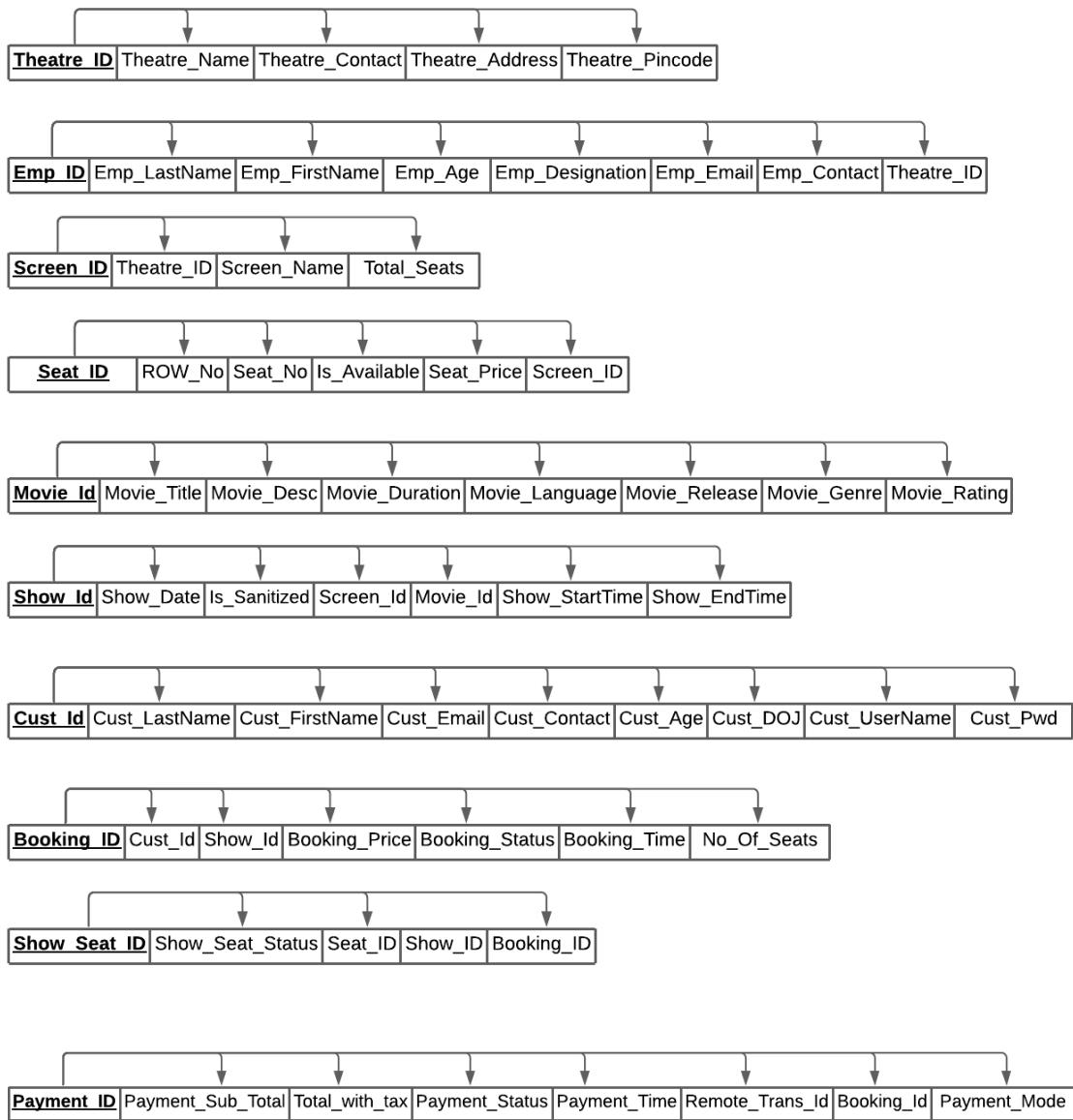
Movie Ticket Booking System with Social Distancing
Term project report

Figure: 4.1: Logical ER-Diagram



Movie Ticket Booking System with Social Distancing
Term project report

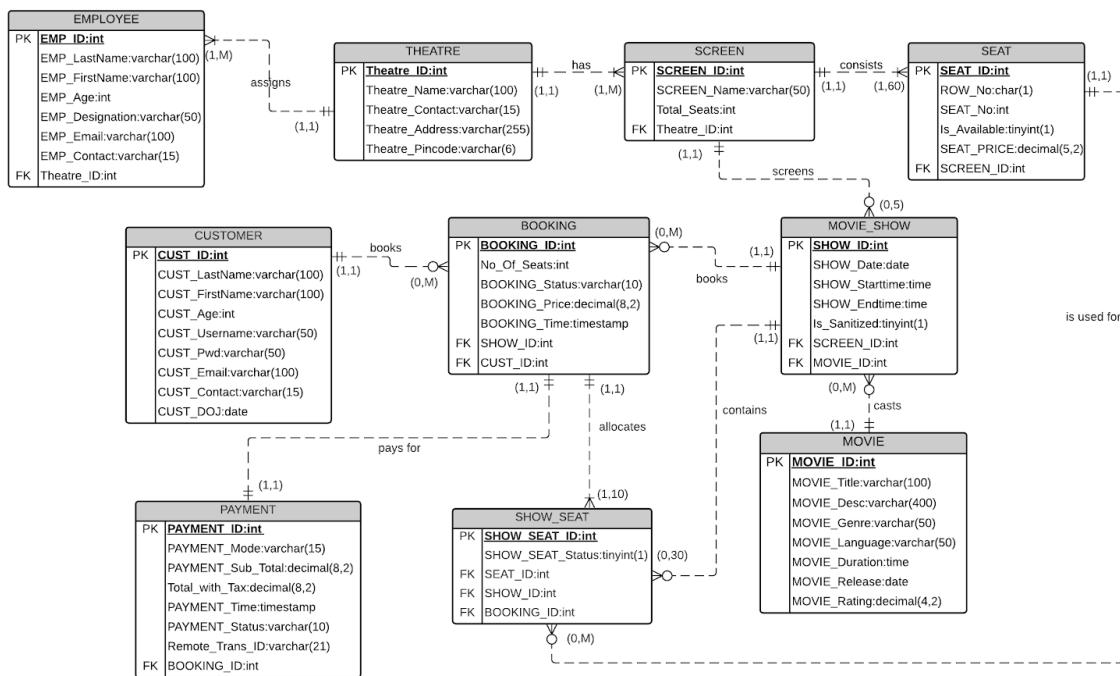
Figure: 4.2: Normalization process



Database Design – Physical Design

The following ERD illustrates the storage capacity of the designed database.

Figure: 4.2: Physical ER-Diagram



Physical storage estimates

Let's assume that we have 5000 theatres. If there are 6 screens in each theatre and 60 seats in each screen. Also, in an average, there are five shows every day.

Let's assume each seat booking needs 50 bytes (IDs, Number of Seats, ShowID, MovieID, SeatNumbers, SeatStatus, Timestamp, etc.) to store in the

database. We would also need to store information about movies and theatres, let's assume it'll take 50 bytes. So, to store all the data about all shows of all theatres for a day:

5000 theatres * 6 screens * 60 seats * 5 shows * (50+50) bytes = 0.9GB / day

To store five years of this data, we would need around 1.66TB.

Security and access control

The business logic is hidden from the users and is much safer and thus avoids unauthorized or illegal access or database corruption. Security of the user's information is also safe as there is a login facility.

Security and access control are the important aspects in the database to restrict access to unauthenticated users. In this design we have created a user with a username as 'admin' who can insert and update the movie screenings, Theatre information, Movie details, Screen information, Customer and employee details in the database. If the admin wants to delete any information in the table from the database, he/she has to consult/request for the delete access.

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin123';
```

At this point admin has no permissions to do anything with the databases. In fact, even if admin tries to login (with the password, admin123), they will not be able to reach the MySQL shell.

Therefore, the first thing to do is to provide the user with access to the information they will need.

```
grant select,insert on movie_social_distance_1.* to 'admin'@'localhost';
```

The asterisks in this command refer to the movie_social_distance database and tables (respectively) that they can access—this specific command allows to the admin to read and insert across the movie_social_distance database. After finalizing the permissions in the user 'admin' we have to reload all the privileges to make the change to be in effect.

```
FLUSH PRIVILEGES;
```

Movie Ticket Booking System with Social Distancing Term project report

To know the permissions given to ‘admin’ users we can execute the below query.

```
show grants for 'admin'@'localhost';
```

```
2 •  grant select,insert on *.* to 'admin'@'localhost';
3 •  grant update(Is_Available,seat_price) on movie_social_distance_1.seat to 'admin'@'localhost';
4 •  grant update(movie_rating,movie_release) on movie_social_distance_1.movie to 'admin'@'localhost';
5 •  grant update(Theatre_address/Theatre_contact) on movie_social_distance_1.theatre to 'admin'@'localhost';
6 •  grant update(Screen_name,theatre_id) on movie_social_distance_1.screen to 'admin'@'localhost';
7 •  grant update on movie_social_distance_1.employee to 'admin'@'localhost';
8 •  FLUSH PRIVILEGES;
9
10 • show grants for 'admin'@'localhost';
```

The screenshot shows the MySQL Workbench interface with the results of a SQL query. The query was:

```
show grants for 'admin'@'localhost';
```

The results grid displays the following GRANT statements:

```
GRANT SELECT, INSERT ON *.* TO 'admin'@'localhost'  
GRANT INSERT, UPDATE ON 'movie_social_distance_1'.`employee` TO 'admin'@'localhost'  
GRANT INSERT, UPDATE ('movie_rating', `movie_release`) ON 'movie_social_distance_1'.`movie` TO 'admin'@'localhost'  
GRANT INSERT, UPDATE ('Screen_name', `theatre_id`) ON 'movie_social_distance_1'.`screen` TO 'admin'@'localhost'  
GRANT UPDATE ('Is_Available', `seat_price`) ON 'movie_social_distance_1'.`seat` TO 'admin'@'localhost'  
GRANT INSERT, UPDATE ('Theatre_address', 'Theatre_contact') ON 'movie_social_distance_1'.`theatre` TO 'admin'@'localhost'
```

Below the results grid, the 'Action Output' tab shows the following log entries:

#	Time	Action	Message	Duration / Fetch
115	20:42:23	FLUSH PRIVILEGES	0 row(s) affected	0.359 sec
116	20:42:23	show grants for 'admin'@'localhost'	6 row(s) returned	0.000 sec / 0.000 sec

DBMS Selection

DBMS Analysis

The type of database that will be used should support the following needs.

- To store few fixed details of theatre, theatre employees, theatre screen and theatre seats.
- To store few varying details decided by the theatre management like adding/removing movies and adding/removing/modifying the movie shows.
- Provide security to the database since the customer and their booking and payment details are maintained as well.
- Provide sharing of data to attain Master-slave-slave architecture

DBMS Selection

We need RDBMS to achieve ACID properties and also to store relation between the tables. As we serve multiple theatres and each theatre can have multiple screens and in turn each screen has multiple seats, it makes sense to use RDBMS. As the data gets increased only when we add new theatres, it can be managed easily.

As the data should be distributed across the locations, sharing of data is required. For this we can go with Master-slave-slave architecture. In this architecture, Master will be used for writing the latest data and slaves are used to read the data from the master.

Hence MySQL has been chosen to build the designed database for the movie ticket booking system with social distancing.

Hardware and Software requirements

Minimum Hardware Requirements

- A PC with Processor-Pentium-3 ,Speed-1.1 GHz
- RAM :2GB
- Hard Disk :4GB free space

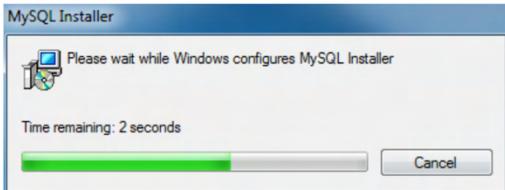
Software Requirements

- Operating System: MAC/Windows XP/7/8/10
- Database: MYSQL

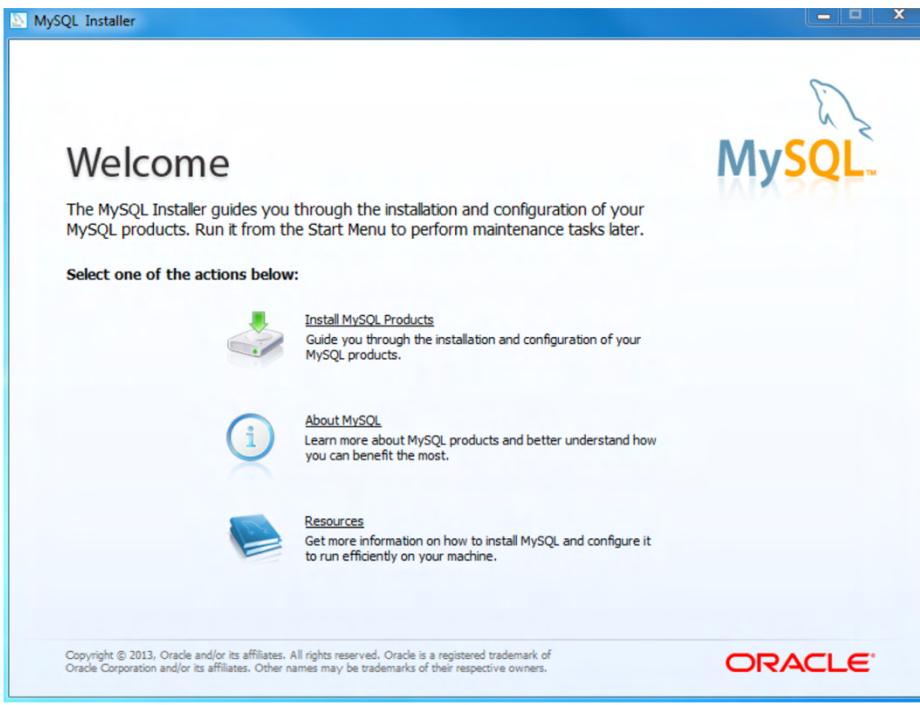
Implementation & Loading

DBMS Installation

To install MySQL using the MySQL installer, double-click on the MySQL installer file and follow the steps below:

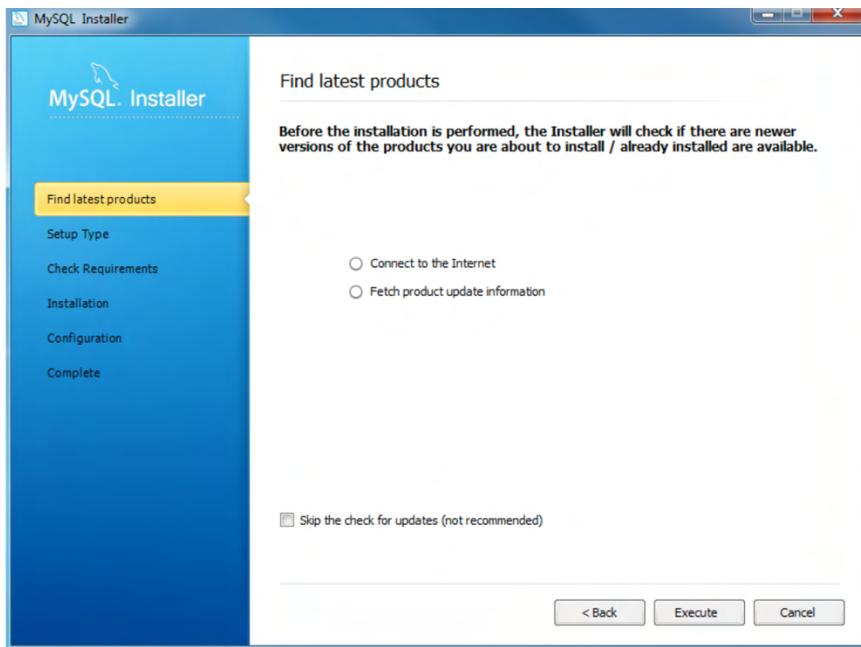


Install MySQL Step 1: Windows configures MySQL Installer

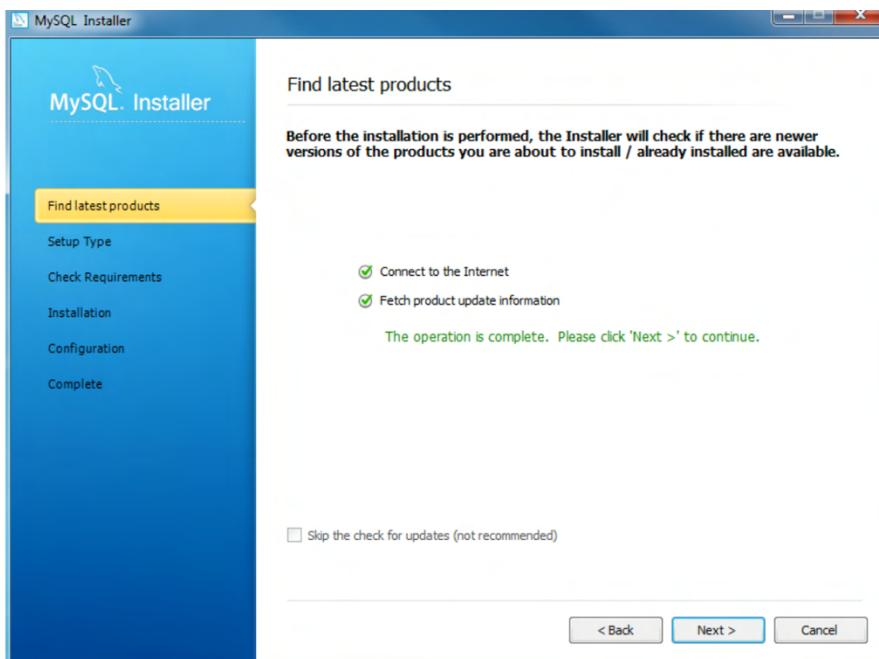


Movie Ticket Booking System with Social Distancing Term project report

Install MySQL Step 2 – Welcome Screen: A welcome screen provides several options. Choose the first option: Install MySQL Products

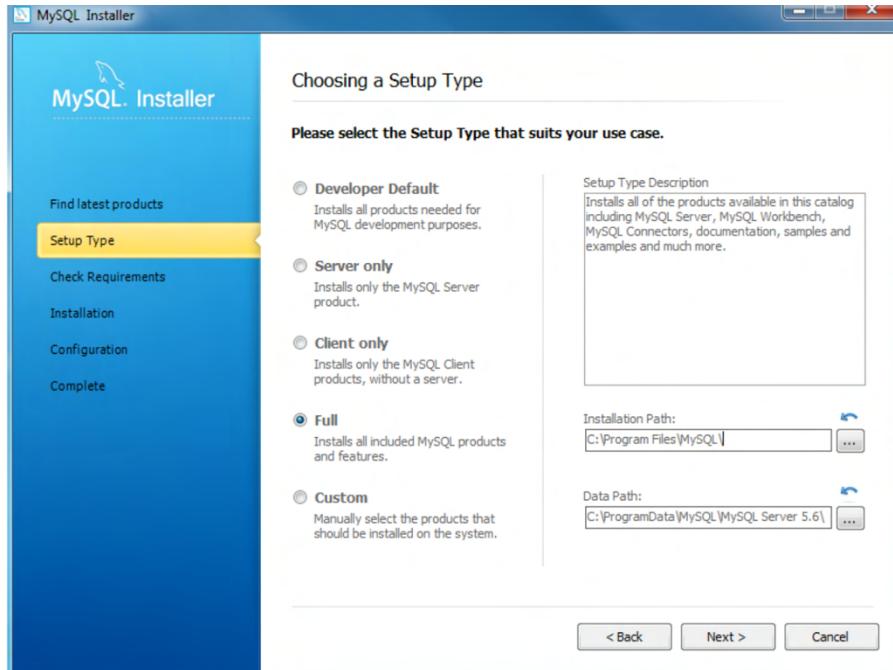


Install MySQL Step 3 – Download the latest MySQL products: MySQL installer checks and downloads the latest MySQL products including MySQL server, MySQL Workbench, etc.

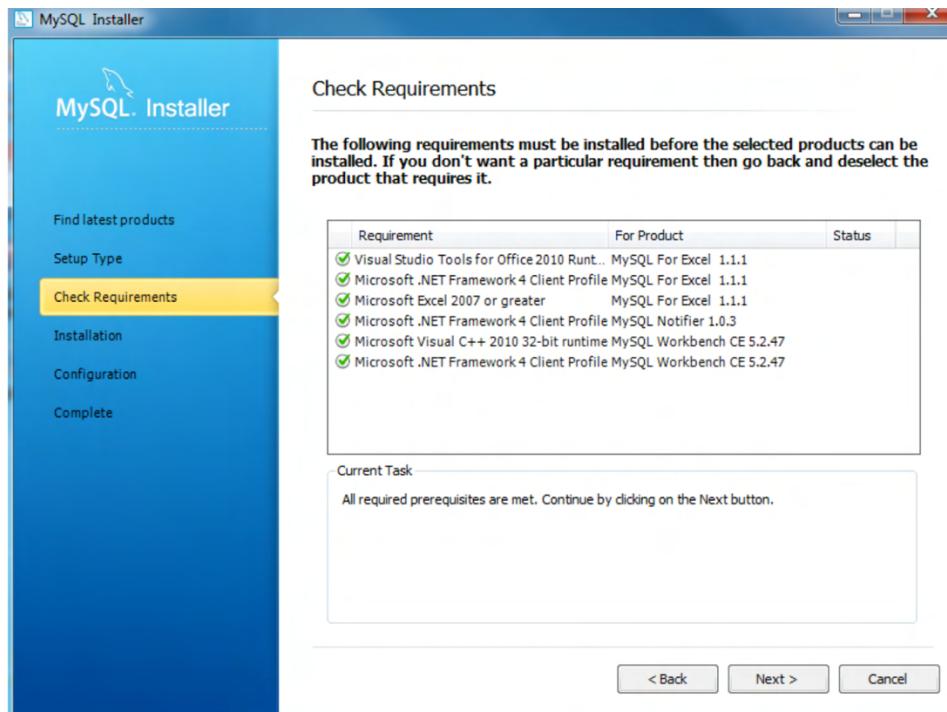


Movie Ticket Booking System with Social Distancing Term project report

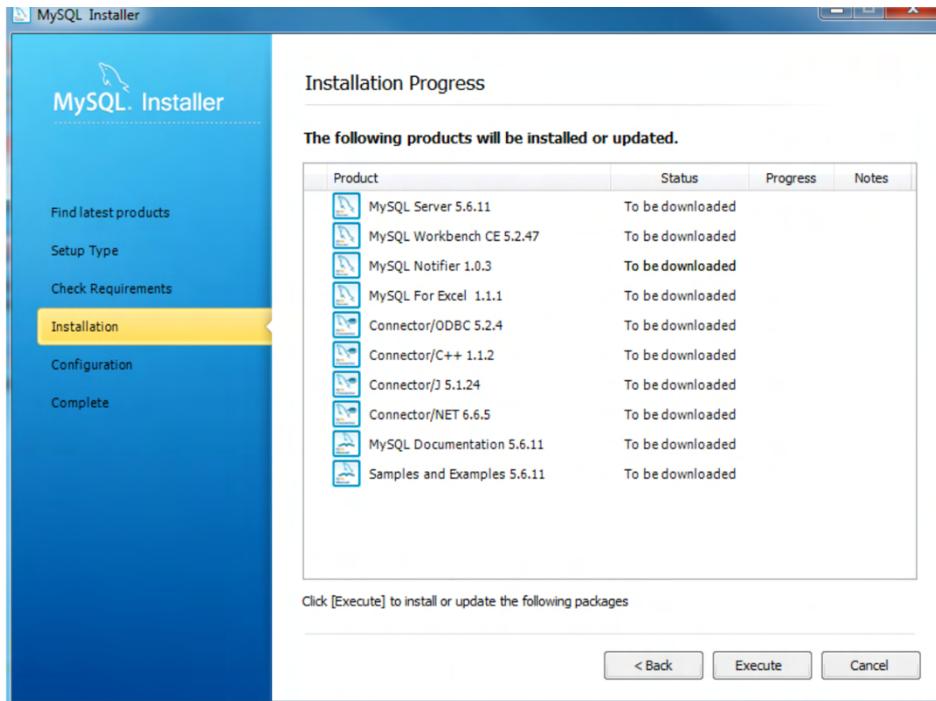
Install MySQL Step 4: Click the Next button to continue



Install MySQL Step 5 – Choosing a Setup Type: there are several setup types available. Choose the Full option to install all MySQL products and features.

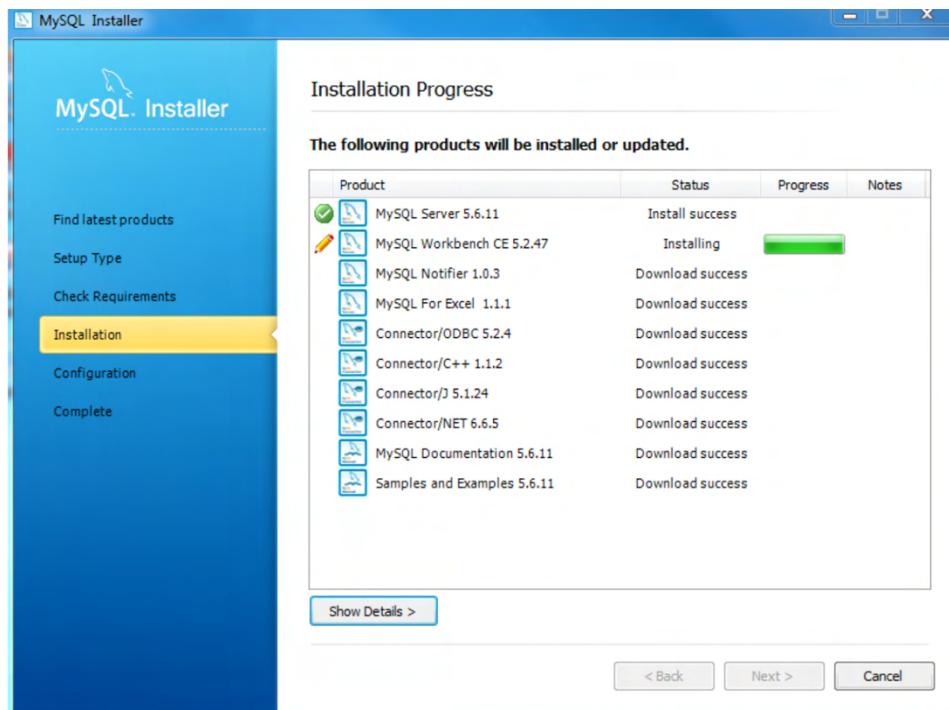


Install MySQL Step 6 – Checking Requirements

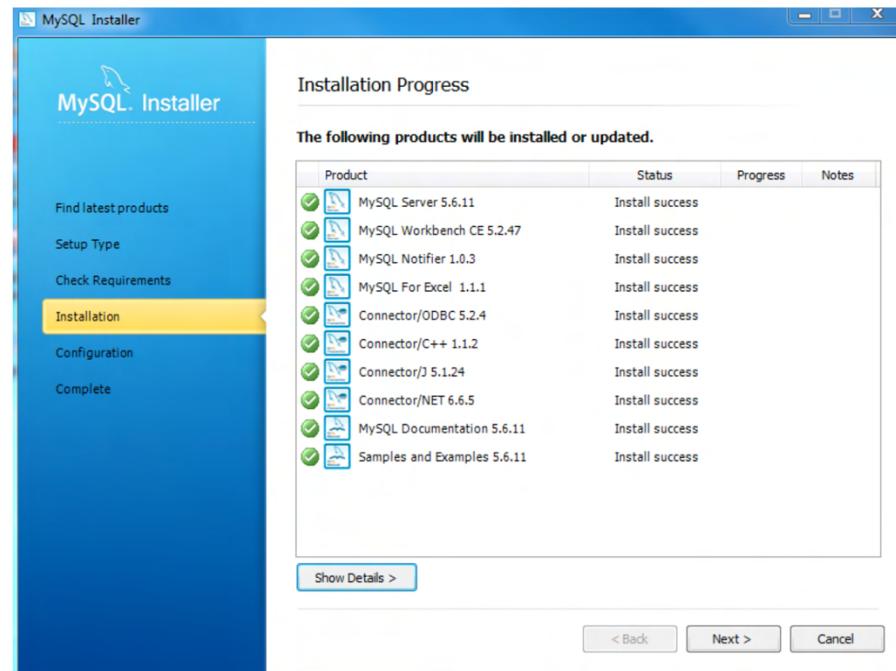


Install MySQL Step 7 – Installation Progress: MySQL Installer downloads all selected products. It will take a while, depending on which products you selected and the speed of your internet connection.

Movie Ticket Booking System with Social Distancing Term project report

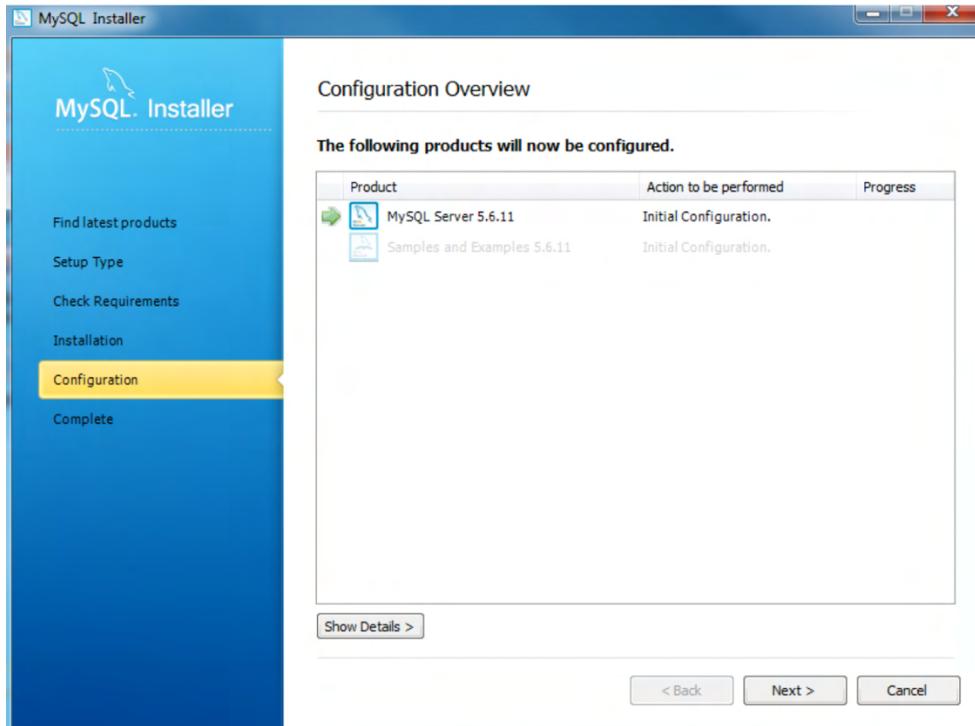


Install MySQL Step 7 – Installation Progress: downloading Products in progress.

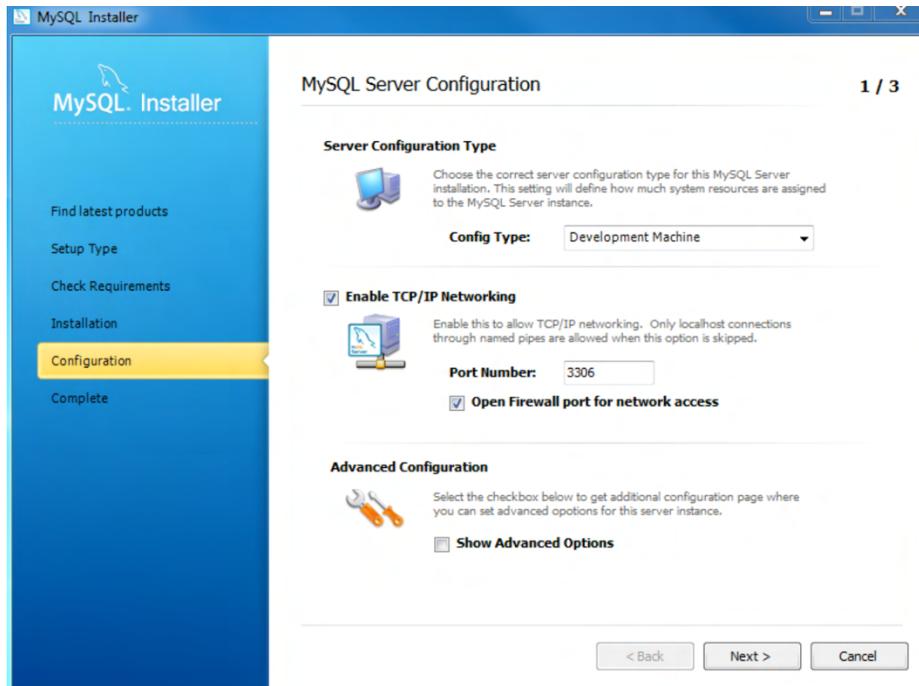


Install MySQL Step 7 – Installation Progress: Complete Downloading. Click the **Next** button to continue...

Movie Ticket Booking System with Social Distancing Term project report



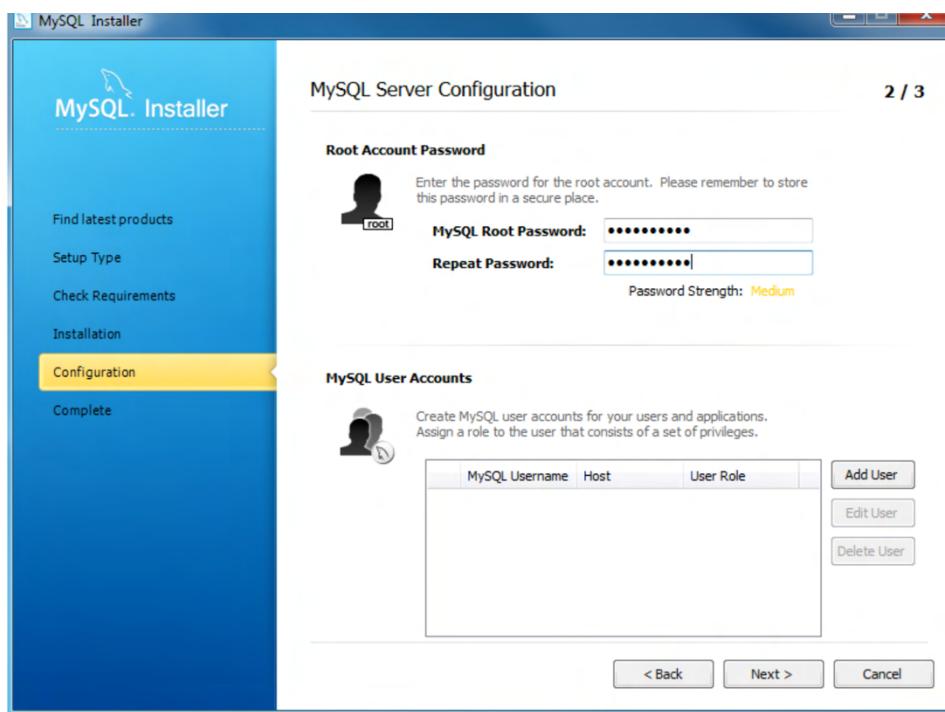
Install MySQL Step 8 – Configuration Overview. Click the Next button to configure MySQL Database Server



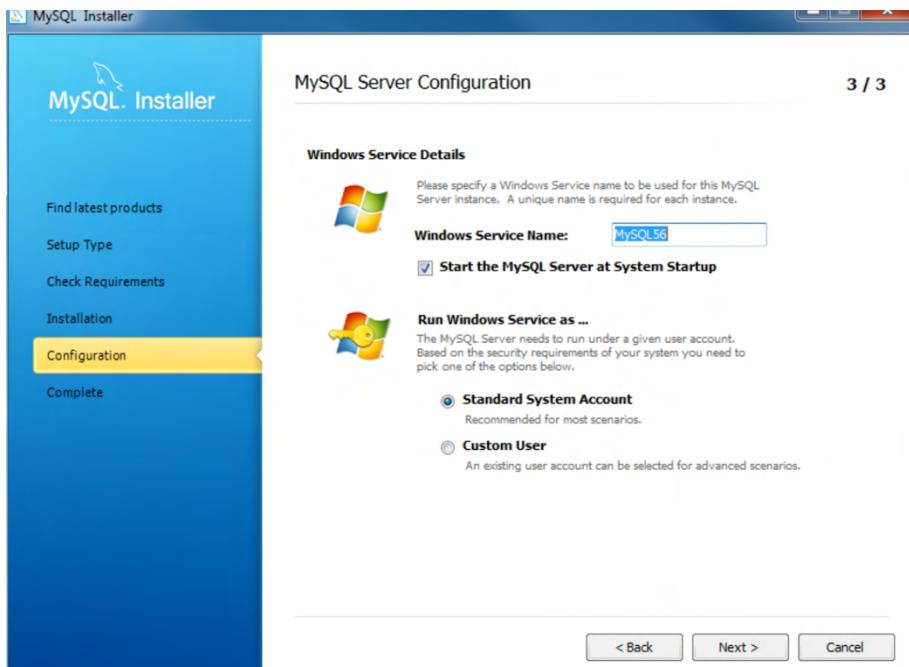
Install MySQL Step 8.1 – MySQL Server Configuration: choose Config Type and MySQL port (3006 by default) and click Next button to continue.

Movie Ticket Booking System with Social Distancing

Term project report

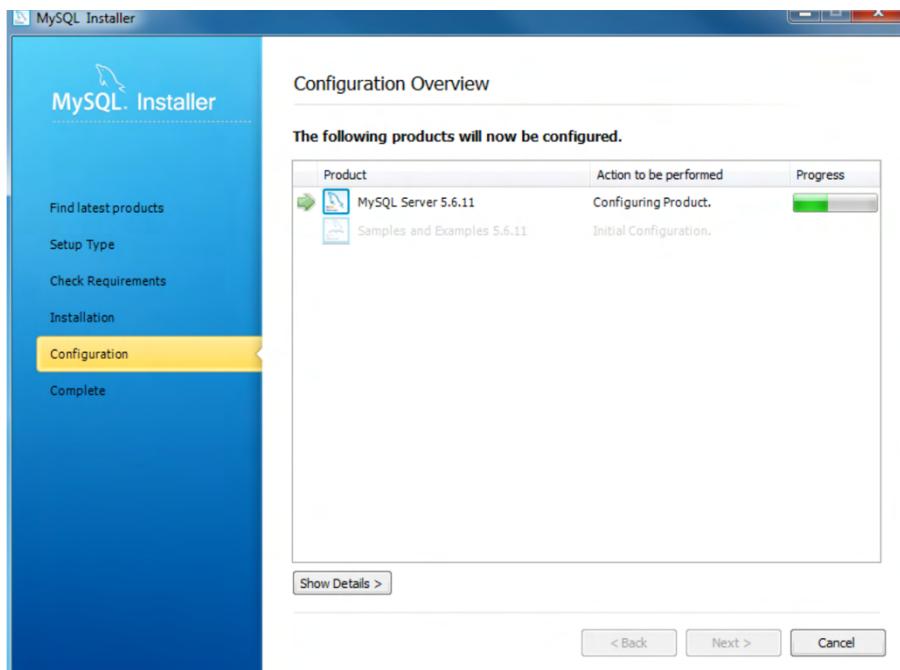


Install MySQL Step 8.1 – MySQL Server Configuration: choose a password for the root account. Please note the password download and keep it securely if you are installing MySQL database server on a production server. If you want to add a more MySQL user, you can do it in this step

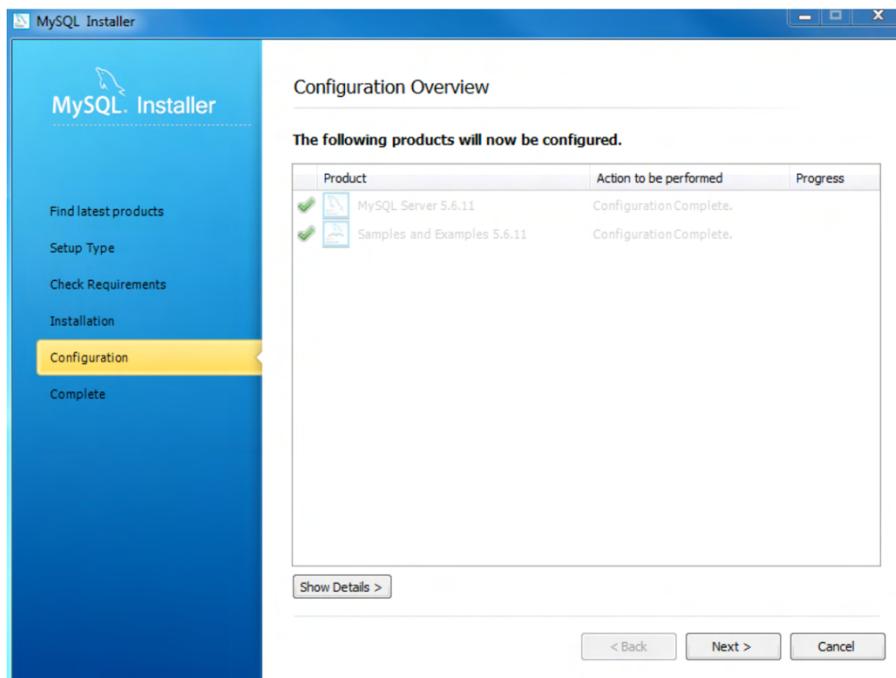


Movie Ticket Booking System with Social Distancing Term project report

Install MySQL Step 8.1 – MySQL Server Configuration: choose Windows service details including Windows Service Name and account type, then click Next button to continue

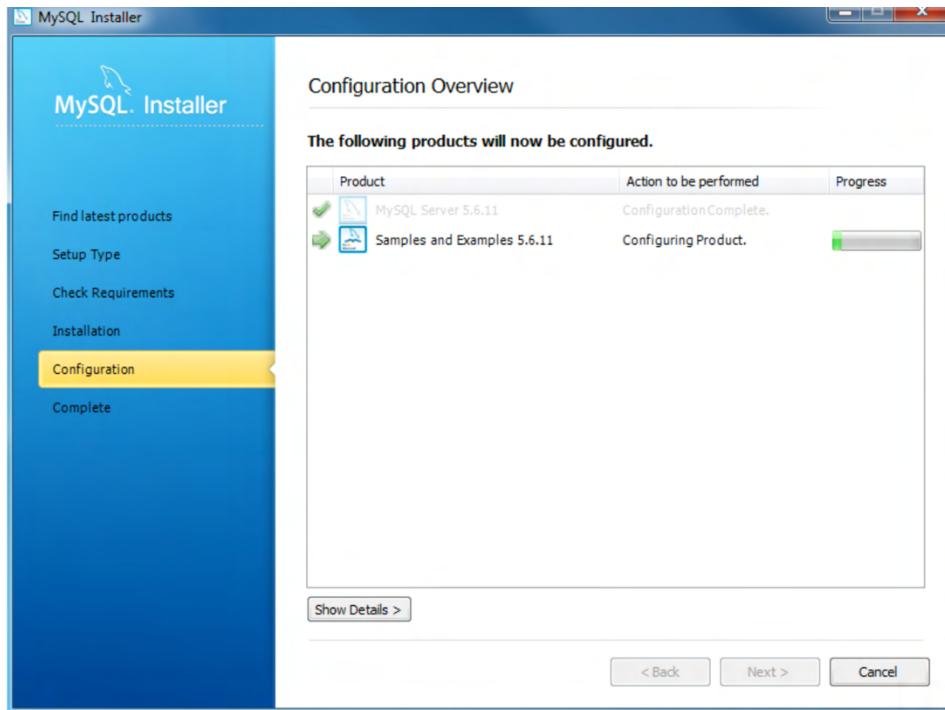


Install MySQL Step 8.1 – MySQL Server Configuration – In Progress: MySQL Installer is configuring MySQL database server. Wait until it is done and click the Next button to continue



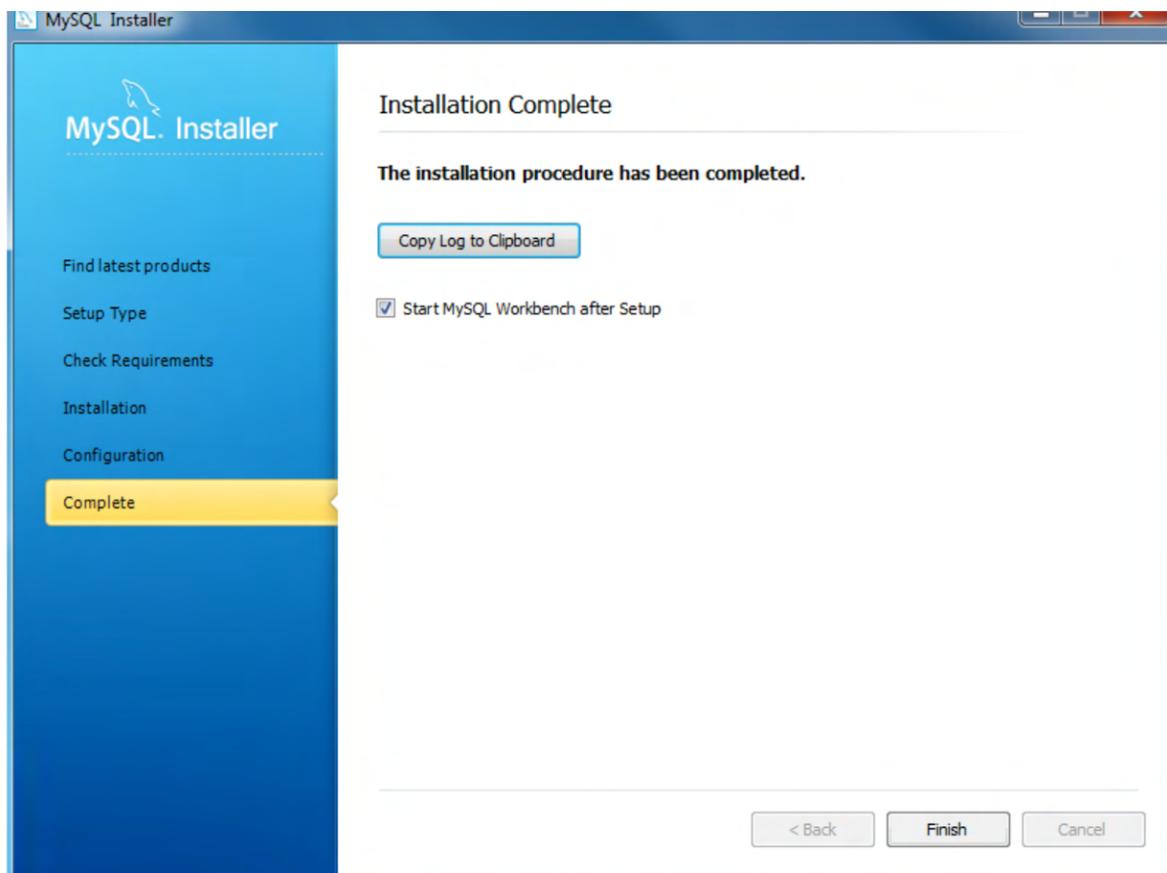
Movie Ticket Booking System with Social Distancing
Term project report

Install MySQL Step 8.1 – MySQL Server Configuration – Done. Click the Next button to continue.



Install MySQL Step 8.2 – Configuration Overview: MySQL Installer installs sample databases and sample models.

Movie Ticket Booking System with Social Distancing
Term project report



Install MySQL Step 9 – Installation Completes: the installation completes. Click the **Finish** button to close the installation wizard and launch the MySQL Workbench.

Schema creation

The SQL code for the creating the designed schema is as follows.

```

CREATE DATABASE IF NOT EXISTS MOVIE_SOCIAL_DISTANCE;
USE MOVIE_SOCIAL_DISTANCE;
#
# Table structure for table 'THEATRE'
#
DROP TABLE IF EXISTS THEATRE;

CREATE TABLE THEATRE (
    Theatre_ID           INTEGER          PRIMARY KEY
    AUTO_INCREMENT,
    Theatre_Name         VARCHAR(100)     NOT NULL,
    Theatre_Contact      VARCHAR(15)      NOT NULL,
    Theatre_Address      VARCHAR(255)     NOT NULL,
    Theatre_Pincode      VARCHAR(6)       NOT NULL,
    CONSTRAINT Theatre_UI1 UNIQUE(Theatre_Contact)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
SET autocommit=1;

ALTER TABLE THEATRE AUTO_INCREMENT = 500;

#
# Table structure for table 'EMPLOYEE'
#
DROP TABLE IF EXISTS EMPLOYEE;

CREATE TABLE EMPLOYEE (
    EMP_ID               INTEGER          PRIMARY KEY
    AUTO_INCREMENT,
    EMP_LastName         VARCHAR(100)     NOT NULL,
    EMP_FirstName        VARCHAR(100)     NOT NULL,
    EMP_Age              INTEGER          NOT NULL,
    EMP_Designation      VARCHAR(50)      NOT NULL,
    EMP_Email             VARCHAR(100)     NOT NULL,
    EMP_Contact           VARCHAR(15)      NOT NULL,
    Theatre_ID            INTEGER,
    CONSTRAINT Emp_theatre_FK FOREIGN KEY (Theatre_ID)
    REFERENCES THEATRE (Theatre_ID),
    CONSTRAINT Employee_UI1 UNIQUE(EMP_Email,EMP_Contact),
    CONSTRAINT Employee_age CHECK(EMP_Age >= 18 AND EMP_AGE
    <=60)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

Movie Ticket Booking System with Social Distancing
Term project report

```
SET autocommit=1;

#
# Table structure for table 'SCREEN'
#
DROP TABLE IF EXISTS SCREEN;

CREATE TABLE SCREEN (
    SCREEN_ID      INTEGER          PRIMARY KEY
AUTO_INCREMENT,
    SCREEN_Name    VARCHAR(50)      NOT NULL,
    Total_Seats    INTEGER          NOT NULL,
    Theatre_ID     INTEGER,
    CONSTRAINT Screen_theatre_FK FOREIGN KEY (Theatre_ID)
REFERENCES THEATRE (Theatre_ID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
SET autocommit=1;

ALTER TABLE SCREEN AUTO_INCREMENT = 1000;

#
# Table structure for table 'SEAT'
#
DROP TABLE IF EXISTS SEAT;

CREATE TABLE SEAT (
    SEAT_ID        INTEGER          PRIMARY KEY
AUTO_INCREMENT,
    ROW_No         CHAR(1)          NOT NULL,
    SEAT_No        INTEGER          NOT NULL,
    Is_Available   BOOL            NOT NULL           DEFAULT 1,
    SEAT_Price     DECIMAL(5,2)     NOT NULL,
    SCREEN_ID      INTEGER,
    CONSTRAINT Seat_screen_FK FOREIGN KEY (SCREEN_ID)
REFERENCES SCREEN (SCREEN_ID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
SET autocommit=1;

ALTER TABLE SEAT AUTO_INCREMENT = 10000;

#
# Table structure for table 'MOVIE'
#
DROP TABLE IF EXISTS MOVIE;

CREATE TABLE MOVIE (
```

Movie Ticket Booking System with Social Distancing
Term project report

```

        MOVIE_ID      INTEGER      PRIMARY KEY
AUTO_INCREMENT,
        MOVIE_Title VARCHAR(100) NOT NULL,
        MOVIE_Desc   VARCHAR(400),
        MOVIE_Genre  VARCHAR(50),
        MOVIE_Language VARCHAR(50),
        MOVIE_Duration TIME       NOT NULL,
        MOVIE_Release DATE      NOT NULL,
        MOVIE_Rating  DECIMAL(4,2)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
SET autocommit=1;

```

```
ALTER TABLE MOVIE AUTO_INCREMENT = 7000;
```

```

#
# Table structure for table 'Movie_SHOW'
#
DROP TABLE IF EXISTS MOVIE_SHOW;

```

```

CREATE TABLE MOVIE_SHOW (
        SHOW_ID      INTEGER      PRIMARY KEY
AUTO_INCREMENT,
        SHOW_Date    DATE        NOT NULL,
        SHOW_Starttime TIME       NOT NULL,
        SHOW_Endtime TIME       NOT NULL,
        Is_SanitizedBOOL NOT NULL      DEFAULT 1,
        SCREEN_ID    INTEGER,
        MOVIE_ID     INTEGER,
        CONSTRAINT Show_Screen_FK FOREIGN KEY (SCREEN_ID)
REFERENCES SCREEN (SCREEN_ID),
        CONSTRAINT Show_Movie_FK FOREIGN KEY (MOVIE_ID)
REFERENCES MOVIE (MOVIE_ID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
SET autocommit=1;

```

```
ALTER TABLE MOVIE_SHOW AUTO_INCREMENT = 50000;
```

```

#
# Table structure for table 'CUSTOMER'
#
DROP TABLE IF EXISTS CUSTOMER;

```

```

CREATE TABLE CUSTOMER (
        CUST_ID      INTEGER      PRIMARY KEY
AUTO_INCREMENT,
        CUST_LastName VARCHAR(100) NOT NULL,

```

Movie Ticket Booking System with Social Distancing
Term project report

```

CUST_FirstName  VARCHAR(100)      NOT NULL,
CUST_Age        INTEGER          NOT NULL,
CUST_Username   VARCHAR(50)       NOT NULL,
CUST_Pwd        VARCHAR(50)      NOT NULL,
CUST_Email      VARCHAR(100)      NOT NULL,
CUST_Contact    VARCHAR(15)       NOT NULL,
CUST DOJ        DATE,           CONSTRAINT Customer_UI1
UNIQUE(CUST_Username,CUST_Email,CUST_Contact),
CONSTRAINT Customer_age CHECK(CUST_Age >= 18)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
SET autocommit=1;

```

```
ALTER TABLE CUSTOMER AUTO_INCREMENT = 200;
```

```

#
# Table structure for table 'BOOKING'
#
DROP TABLE IF EXISTS BOOKING;

```

```

CREATE TABLE BOOKING (
    BOOKING_ID          INTEGER      PRIMARY KEY
    AUTO_INCREMENT,
    No_Of_Seats         INTEGER      NOT NULL,
    BOOKING_Status      VARCHAR(10)  NOT NULL,
    BOOKING_Price       DECIMAL(8,2) NOT NULL,
    BOOKING_Time        TIMESTAMP    NOT NULL,
    SHOW_ID              INTEGER,
    CUST_ID              INTEGER,
    CONSTRAINT Booking_Show_FK FOREIGN KEY (SHOW_ID)
    REFERENCES MOVIE_SHOW (SHOW_ID),
    CONSTRAINT Booking_Customer_FK FOREIGN KEY (CUST_ID)
    REFERENCES CUSTOMER (CUST_ID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
SET autocommit=1;

```

```
ALTER TABLE BOOKING AUTO_INCREMENT = 30000;
```

```

#
# Table structure for table 'SHOW_SEAT'
#
DROP TABLE IF EXISTS SHOW_SEAT;

```

```

CREATE TABLE SHOW_SEAT (
    SHOW_SEAT_ID         INTEGER      PRIMARY KEY
    AUTO_INCREMENT,

```

Movie Ticket Booking System with Social Distancing
Term project report

```
        SHOW_SEAT_Status BOOL          NOT NULL
DEFAULT 1,
        SEAT_ID           INTEGER,
        SHOW_ID           INTEGER,
        BOOKING_ID        INTEGER,
        CONSTRAINT Showseat_Seat_FK FOREIGN KEY (SEAT_ID)
REFERENCES SEAT (SEAT_ID),
        CONSTRAINT Showseat_Movieshow_FK FOREIGN KEY (SHOW_ID)
REFERENCES MOVIE_SHOW (SHOW_ID),
        CONSTRAINT Showseat_Booking_FK FOREIGN KEY (BOOKING_ID)
REFERENCES BOOKING (BOOKING_ID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
SET autocommit=1;
```

```
ALTER TABLE SHOW_SEAT AUTO_INCREMENT = 70000;
```

```
#  
# Table structure for table 'PAYMENT'  
#  
DROP TABLE IF EXISTS PAYMENT;
```

```
CREATE TABLE PAYMENT (
        PAYMENT_ID          INTEGER  PRIMARY KEY
AUTO_INCREMENT,
        PAYMENT_Mode        VARCHAR(15),
        PAYMENT_Sub_Total DECIMAL(8,2)    NOT NULL,
        PAYMENT_Status      VARCHAR(10),
        Total_With_Tax     DECIMAL(8,2)    NOT NULL,
        PAYMENT_Time       TIMESTAMP,
        Remote_Trans_ID   VARCHAR(21),
        BOOKING_ID         INTEGER,
        CONSTRAINT Payment_Booking_FK FOREIGN KEY (BOOKING_ID)
REFERENCES BOOKING (BOOKING_ID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
SET autocommit=1;
```

```
ALTER TABLE PAYMENT AUTO_INCREMENT = 50;
```

Movie Ticket Booking System with Social Distancing

Term project report

Below is the screenshot of the above code after execution.

The screenshot shows the MySQL Workbench interface with the following details:

- Schema:** MOVIE_SOCIAL_DISTANCE
- Query Editor:** Query 3 titled "project_table_creation".
- Code Content:**

```

1 • CREATE DATABASE IF NOT EXISTS MOVIE_SOCIAL_DISTANCE;
2 • USE MOVIE_SOCIAL_DISTANCE;
3 #
4 # Table structure for table 'THEATRE'
5 #
6 • DROP TABLE IF EXISTS THEATRE;
7
8 • CREATE TABLE THEATRE (
9     Theatre_ID      INTEGER      PRIMARY KEY      AUTO_INCREMENT,
10    Theatre_Name    VARCHAR(100)  NOT NULL,
11    Theatre_Contact VARCHAR(15)   NOT NULL,
12    Theatre_Address VARCHAR(255)  NOT NULL,
13    Theatre_Pincode VARCHAR(6)   NOT NULL,
14    CONSTRAINT Theatre_U1 UNIQUE(Theatre_Contact)
15 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
16 • SET autocommit=1;
17
18 • ALTER TABLE THEATRE AUTO_INCREMENT = 500;
19
20 #
21 # Table structure for table 'EMPLOYEE'
22 #
23 • DROP TABLE IF EXISTS EMPLOYEE;
24
25 • CREATE TABLE EMPLOYEE (
26     EMP_ID        INTEGER      PRIMARY KEY      AUTO_INCREMENT,
27     EMP_LastName  VARCHAR(100)  NOT NULL,

```
- Action Output:** Shows the execution log with 139 entries, indicating successful creation of tables and constraints.

Data Loading & Conversion

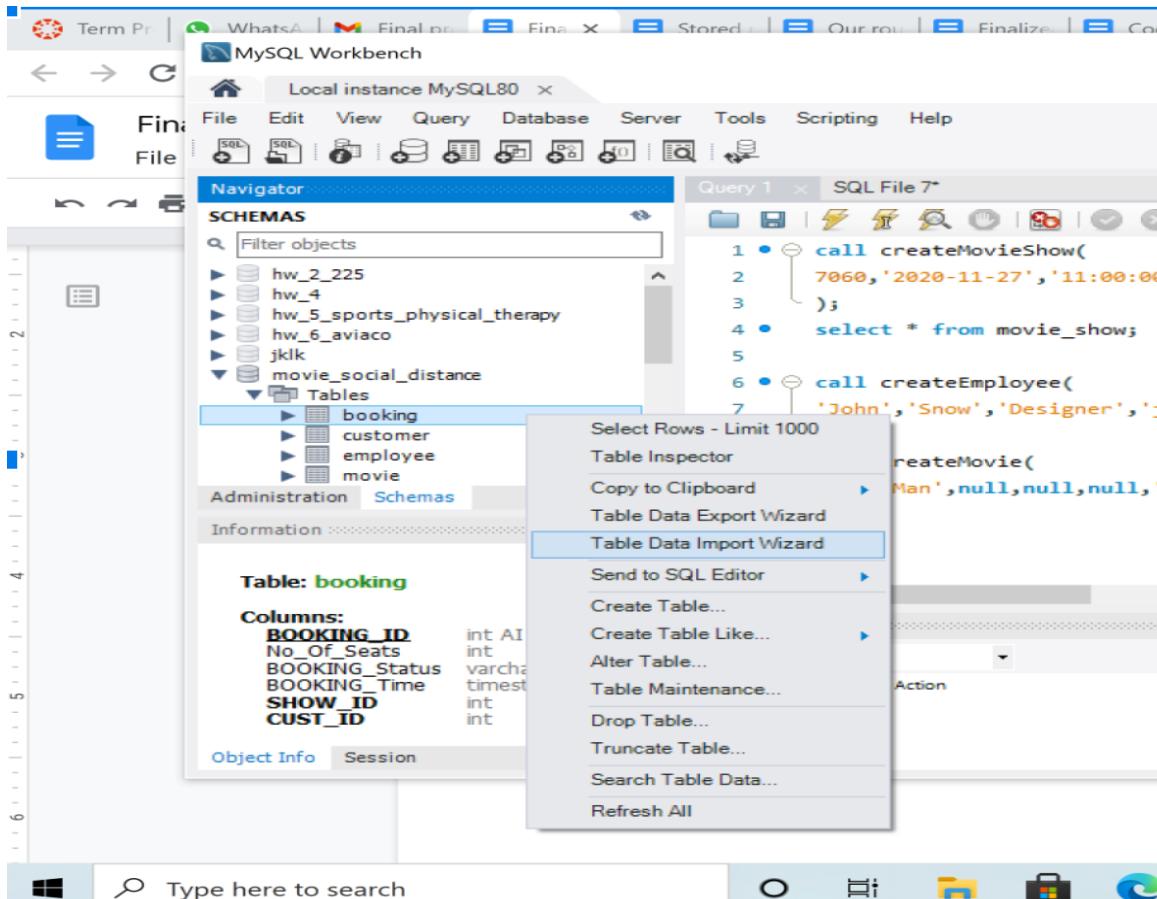
Data has been loaded into tables from CSV files using table data import wizard which supports import and export operations using CSV and JSON files and several configuration options are included such as separators, column selection, encoding selection, and more. The wizard can be performed in local or remotely connected MySQL servers, and the column, type and table mapping can be imported.

CSV File:

	BOOKING	No_Of_Sel	Price	BOOKING_Status	BOOKING_Time	SHOW_ID	CUST_ID
2	30000	2	40	Success	4/12/2020 12:00	50000	200
3	30001	3	60	Success	4/12/2020 12:00	50000	201
4	30002	1	20	Success	4/12/2020 12:00	50000	202
5	30003	2	40	Success	4/12/2020 12:00	50001	203

Movie Ticket Booking System with Social Distancing Term project report

This wizard can be accessed from the table's context menu by right-clicking on a table and choose **Table Data Import Wizard**, as the below figure shows and choose the path from which the csv file has to be imported.



And it maps the columns with table schema as shown

Movie Ticket Booking System with Social Distancing Term project report

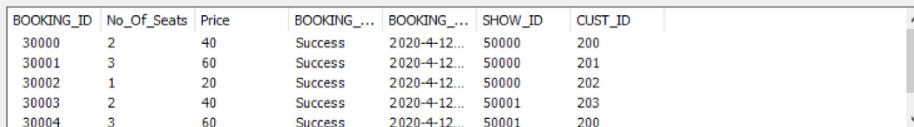
Configure Import Settings

Detected file format: csv 

Encoding: utf-8

Columns:

Source Column	Dest Column
<input checked="" type="checkbox"/> BOOKING_ID	BOOKING_ID
<input checked="" type="checkbox"/> No_Of_Seats	No_Of_Seats
<input checked="" type="checkbox"/> Price	BOOKING_Status
<input checked="" type="checkbox"/> BOOKING_Status	BOOKING_Status
<input checked="" type="checkbox"/> BOOKING_Time	BOOKING_Time
<input checked="" type="checkbox"/> SHOW_ID	SHOW_ID


BOOKING_ID | No_Of_Seats | Price | BOOKING_Status | BOOKING_Time | SHOW_ID | CUST_ID |
30000 | 2 | 40 | Success | 2020-4-12... | 50000 | 200 |
30001 | 3 | 60 | Success | 2020-4-12... | 50000 | 201 |
30002 | 1 | 20 | Success | 2020-4-12... | 50000 | 202 |
30003 | 2 | 40 | Success | 2020-4-12... | 50001 | 203 |
30004 | 3 | 60 | Success | 2020-4-12... | 50001 | 200 |

< Back | Next > | Cancel

And if the format matches with the data type, it successfully imports the data from csv into the table.

Import Data

The following tasks will now be performed. Please monitor the execution.

Prepare Import
 Import data file

Finished performing tasks. Click [Next >] to continue.

Show Logs | < Back | Next > | Cancel

Testing and Evaluation

Id	Action	Name	Expected Working	Test Results	Screenshot
i)	Event	SANITIZE_Show	Should update the Is_Sanitized field to 0 after the end of each show based on the SHOW_Endtime	Event updates the Is_Sanitized field to 0 after the SHOW_Endtime	Attached
ii)	Trigger	BOOK_SHOW_BY_STATUS	If payment_status change to 'failure' then the booking_status should update to 'failure' and the entries in the SHOW_SEATS table for the corresponding booking_id should be deleted	Worked as expected when the payment_status was updated with 'failure'	Attached
iii)	Trigger	upd_user	Set the customer password to the old password in the CUSTOMER table, if the given password to change is NULL or empty by the customer. Else set to the new password.	Worked as expected	Attached
iv)	Trigger	upd_seat_price	Update the price of the seat in each screen based on the price range of the	The seat price gets updated as per pre-defined price	Attached

Movie Ticket Booking System with Social Distancing
Term project report

			row number in the SEAT table. The price range is default and set at the time of design.	range based on the row numbers	
v)	Stored Procedure	BLOCK_ALTERNATE_ROWS	Update the Is_Available field in the SEAT table to 0 for alternate rows	The IS_Available field is set to 0 for alternate fields	Attached
vi)	Stored Procedure	createEmployee	Insert the given employee details in the EMPLOYEE table	The employee details are being inserted in the EMPLOYEE table as expected	Attached
vii)	Stored Procedure	createCustomer	Insert the given customer details into the CUSTOMER table	On execution, the CUSTOMER table is populated with the given details	Attached
viii)	Stored Procedure	createMovie	Insert the new movie details in the MOVIE table	The movie details are inserted into the MOVIE table	Attached
ix)	Stored Procedure	createMovishow	Add the given movie show entries into the MOVIE_SHOW table	The details are entered in the MOVIE_SHOW table	Attached
x)	Stored Procedure	THEATRE_REMOVE	To delete a THEATRE_ID from the THEATRE table, <ul style="list-style-type: none"> • Update the movie_id to NULL in the MOVIE_SHOW table • Delete the entries in the SHOW_SEAT table which 	<ul style="list-style-type: none"> • The movie_id is updated to NULL in the MOVIE_SHOW table • The SHOW_SEAT entries for the theatre_id is getting deleted. 	Attached

Movie Ticket Booking System with Social Distancing
Term project report

			<p>are linked to the show_id that are running on the screen_id in the given theatre_id</p> <ul style="list-style-type: none"> • Delete the entries in the PAYMENT table that are linked to the theatre_id • Delete the entries in the BOOKING table mapped to the given theatre_id • Delete the entry from MOVIE_SH OW that are mapped to the theatre_id via the screen_id • Delete the entries in the SEAT and SCREEN tables that are mapped to the given theatre_id • Delete the EMPLOYEE table entries that are mapped to 	<ul style="list-style-type: none"> • The entries in the PAYMENT table are deleted for the given theatre_id • The BOOKING table entries are deleted for the theatre_id • The entries of MOVIE_SH OW table is getting deleted for the theatre_id • The SEAT table entry is deleted for the theatre_id • The SCREEN table entries are deleted for the given theatre_id • The EMPLOYEE table entries are being deleted for the given theatre_id • The theatre entry for the given THEATRE_I
--	--	--	---	--

Movie Ticket Booking System with Social Distancing
Term project report

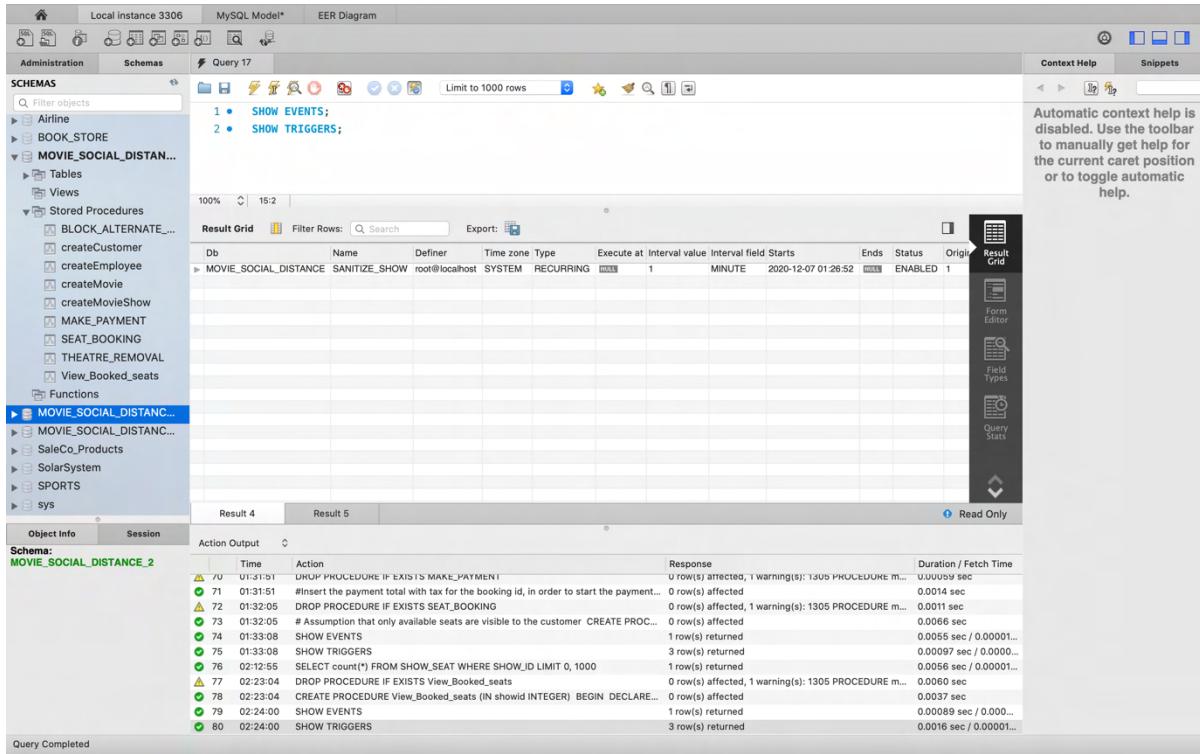
			<p>the given theatre id</p> <ul style="list-style-type: none"> Finally delete the theatre entry for the given theatre_id in the THEATRE table 	D is deleted in the THEATRE table.	
xi)	Stored Procedure	View_Booked_seats	Should return the total number of seats that are been booked by the customers for a given movie show	Returns the count of booked tickets for the given show id.	Attached
xii)	Stored Procedure	MAKE_PAYMENT	Insert into the PAYMENT table with the payment_subtotal,to tal_with_tax,bookin g id	The PAYMENT table is updated with the values of payment_subtotal,to tal_with_tax,bookin g id fields.	Attached
xiii)	Stored Procedure	SEAT_BOOKING	Insert the BOOKING table with all the details and insert into the SHOW_SEAT table with all the details of the seat and show that is booked with SHOW_SEAT_STA TUS as 0	The booking details are entered into BOOKING details and the SHOW_SEAT table is updated with the SHOW_SEAT_STA TUS as 0 for the corresponding booking_id	Attached

Movie Ticket Booking System with Social Distancing

Term project report

Screenshots

Screenshots of all event, triggers and stored procedure in the designed database.



The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
1 • SHOW EVENTS;
2 • SHOW TRIGGERS;
```

The results are displayed in two tabs: 'Result Grid' and 'Result 5'. The 'Result Grid' tab shows a single row of data for a trigger named 'SANITIZE_SHOW'. The 'Result 5' tab shows a detailed log of 80 actions, including schema creation, procedure execution, and trigger creation.

Action Output	Time	Action	Response	Duration / Fetch Time
	70	01:31:51	DROP PROCEDURE IF EXISTS `MAKE_PAYMENT`	0.0000 sec
	71	01:31:51	#Insert the payment total with tax for the booking id, in order to start the payment...	0.0014 sec
	72	01:32:05	DROP PROCEDURE IF EXISTS `SEAT_BOOKING`	0.0011 sec
	73	01:32:05	# Assumption that only available seats are visible to the customer. CREATE PROC...	0.0066 sec
	74	01:33:00	SHOW EVENTS	0.0055 sec / 0.00001...
	75	01:33:00	SHOW TRIGGERS	0.00097 sec / 0.0000...
	76	02:12:55	SELECT count(*) FROM SHOW_SEAT WHERE SHOW_ID LIMIT 0, 1000	0.0056 sec / 0.00001...
	77	02:23:04	DROP PROCEDURE IF EXISTS `View_Booked_seats`	0.0060 sec
	78	02:23:04	CREATE PROCEDURE `View_Booked_seats` (IN showid INTEGER) BEGIN DECLARE...	0.0037 sec
	79	02:24:00	SHOW EVENTS	0.00089 sec / 0.0000...
	80	02:24:00	SHOW TRIGGERS	0.0016 sec / 0.00001...

Movie Ticket Booking System with Social Distancing

Term project report

The screenshot shows the MySQL Workbench interface with the following details:

- Top Bar:** Local instance 3306, MySQL Model*, EER Diagram.
- Schemas:** MOVIE_SOCIAL_DISTANCING selected.
- Query Editor:** Query 17, containing the following SQL:


```

1 • SHOW EVENTS;
2 • SHOW TRIGGERS;
      
```
- Result Grid:** Shows the results of the triggers and events. The triggers listed are:

Trigger	Event	Table	Statement	Timing	Created
upd_user	UPDATE	CUSTOMER	BEGIN IF (NEW.cust_pwd IS NULL OR NEW.cust_pwd = '') THEN SET N...	BEFORE	2020-12-07 01:28:06.83
BOOK_SHOW_BYSTATUS	UPDATE	PAYMENT	BEGIN IF NEW.PAYMENT_Status = 'Failure' THEN UPDATE BOOKING S...	BEFORE	2020-12-07 01:26:56.60
upd_seat_price	INSERT	SEAT	BEGIN IF NEW.ROW_No = 'A' OR NEW.ROW_No = 'B' THEN SET NEW.SEAT...	BEFORE	2020-12-07 01:28:40.80
- Right Panel:** Context Help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
- Log:** Shows the history of actions taken during the session, including the creation of procedures and triggers.

Screenshots of the testing of the above actions.

SANITIZE_Show

Execution of the event:

Movie Ticket Booking System with Social Distancing

Term project report

```

1 • DROP EVENT IF EXISTS SANITIZE_SHOW;
2
3 • SET GLOBAL event_scheduler = ON;
4
5 • CREATE EVENT SANITIZE_SHOW
6   ON SCHEDULE EVERY 1 MINUTE
7   STARTS CURRENT_TIMESTAMP + INTERVAL 1 MINUTE
8 DO
9   UPDATE MOVIE_SHOW SET Is_Sanitized = 0 WHERE SHOW_Date = DATE_FORMAT(NOW(), '%Y-%m-%d')
10  AND DATE_FORMAT(SHOW_Endtime, '%h:%i') = DATE_FORMAT(NOW(), '%h:%i');
11

```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Object Info Session

Table: PAYMENT

Columns:

PAYMENT_ID	int PK
PAYMENT_Mode	varchar(15)
PAYMENT_Sub_Tot	decimal(8,2)
Remote_Trans_ID	varchar(10)
PAYOUT_Time	timestamp
BOOKING_ID	int
Total_Wth_Tax	decimal(8,2)
PAYOUT_Status	varchar(10)
BOOKING_ID	int

Action Output

Time	Action	Response	Duration / Fetch Time
213 16:24:27	SHOW COLUMNS FROM 'MOVIE_SOCIAL_DISTANCE3'.'payment'	OK	0.000 sec
214 16:24:29	PREPARE stmt FOR INSERT INTO 'MOVIE_SOCIAL_DISTANCE3'.'payment' ('P...')	OK	0.000 sec
215 16:27:29	DEALLOCATE PREPARE stmt	OK	0.000 sec
216 16:27:12	DROP EVENT IF EXISTS SANITIZE_SHOW	0 row(s) affected, 1 warning(s): 1305 Event SANITIZE... 0.0031 sec	
217 16:27:12	SET GLOBAL event_scheduler = ON	0 row(s) affected	0.0010 sec
218 16:27:12	CREATE EVENT SANITIZE_SHOW ON SCHEDULE EVERY 1 MINUTE STARTS CU...	0 row(s) affected	0.0062 sec

Query Completed

It runs every minute. Screen shot taken at local time is 20:09:00 on 2020-12-03. Since the end time is 20:30:00, the Is_Sanitized is still 1.

```

1 • SELECT * FROM MOVIE_SOCIAL_DISTANCE_2.MOVIE_SHOW;

```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Object Info Session

Table: SHOW_SEAT

Columns:

SHOW_SEAT_ID	int AI PK
SHOW_SEAT_Status	tinyint(1)
SEAT_ID	int
SHOW_ID	int
BOOKING_ID	int
SHOW_ID	NULL
BOOKING_ID	NULL
SHOW_SEAT_Status	NULL
SEAT_ID	NULL

Action Output

Time	Action	Response	Duration / Fetch Time
376 19:37:33	DROP TRIGGER IF EXISTS UPDATE_CUSTOMER_D0J	0 row(s) affected	0.0026 sec
377 19:37:33	CREATE TRIGGER UPDATE_CUSTOMER_D0J BEFORE INSERT ON CUSTOMER FOR... 0 row(s) affected	0.0027 sec	
378 19:37:36	insert into customer (CUST_LastName,CUST_FirstName,CUST_Age,CUST_Users... Error Code: 1442. Can't update table 'customer' in st... 0 row(s) affected	0.0018 sec	
379 19:39:53	DROP TRIGGER IF EXISTS UPDATE_CUSTOMER_D0J	0 row(s) affected	0.0028 sec
380 19:39:53	CREATE TRIGGER UPDATE_CUSTOMER_D0J BEFORE INSERT ON CUSTOMER FOR... 0 row(s) affected	0.0027 sec	
381 19:39:56	insert into customer (CUST_LastName,CUST_FirstName,CUST_Age,CUST_Users... 1 row(s) affected	0.0016 sec	
382 19:40:01	SELECT * FROM MOVIE_SOCIAL_DISTANCE_2.CUSTOMER LIMIT 0, 1000	4 row(s) returned	0.00033 sec / 0.000...

SQL script saved to '/Users/swarupagowri/Desktop/PROJECT/project_table.creation.sql'

Screen shot taken at local time is 20:30:00 on 2020-12-03. The Is_Sanitized has been updated to 0.

Movie Ticket Booking System with Social Distancing

Term project report

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
1 • SELECT * FROM MOVIE_SOCIAL_DISTANCE_2.MOVIE_SHOW;
```

The results grid displays data from the MOVIE_SHOW table:

SHOW_ID	SHOW_Date	SHOW_Starttime	SHOW_Endtime	Is_Sanitized	SCREEN_ID	MOVIE_ID
50000	2020-12-03	17:00:00	18:30:00	1	1000	7000
50001	2020-12-03	17:00:00	18:30:00	1	1001	7001
50003	2020-12-03	17:00:00	18:30:00	1	1002	7000
50003	2020-12-03	17:00:00	18:30:00	1	1003	7001
50004	2020-12-03	19:00:00	20:30:00	0	1000	7000
50005	2020-12-03	19:00:00	20:30:00	0	1001	7001
50006	2020-12-03	19:00:00	20:30:00	0	1002	7000
50007	2020-12-03	19:00:00	20:30:00	0	1003	7001

Below the results grid, there is a detailed log of the execution steps:

Action	Time	Action	Response	Duration / Fetch Time
SELECT * FROM MOVIE_SOCIAL_DISTANCE_2.PAYMENT LIMIT 0, 1000	20:25:02	16 row(s) returned	0.00031 sec / 0.0000...	
SELECT * FROM MOVIE_SOCIAL_DISTANCE_2.SHOW_SEAT LIMIT 0, 1000	20:25:06	65 row(s) returned	0.0039 sec / 0.0000...	
update payment set payment_status = 'Failure' where booking_id = 30000	20:33:49	1 row(s) affected Rows matched: 1 Changed: 1 Warni...	0.012 sec	
SELECT * FROM MOVIE_SOCIAL_DISTANCE_2.PAYMENT LIMIT 0, 1000	20:33:53	16 row(s) returned	0.00039 sec / 0.0000...	
SELECT * FROM MOVIE_SOCIAL_DISTANCE_2.SHOW_SEAT LIMIT 0, 1000	20:34:13	16 row(s) returned	0.00028 sec / 0.0000...	
SELECT * FROM MOVIE_SOCIAL_DISTANCE_2.SHOW_SEAT LIMIT 0, 1000	20:34:29	65 row(s) returned	0.00027 sec / 0.0000...	
SELECT * FROM MOVIE_SOCIAL_DISTANCE_2.MOVIE_SHOW LIMIT 0, 1000	20:36:15	8 row(s) returned	0.00026 sec / 0.0000...	

BOOK_SHOW_BYSTATUS

Execution of the trigger:

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
1 • DROP TRIGGER IF EXISTS BOOK_SHOW_BYSTATUS;
```

The results grid displays data from the PAYMENT table:

PAYMENT_ID	PAYMENT_Mode	PAYMENT_Sub_Tot	PAYMENT_Status	Total_With_Tax	PAYMENT_Time	Remote_Trans_ID	BOOKING_ID
219	16-39-21	SELECT * FROM MOVIE_SOCIAL_DISTANCE_3.SHOW_SEAT LIMIT 0, 1000	4 row(s) returned	0.0010 sec / 0.00002...			
220	16-39-29	SELECT * FROM MOVIE_SOCIAL_DISTANCE_3.PAYMENT LIMIT 0, 1000	41 row(s) returned	0.0019 sec / 0.00001...			
221	16-39-12	DELETE FROM SHOW_SEAT WHERE BOOKING_ID = 30002	11 rows affected	0.0011 sec			
222	16-39-16	SELECT * FROM MOVIE_SOCIAL_DISTANCE_3.SHOW_SEAT LIMIT 0, 1000	3 row(s) returned	0.00034 sec / 0.000...			
223	16-39-23	DROP TRIGGER IF EXISTS BOOK_SHOW_BYSTATUS	0 row(s) affected, 1 warning(s): 1360 Trigger does no...	0.0040 sec			
224	16-39-23	CREATE TRIGGER BOOK_SHOW_BYSTATUS BEFORE UPDATE ON PAYMENT FOR EACH ROW	0 row(s) affected	0.0073 sec			

Movie Ticket Booking System with Social Distancing
Term project report

Entries in tables payment, booking and show_seat before a payment failure.

The screenshot shows the Oracle SQL Developer interface with a query window displaying the results of the following SQL statement:

```
1 • SELECT * FROM MOVIE_SOCIAL_DISTANCE3.BOOKING;
```

The results grid shows the following data:

BOOKING_ID	No_Of_Seats	BOOKING_Status	BOOKING_Time	SHOW_ID	CUST_ID	BOOKING_Price
30000	2	Success	2020-04-15 12:00:00	50000	201	20.00
30001	3	Success	2020-04-15 12:00:00	50000	202	10.00
30002	1	Success	2020-04-15 12:00:00	50001	203	10.00
30003	2	Success	2020-04-16 12:00:00	50001	200	60.00
30004	3	Success	2020-04-16 12:00:00	50001	201	60.00
30005	4	Success	2020-04-16 12:00:00	50002	202	80.00
30006	2	Success	2020-04-16 12:00:00	50001	203	40.00
30007	1	Success	2020-04-19 12:00:00	50002	203	10.00
30008	5	Success	2020-04-20 12:00:00	50002	200	50.00
30009	3	Success	2020-04-21 12:00:00	50002	201	30.00
30010	1	Success	2020-04-21 12:00:00	50002	202	20.00
30011	2	Success	2020-04-21 12:00:00	50003	204	30.00
30012	2	Success	2020-04-24 12:00:00	50003	204	10.00
30013	1	Success	2020-04-25 12:00:00	50003	205	10.00
30014	2	Success	2020-04-26 12:00:00	50003	200	60.00
30015	3	Success	2020-04-27 12:00:00	50003	201	80.00
30016	4	Success	2020-04-28 12:00:00	50003	202	40.00
30017	2	Success	2020-04-29 12:00:00	50003	203	10.00
30018	1	Success	2020-04-30 12:00:00	50004	208	50.00
30019	1	Success	2020-05-01 12:00:00	50004	200	30.00
30020	2	Success	2020-05-02 12:00:00	50005	203	20.00
30021	2	Success	2020-05-04 12:00:00	50005	204	30.00
30022	1	Success	2020-05-04 12:00:00	50006	202	10.00
30023	2	Success	2020-05-05 12:00:00	50006	203	10.00
30024	1	Success	2020-05-06 12:00:00	50006	204	60.00

Query Completed

The screenshot shows the Oracle SQL Developer interface with a query window displaying the results of the following SQL statement:

```
1 • SELECT * FROM MOVIE_SOCIAL_DISTANCE3.PAYMENT;
```

The results grid shows the following data:

PAYMENT_ID	PAYMENT_Mode	PAYMENT_Sub_Total	PAYMENT_Status	Total_With_Tax	PAYMENT_Time	Remote_Trans_ID	BOOKING_ID
50	Debit	20.00	Success	0.00	2020-04-12 12:04:00	P4A00000	30000
51	Credit	30.00	Success	0.00	2020-04-13 12:04:00	P4A00001	30001
52	Wallet	10.00	Success	0.00	2020-04-14 12:04:00	P4A00002	30002
53	Net Banking	20.00	Success	0.00	2020-04-15 12:04:00	P4A00003	30003
54	Debit	30.00	Success	0.00	2020-04-16 12:04:00	P4A00004	30004
55	Credit	40.00	Success	0.00	2020-04-17 12:04:00	P4A00005	30005
56	Wallet	20.00	Success	0.00	2020-04-18 12:04:00	P4A00006	30006
57	Net Banking	10.00	Success	0.00	2020-04-19 12:04:00	P4A00007	30007
58	Debit	50.00	Success	0.00	2020-04-20 12:04:00	P4A00008	30008
59	Credit	30.00	Success	0.00	2020-04-21 12:04:00	P4A00009	30009
60	Net Banking	10.00	Success	0.00	2020-04-22 12:04:00	P4A00010	30010
61	Net Banking	20.00	Success	0.00	2020-04-23 12:04:00	P4A00011	30011
62	Net Banking	20.00	Success	0.00	2020-04-24 12:04:00	P4A00012	30012
63	Net Banking	10.00	Success	0.00	2020-04-25 12:04:00	P4A00013	30013
64	Net Banking	20.00	Success	0.00	2020-04-26 12:04:00	P4A00014	30014
65	Wallet	30.00	Success	0.00	2020-04-27 12:04:00	P4A00015	30015
66	Net Banking	40.00	Success	0.00	2020-04-28 12:04:00	P4A00016	30016
67	Debit	20.00	Success	0.00	2020-04-29 12:04:00	P4A00017	30017
68	Debit	10.00	Success	0.00	2020-04-30 12:04:00	P4A00018	30018
69	Debit	10.00	Success	0.00	2020-05-01 12:04:00	P4A00019	30019
70	Debit	20.00	Success	0.00	2020-05-02 12:04:00	P4A00020	30020
71	Debit	20.00	Success	0.00	2020-05-03 12:04:00	P4A00021	30021
72	Credit	10.00	Success	0.00	2020-05-04 12:04:00	P4A00022	30022
73	Wallet	20.00	Success	0.00	2020-05-05 12:04:00	P4A00023	30023
74	Wallet	10.00	Success	0.00	2020-05-06 12:04:00	P4A00024	30024

Query Completed

Movie Ticket Booking System with Social Distancing

Term project report

The screenshot shows the MySQL Workbench interface with a query results grid. The query executed is:

```
1 • SELECT * FROM MOVIE_SOCIAL_DISTANCE3.SHOW_SEAT;
```

The results grid displays the following data:

SHOW_SEAT_ID	SHOW_SEAT_Status	SEAT_ID	SHOW_ID	BOOKING_ID
70001	0	10001	50000	30000
70002	0	10002	50000	30000
70003	0	10003	50000	30000
70004	0	10004	50000	30000

The sidebar on the left shows the schema structure, and the right sidebar contains context help and execution plan information.

Entries in tables payment, booking and show_seat after a payment failure.

Movie Ticket Booking System with Social Distancing

Term project report

Local instance 3306

Administration Schemas

Table: PAYMENT

Columns:

PAYMENT_ID	int PK
PAYMENT_Mode	varchar(15)
PAYMENT_Sub_Tot al	decimal(8,2))
PAYMENT_Status	varchar(10)
Total_With_Tax	decimal(8,2))
PAYMENT_Time	timestamp
Remote_Trans_ID	varchar(21)
BOOKING_ID	int

Action Output

Time	Action	Response	Duration / Fetch Time
226 16:45:18	SELECT * FROM MOVIE_SOCIAL_DISTANCE3.SHOW_SEAT LIMIT 0, 1000	3 row(s) returned	0.00030 sec / 0.000...
227 16:45:38	SELECT * FROM MOVIE_SOCIAL_DISTANCE3.BOOKING LIMIT 0, 1000	41 row(s) returned	0.0038 sec / 0.0001...
228 16:49:02	UPDATE PAYMENT SET PAYMENT_Status = 'Failure' WHERE BOOKING_ID = 30000	1 row(s) affected Rows matched: 1 Changed: 1 Warni...	0.0035 sec
229 16:49:13	SELECT * FROM MOVIE_SOCIAL_DISTANCE3.PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00034 sec / 0.000...
230 16:49:16	SELECT * FROM MOVIE_SOCIAL_DISTANCE3.SHOW_SEAT LIMIT 0, 1000	2 row(s) returned	0.00031 sec / 0.000...
231 16:49:20	SELECT * FROM MOVIE_SOCIAL_DISTANCE3.BOOKING LIMIT 0, 1000	41 row(s) returned	0.00042 sec / 0.000...

Query Completed

Local instance 3306

Administration Schemas

Table: PAYMENT

Columns:

PAYMENT_ID	int PK
PAYMENT_Mode	varchar(15)
PAYMENT_Sub_Tot al	decimal(8,2))
PAYMENT_Status	varchar(10)
Total_With_Tax	decimal(8,2))
PAYMENT_Time	timestamp
Remote_Trans_ID	varchar(21)
BOOKING_ID	int

Action Output

Time	Action	Response	Duration / Fetch Time
226 16:45:18	SELECT * FROM MOVIE_SOCIAL_DISTANCE3.SHOW_SEAT LIMIT 0, 1000	3 row(s) returned	0.00030 sec / 0.000...
227 16:45:38	SELECT * FROM MOVIE_SOCIAL_DISTANCE3.BOOKING LIMIT 0, 1000	41 row(s) returned	0.0038 sec / 0.0001...
228 16:49:02	UPDATE PAYMENT SET PAYMENT_Status = 'Failure' WHERE BOOKING_ID = 30000	1 row(s) affected Rows matched: 1 Changed: 1 Warni...	0.0035 sec
229 16:49:13	SELECT * FROM MOVIE_SOCIAL_DISTANCE3.PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00034 sec / 0.000...
230 16:49:16	SELECT * FROM MOVIE_SOCIAL_DISTANCE3.SHOW_SEAT LIMIT 0, 1000	2 row(s) returned	0.00031 sec / 0.000...
231 16:49:20	SELECT * FROM MOVIE_SOCIAL_DISTANCE3.BOOKING LIMIT 0, 1000	41 row(s) returned	0.00042 sec / 0.000...

Query Completed

Movie Ticket Booking System with Social Distancing

Term project report

The screenshot shows the Oracle SQL Developer interface with the following details:

- Toolbar:** Local instance 3306, Home, Schemas, Query 3, Book_show_bystatus, PAYMENT, SHOW_SEAT, BOOKING.
- Schemas:** Airline, BOOK_STORE, MOVIE_SOCIAL_DISTANCE, MOVIE_SOCIAL_DISTANC..., MOVIE_SOCIAL_DISTAN... (selected).
- Table:** PAYMENT (selected in the schema tree).
- Query:** SELECT * FROM MOVIE_SOCIAL_DISTANCE3.SHOW_SEAT;
- Result Grid:** Shows data from the SHOW_SEAT table with columns: SHOW_SEAT_ID, SHOW_SEAT_Status, SEAT_ID, SHOW_ID, BOOKING_ID. Rows shown: 70003, 70004.
- Right Panel:** Context Help is disabled. It includes tabs for Result Grid, Form Editor, Field Types, Query Stats, and Execution Plan.
- Bottom Panel:** Shows the execution history for the query, including time, action, response, and duration.

upd_user

Execution of the trigger:

Movie Ticket Booking System with Social Distancing

Term project report

```

1 • DROP TRIGGER IF EXISTS upd_user;
2
3 DELIMITER $$
4 • CREATE TRIGGER upd_user BEFORE UPDATE ON customer
5 FOR EACH ROW BEGIN
6     IF (NEW.cust_pwd IS NULL OR NEW.cust_pwd = '') THEN
7         SET NEW.cust_pwd = OLD.cust_pwd;
8     ELSE
9         SET NEW.cust_pwd = NEW.cust_pwd;
10    END IF;
11    END$$
12
13 DELIMITER ;

```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Table: PAYMENT

Columns:

PAYMENT_ID	int.PK	100%	51:6		
PAYMENT_Mode	varchar(15)				
PAYMENT_Sub_Tot al	decimal[8,2]	Action Output	Time		
PAYMENT_Status	varchar(10)		Action	Response	Duration / Fetch Time
Total_With_Tax	decimal[8,2]				
PAYMENT_Time	timestamp				
Remote_Trans_ID	varchar(21)				
BOOKING_ID	int				

228 16:49:02 UPDATE PAYMENT SET PAYMENT_Status = 'Failure' WHERE BOOKING_ID = 30000 1 row(s) affected Rows matched: 1 Changed: 1 Warn... 0.0035 sec
229 16:49:13 SELECT * FROM MOVIE_SOCIAL_DISTANCE3.PAYMENT LIMIT 0, 1000 41 row(s) returned 0.00034 sec / 0.000...
230 16:49:16 SELECT * FROM MOVIE_SOCIAL_DISTANCE3.SHOW_SEAT LIMIT 0, 1000 2 row(s) returned 0.00031 sec / 0.000...
231 16:49:20 SELECT * FROM MOVIE_SOCIAL_DISTANCE3.BOOKING LIMIT 0, 1000 41 row(s) returned 0.00042 sec / 0.000...
232 16:49:22 DROP TRIGGER IF EXISTS upd_user 0 rows affected, 1 warning(s): 1360 Trigger does no... 0.00069 sec
233 16:49:22 CREATE TRIGGER upd_user BEFORE UPDATE ON customer FOR EACH ROW BE... 0 row(s) affected 0.0041 sec

Query Completed

Before update in the CUSTOMER table

Table: CUSTOMER

Columns:

CUST_ID	CUST_LastName	CUST_FirstName	CUST_Age	CUST_Username	CUST_Pwd	CUST_Email	CUST>Contact	CUST_DOJ
200	Rupert	Washington	25	rashington	cust12345	rashington@gmail.com	6999987777	2003-10-05
201	Edward	Johnson	26	ejohnson	cust12346	ejohnson@gmail.com	6999987778	2003-12-06
202	Melanie	Smythe	30	msmythe	cust12347	msmythe@gmail.com	6999987779	2003-12-07
203	Marie	Brandon	40	mbrandon	cust12348	mbrandon@gmail.com	6999987780	2003-12-08
204	Hermine	Saranda	50	hsaranda	cust12349	hsaranda@gmail.com	6999987781	2003-12-09
205	George	Smith	45	gsmith	cust12350	gsmith@gmail.com	6999987782	2003-12-10
206	Leigha	Genkazi	35	lgenkazi	cust12351	lgenkazi@gmail.com	6999987783	2003-12-11
207	Paul	Wiesenbach	22	pwiessenbach	cust12352	pwiessenbach@gmail.com	6999987784	2003-12-12

145 03:04:00 SHOW SESSION VARIABLES LIKE 'lower_case_table_names' OK 0.000 sec
146 03:04:00 SHOW COLUMNS FROM 'MOVIE_SOCIAL_DISTANCE'.show_seat' OK 0.000 sec
147 03:04:12 PREPARE stmt FROM 'INSERT INTO `MOVIE_SOCIAL_DISTANCE`.`show_seat` (' B... OK 0.000 sec
148 03:04:12 DEALLOCATE PREPARE stmt OK 0.000 sec
149 03:07:19 SHOW SESSION VARIABLES LIKE 'lower_case_table_names' OK 0.000 sec
150 03:07:19 SHOW DATABASES OK 0.000 sec
151 03:07:24 SHOW SESSION VARIABLES LIKE 'lower_case_table_names' OK 0.000 sec
152 03:07:24 SHOW COLUMNS FROM 'MOVIE_SOCIAL_DISTANCE'.payment' OK 0.000 sec
153 03:07:26 PREPARE stmt FROM 'INSERT INTO `MOVIE_SOCIAL_DISTANCE`.`payment` (' PA... OK 0.000 sec
154 03:07:26 DEALLOCATE PREPARE stmt OK 0.000 sec
155 03:12:05 SELECT * FROM MOVIE_SOCIAL_DISTANCE.CUSTOMER LIMIT 0, 1000 8 row(s) returned 0.00050 sec / 0.000...

Query Completed

After update in the CUSTOMER table

Movie Ticket Booking System with Social Distancing

Term project report

The screenshot shows the MySQL Workbench interface with the following details:

- Toolbar:** Local instance 3306, MySQL Model*, EER Diagram.
- Schemas:** Airline, BOOK_STORE, MOVIE_SOCIAL_DISTAN...
- Tables:** CUSTOMER, EMPLOYEE, MOVIE, PAYMENT, SCREEN, SEAT, SHOW_SEAT, THEATRE.
- Query Grid:** Shows a result set for the CUSTOMER table with columns: CUST_ID, CUST_LastName, CUST_FirstName, CUST_Age, CUST_Username, CUST_Pwd, CUST_Email, CUST_Contact, CUST DOJ. Data rows include: (200, Rupert Washington, 25, rwashingt..., cust12345, rwashington@gmail.com, 6999987777, 2003-12-05), (201, Edward Johnson, 26, ejohnson, cust12346, ejohnson@gmail.com, 6999987778, 2003-12-06), etc.
- Result Grid:** Shows the same data as the Query Grid.
- Action Output:** Displays the execution log for the query, showing various MySQL statements and their execution times.
- Session:** Object Info tab selected.

upd_seat_price

Execution of the trigger:

The screenshot shows the MySQL Workbench interface with the following details:

- Toolbar:** Local instance 3306.
- Schemas:** Airline, BOOK_STORE, MOVIE_SOCIAL_DISTANCE, MOVIE_SOCIAL_DISTAN...
- Tables:** BOOKING, CUSTOMER, EMPLOYEE, MOVIE, PAYMENT, SCREEN, SEAT, SHOW_SEAT, THEATRE.
- Query Grid:** Shows a result set for the PAYMENT table with columns: PAYMENT_ID, PAYMENT_Mode, PAYMENT_Sub_Tot, PAYMENT_Status, Total_With_Tax, PAYMENT_Time, Remote_Trans_ID, BOOKING_ID.
- Action Output:** Displays the execution log for the trigger creation script, showing statements like:


```

1 • DROP TRIGGER IF EXISTS upd_seat_price;
2
3 DELIMITER $$
```

```

5 • CREATE TRIGGER upd_seat_price BEFORE INSERT ON SEAT FOR EACH ROW
6 BEGIN
7
8     IF NEW.ROW_No = 'A' OR NEW.ROW_No = 'B' THEN
9         SET NEW.SEAT_price = 20;
10    ELSE
11        IF NEW.ROW_No = 'C' OR NEW.ROW_No = 'D' THEN
12            SET NEW.SEAT_price = 15;
13        ELSE
14            SET NEW.SEAT_price = 10;
15        END IF;
16    END IF;
17
18 END$$
19
20 DELIMITER ;
```
- Session:** Object Info tab selected.

Movie Ticket Booking System with Social Distancing Term project report

After insert in the SEAT table

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** MOVIE_SOCIAL_DISTANCE
- Tables:** SEAT
- Result Grid:** Shows 18 rows of data inserted into the SEAT table.
- Action Output:** Shows the execution history of the SQL query used to insert the data.

SEAT_ID	ROW_No	SEAT_No	Is_Available	SEAT_Price	SCREEN_ID
10000	A	1	1	20.00	1000
10001	A	2	1	20.00	1000
10002	A	3	1	20.00	1000
10003	A	4	1	20.00	1000
10004	A	5	1	20.00	1000
10005	A	6	1	20.00	1000
10006	A	7	1	20.00	1000
10007	A	8	1	20.00	1000
10008	A	9	1	20.00	1000
10009	A	10	1	20.00	1000
10010	B	1	1	20.00	1000
10011	B	2	1	20.00	1000
10012	B	3	1	20.00	1000
10013	B	4	1	20.00	1000
10014	B	5	1	20.00	1000
10015	B	6	1	20.00	1000
10016	B	7	1	20.00	1000
10017	B	8	1	20.00	1000
10018	B	9	1	20.00	1000

Action Output:

Action	Time	Action	Response	Duration / Fetch Time
DELETE FROM MOVIE_SOCIAL_DISTANCE.SCREEN WHERE theatre_ID = 502	100	1 row(s) affected	0.00045 sec	
SELECT * FROM MOVIE_SOCIAL_DISTANCE.SCREEN LIMIT 0, 1000	101	5 row(s) returned	0.00029 sec / 0.000...	
INSERT INTO SCREEN VALUES (1005,'ST160,502')	102	1 row(s) affected	0.00071 sec	
SELECT * FROM MOVIE_SOCIAL_DISTANCE.SCREEN LIMIT 0, 1000	103	6 row(s) returned	0.00026 sec / 0.000...	
SHOW SESSION VARIABLES LIKE 'lower_case_table_names'	104	OK	0.000 sec	
SHOW DATABASES	105	OK	0.000 sec	
SHOW SESSION VARIABLES LIKE 'lower_case_table_names'	106	OK	0.000 sec	
SHOW COLUMNS FROM 'MOVIE_SOCIAL_DISTANCE'.`seat`	107	OK	0.000 sec	
PREPARE stmt FROM 'INSERT INTO `MOVIE_SOCIAL_DISTANCE`.`seat` ('SEAT_P...	108	OK	0.000 sec	
DEALLOCATE PREPARE stmt	109	OK	0.000 sec	
SELECT * FROM MOVIE_SOCIAL_DISTANCE.SEAT LIMIT 0, 1000	110	360 row(s) returned	0.0021 sec / 0.00003...	

BLOCK_ALTERNATE_ROWS

Execution of the stored procedure:

Movie Ticket Booking System with Social Distancing

Term project report

```

1 • DROP PROCEDURE IF EXISTS BLOCK_ALTERNATE_ROWS;
2
3 DELIMITER $$ 
4
5 • CREATE PROCEDURE BLOCK_ALTERNATE_ROWS () 
6 BEGIN
7     UPDATE SEAT SET Is_Available = 0 WHERE Row_No = 'B' OR Row_No = 'D' OR Row_No = 'F';
8 END $$ 
9
10 DELIMITER ;
11

```

The screenshot shows the MySQL Workbench interface with the 'Query 17' tab active. The code above is being run to create a stored procedure named 'BLOCK_ALTERNATE_ROWS'. The procedure uses an UPDATE statement to set the 'Is_Available' field to 0 for rows with values 'B', 'D', and 'F' in the 'Row_No' column of the 'SEAT' table.

Before calling the procedure

SEAT_ID	ROW_No	SEAT_No	Is_Available	SEAT_Price	SCREEN_ID
10000	A	1	1	20.00	1000
10001	A	2	1	20.00	1000
10002	A	3	1	20.00	1000
10003	A	4	1	20.00	1000
10004	A	5	1	20.00	1000
10005	A	6	1	20.00	1000
10006	A	7	1	20.00	1000
10007	A	8	1	20.00	1000
10008	A	9	1	20.00	1000
10009	A	10	1	20.00	1000
10100	B	1	1	20.00	1000
10101	B	2	1	20.00	1000
10102	B	3	1	20.00	1000
10103	B	4	1	20.00	1000
10104	B	5	1	20.00	1000
10105	B	6	1	20.00	1000
10106	B	7	1	20.00	1000
10107	B	8	1	20.00	1000
10108	B	9	1	20.00	1000

The screenshot shows the MySQL Workbench interface with the 'Query 17' tab active. A SELECT query is run against the 'SEAT' table, which contains 18 rows. The table has columns: SEAT_ID, ROW_No, SEAT_No, Is_Available, SEAT_Price, and SCREEN_ID. All rows currently have Is_Available set to 1.

After calling the procedure

Movie Ticket Booking System with Social Distancing

Term project report

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Local instance 3306, MySQL Model*, EER Diagram.
- Tables:** MOVIE_SOCIAL_DISTAN... (selected), BOOKING, CUSTOMER, EMPLOYEE, MOVIE, MOVIE_SHOW, PAYMENT, SCREEN, SEAT, SHOW_SEAT, THEATRE.
- Query Grid:** Contains the following SQL code:


```

1 • CALL BLOCK_ALTERNATE_ROWS();
2 • SELECT * FROM MOVIE_SOCIAL_DISTANCE.SEAT;
            
```
- Result Grid:** Shows the results of the second query, listing seats from SEAT_ID 10000 to 10118 across rows A and B, with various availability status (e.g., 1, 0).
- Action Output:** Displays the execution log for the stored procedure creation and execution, including time, action, response, and duration.
- Session:** Object Info tab selected, showing the table structure for SEAT.

createEmployee

Execution of the Stored Procedure:

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Local instance 3306, MySQL Model*, EER Diagram.
- Tables:** BOOK_SHOW_BYSTATUS, PAYMENT, SEAT, SHOW_SEAT, THEATRE.
- Query Grid:** Contains the following SQL code:


```

1 • DROP PROCEDURE IF EXISTS createEmployee;
2
3 • DELIMITER $$

4
5 • CREATE PROCEDURE createEmployee(
6     e_EMP_LastName VARCHAR(100),
7     e_EMP_FirstName VARCHAR(100),
8     e_EMP_Age INTEGER,
9     e_EMP_Designation VARCHAR(50),
10    e_EMP_Email VARCHAR(100),
11    e_EMP_Contact VARCHAR(15),
12    e_Theatre_ID INTEGER)
13 BEGIN
14
15    INSERT INTO EMPLOYEE(EMP_LastName,EMP_FirstName,EMP_Age,EMP_Designation,EMP_Email,EMP_Contact,Theatre_ID)
16    VALUES (e_EMP_LastName,e_EMP_FirstName,e_EMP_Age,e_EMP_Designation,e_EMP_Email,e_EMP_Contact,e_Theatre_ID);
17
18 END$$
19
20 DELIMITER ;
21
            
```
- Action Output:** Displays the execution log for the stored procedure creation, including time, action, response, and duration.
- Session:** Object Info tab selected, showing the table structure for PAYMENT.

Before calling the procedure

Movie Ticket Booking System with Social Distancing
Term project report

```
1 • | SELECT * FROM movie_social_distance_1.employee;
```

Result Grid | Filter Rows: Edit: Export/Import: Wrap Cell Content:

	EMP_ID	EMP_LastName	EMP_FirstName	EMP_Designation	EMP_Email	EMP_Contact	Theatre_ID
2	Rhonda	Lewis	Chairman	Irhonda@tth.com	8221182212	501	
3	Rhett	Vandam	Manager	vrhett@tth.com	8221182213	500	
4	Anne	Jones	Manager	janne@tth.com	8221182214	501	
5	John	Lange	Associate Manager	ljohn@tth.com	8221182215	501	
*	NUL	NUL	NUL	NUL	NUL	NUL	NUL

After calling the procedure

```
Query 1      SQL File 6*   employee  x  createEmployee - Routine
 1 • | call createEmployee(
 2   |   'John','Snow','Designer','johnsnow@tth.com','12345654189',500
 3   | );
 4
 5 • | SELECT * FROM movie_social_distance_1.employee;|
```

Result Grid | Filter Rows: Edit: Export/Import: Wrap Cell Content:

	EMP_ID	EMP_LastName	EMP_FirstName	EMP_Designation	EMP_Email	EMP_Contact	Theatre_ID
3	Rhett	Vandam	Manager	vrhett@tth.com	8221182213	500	
4	Anne	Jones	Manager	janne@tth.com	8221182214	501	
5	John	Lange	Associate Manager	ljohn@tth.com	8221182215	501	
6	John	Snow	Designer	johnsnow@tth.com	12345654189	500	
*	NUL	NUL	NUL	NUL	NUL	NUL	NUL

Movie Ticket Booking System with Social Distancing

Term project report

createCustomer

```

1 • DROP PROCEDURE IF EXISTS createCustomer;
2
3 DELIMITER $$

5 • CREATE PROCEDURE createCustomer(
6     IN c_CUST_LastName VARCHAR(50) ,
7     IN c_CUST_FirstName VARCHAR(50) ,
8     IN c_CUST_Age INTEGER ,
9     IN c_CUST_Username VARCHAR(50),
10    IN c_CUST_Pwd VARCHAR(15),
11    IN c_CUST_Email VARCHAR(50),
12    IN c_CUST_Contact VARCHAR(10)
13 )

14
15 BEGIN
16
17     INSERT INTO CUSTOMER(CUST_LastName,CUST_FirstName,CUST_Age,CUST_Username,CUST_Pwd,CUST_Email,CUST_Contact,
18     CUST_DOB) VALUES(c_CUST_LastName,c_CUST_FirstName,c_CUST_Age,c_CUST_Username,c_CUST_Pwd,c_CUST_Email,
19     c_CUST_Contact,DATE_FORMAT(NOW(),'%Y-%m-%d'));
20
21 END$$
22
23
DELMITER ;
24

```

Object Info **Session**

Table: PAYMENT

Columns:

PAYMENT_ID	int PK
PAYMENT_Mode	varchar(15)
PAYMENT_Sub_Tot al	decimal(8,2)
PAYMENT_Status	varchar(10)
Total_With_Tax	decimal(8,2)
PAYMENT_Time	timestamp
Remote_Trans_ID	varchar(21)
BOOKING_ID	int
PAYMENT_ID	int PK
PAYMENT_Mode	varchar(15)
PAYMENT_Sub_Tot al	decimal(8,2)
PAYMENT_Status	varchar(10)
Total_With_Tax	decimal(8,2)
PAYMENT_Time	timestamp
Remote_Trans_ID	varchar(21)
BOOKING_ID	int

Action Output

Time Action Response Duration / Fetch Time

241 17:32:33 SELECT * FROM MOVIE_SOCIAL_DISTANCE3.EMPLOYEE LIMIT 0, 1000 5 row(s) returned 0.0027 sec / 0.00000...

242 17:34:04 DROP PROCEDURE IF EXISTS createEmployee 0 row(s) affected, 1 warning(s): 1305 PROCEDURE m... 0.00046 sec

243 17:34:04 CREATE PROCEDURE createEmployee(e_EMP_LastName VARCHAR(100), e_EMP_... 0 row(s) affected 0.0052 sec

244 19:07:06 SELECT * FROM MOVIE_SOCIAL_DISTANCE3.CUSTOMER LIMIT 0, 1000 8 row(s) returned 0.0038 sec / 0.00001...

245 19:08:06 DROP PROCEDURE IF EXISTS createCustomer 0 row(s) affected, 1 warning(s): 1305 PROCEDURE m... 0.00076 sec

246 19:08:06 CREATE PROCEDURE createCustomer(IN c_CUST_LastName VARCHAR(50), IN... 0 row(s) affected 0.0026 sec

Before calling the procedure

Query 1 **customer**

1 • `SELECT * FROM movie_social_distance_1.customer;`

Result Grid

CUST_ID	CUST_LastName	CUST_FirstName	CUST_Age	CUST_Username	CUST_Pwd	CUST_Email	CUST_Contact	CUST DOJ
204	Hermine	Saranda	50	hsaranda	cust12349	hsaranda@gmail.com	6999987781	2012-09-03
205	George	Smith	45	gsmith	cust12350	gsmith@gmail.com	6999987782	2012-10-03
206	Leighla	Genkazi	35	lgenkazi	cust12351	lgenkazi@gmail.com	6999987783	2012-11-03
207	Paul	Wiesenbach	22	pwiesenbach	cust12352	pwiesenbach@gmail.com	6999987784	2012-12-03
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Movie Ticket Booking System with Social Distancing

Term project report

After calling the procedure

Query 1 customer x createCustomer - Routine createEmployee - Routine

```

1 • call createCustomer('John', 'snow', 23,
2   'Jsnow', 'cust567', 'jsnow@gh.com', '6783569878');
3
4 • SELECT * FROM movie_social_distance_1.customer;
5

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	CUST_ID	CUST_LastName	CUST_FirstName	CUST_Age	CUST_Username	CUST_Pwd	CUST_Email	CUST_Contact	CUST DOJ
205	George	Smith	45	gsmith	cust12350	gsmith@gmail.com	6999987782	2012-10-03	
206	Leighla	Genkazi	35	lgenkazi	cust12351	lgenkazi@gmail.com	6999987783	2012-11-03	
207	Paul	Wiesenbach	22	pwiesenbach	cust12352	pwiesenbach@gmail.com	6999987784	2012-12-03	
208	John	snow	23	Jsnow	cust567	jsnow@gh.com	6783569878	2020-12-04	
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

createMovie

Execution of the Stored Procedure:

Local Instance 3306

Administration Schemas Book_show_bystatus Update_seat_price Block_alternate_rows Employee Insertion Customer Insertion Movie Insertion MOVIE Context Help Snippets

SCHEMAS

- Airline
- BOOK_STORE
- MOVIE_SOCIAL_DISTANCE
- MOVIE_SOCIAL_DISTANC...
- MOVIE_SOCIAL_DISTAN...
- Tables
- BOOKING
- CUSTOMER
- EMPLOYEE
- MOVIE
- MOVIE_SHOW
- PAYMENT
- SCREEN
- SEAT
- SHOW_SEAT
- THEATRE
- Views
- Stored Procedures
- Functions
- SaleCo_Products
- SolarSystem
- SPORTS

Object Info Session

Table: PAYMENT

Columns:

PAYMENT_ID	int PK	Action Output	Time	Action	Response	Duration / Fetch Time
PAYMENT_Mode	varchar(15)					
PAYMENT_Sub_Tot al	decimal(8,2)					
PAYMENT_Status	varchar(10)					
Total_With_Tax	decimal(8,2)					
PAYMENT_Time	timestamp					
Remote_Trans_ID	varchar(21)					
BOOKING_ID	int					

100% 1:20

Time Action Response Duration / Fetch Time

244 19:07:06 SELECT * FROM MOVIE_SOCIAL_DISTANCE3.CUSTOMER LIMIT 0, 1000 8 rows(s) returned 0.0038 sec / 0.00001...

245 19:08:06 DROP PROCEDURE IF EXISTS createCustomer 0 row(s) affected, 1 warning(s): 1305 PROCEDURE m... 0.00076 sec

246 19:08:06 CREATE PROCEDURE createCustomer(IN c_CUST_LastName VARCHAR(50), IN c_CUST_Fir... 0 row(s) affected 0.0026 sec

247 19:12:06 SELECT * FROM MOVIE_SOCIAL_DISTANCE3.MOVIE LIMIT 0, 1000 5 row(s) returned 0.0055 sec / 0.00001...

248 19:12:31 DROP PROCEDURE IF EXISTS createMovie 0 row(s) affected, 1 warning(s): 1305 PROCEDURE m... 0.0011 sec

249 19:12:31 CREATE PROCEDURE createMovie(IN m_MOVIE_Title VARCHAR(50) , IN m_MO... 0 row(s) affected 0.0019 sec

Movie Ticket Booking System with Social Distancing Term project report

Before calling the procedure

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

MOVIE_ID	MOVIE_Title	MOVIE_Desc	MOVIE_Genre	MOVIE_Language	MOVIE_Duration	MOVIE_Release	MOVIE_Rating
7000	Independence Day	Independence Day is a 1996 American sci-fi... Sci-fi	English	02:33:00	1996-05-14	4.00	
7001	Iron Man	Iron Man is a 2008 American superhero film bas... Action	English	02:06:00	2008-04-01	3.50	
7002	Dangal	Dangal is a 2016 Indian Hindi language biograph... Drama	Hindi	02:49:00	2016-12-23	2.50	
7003	Baahubali	Baahubali is a 2015 Indian epic action film direct... Action	Telugu	02:39:00	2015-07-10	5.00	
7004	The Jungle Book	The Jungle Book is a 2016 American fantasy ad... Fiction	English	01:46:00	2016-04-16	2.00	
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

After calling the procedure

Query 1 movie x createMovie - Routine

call createMovie(
 'Lost Man',null,null,null,'2:00:00','2015-1-1',null
);
 5 • SELECT * FROM movie_social_distance_1.movie;

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

MOVIE_ID	MOVIE_Title	MOVIE_Desc	MOVIE_Genre	MOVIE_Language	MOVIE_Duration	MOVIE_Release	MOVIE_Rating
7000	Independence Day	Independence Day is a 1996 American sci-fi... Sci-fi	English	02:33:00	1996-05-14	4.00	
7001	Iron Man	Iron Man is a 2008 American superhero film bas... Action	English	02:06:00	2008-04-01	3.50	
7002	Dangal	Dangal is a 2016 Indian Hindi language biograph... Drama	Hindi	02:49:00	2016-12-23	2.50	
7003	Baahubali	Baahubali is a 2015 Indian epic action film direct... Action	Telugu	02:39:00	2015-07-10	5.00	
7004	The Jungle Book	The Jungle Book is a 2016 American fantasy ad... Fiction	English	01:46:00	2016-04-16	2.00	
7005	Lost Man	NULL	NULL	NULL	02:00:00	2015-01-01	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Movie Ticket Booking System with Social Distancing

Term project report

createMovieShow

```

1 • DROP PROCEDURE IF EXISTS createMovieShow;
2
3   DELIMITER $$;
4
5   CREATE PROCEDURE createMovieShow(
6     IN s_SHOW_Date DATE,
7     IN s_SHOW_Starttime TIME,
8     IN s_SHOW_Endtime TIME,
9     IN Is_Sanitized BOOL,
10    IN SCREEN_ID INTEGER,
11    IN MOVIE_ID INTEGER
12  );
13
14  BEGIN
15    INSERT INTO MOVIE_SHOW(SHOW_Date, SHOW_Starttime, SHOW_Endtime, Is_Sanitized, SCREEN_ID, MOVIE_ID)
16    VALUES(s_SHOW_Date, s_SHOW_Starttime, s_SHOW_Endtime, Is_Sanitized, SCREEN_ID, MOVIE_ID);
17
18  END
19

```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Before calling the procedure

Query 1 new_procedure - Routine movie_show ×

1 • SELECT * FROM movie_social_distance_1.movie_show;

	SHOW_ID	SHOW_Date	SHOW_Starttime	SHOW_Endtime	Is_Sanitized	SCREEN_ID	MOVIE_ID
	50006	2020-12-02	15:00:00	18:00:00	1	1003	7002
	50007	2020-12-02	19:00:00	22:00:00	1	1003	7004
	50008	2020-12-02	15:00:00	18:00:00	1	1004	7004
*	50009	2020-12-02	19:00:00	22:00:00	1	1004	7003
		NUL	NUL	NUL	NUL	NUL	NUL

After calling the procedure

The screenshot shows a MySQL Workbench interface. The top window is titled "Query 1" and contains the following SQL code:

```
1 • call createMovieShow(
2     '2020-11-27','11:00:00','12:30:00',1,1001,7001
3 );
4
5 • SELECT * FROM movie_social_distance_1.movie_show;
```

Below this is a "Result Grid" window displaying the results of the last query:

	SHOW_ID	SHOW_Date	SHOW_Starttime	SHOW_Endtime	Is_Sanitized	SCREEN_ID	MOVIE_ID
	50005	2020-12-02	18:00:00	21:00:00	1	1002	7003
	50006	2020-12-02	15:00:00	18:00:00	1	1003	7002
	50007	2020-12-02	19:00:00	22:00:00	1	1003	7004
	50008	2020-12-02	15:00:00	18:00:00	1	1004	7004
	50009	2020-12-02	19:00:00	22:00:00	1	1004	7003
	50010	2020-11-27	11:00:00	12:30:00	1	1001	7001
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

THEATRE_REMOVAL

Movie Ticket Booking System with Social Distancing

Term project report

The screenshot shows the MySQL Workbench interface with the 'Schemas' tree on the left. The 'PAYMENT' schema is selected. A query editor window displays the creation of a stored procedure 'THEATRE_REMOVAL'. The procedure takes an integer parameter 'theatre_no' and performs a series of DELETE operations to remove associated data from tables like MOVIE_SHOW, SHOW_SEAT, and PAYMENT.

```

1 • DROP PROCEDURE IF EXISTS THEATRE_REMOVAL;
2
3 DELIMITER $$

5 • CREATE PROCEDURE THEATRE_REMOVAL (IN theatre_no INTEGER)
6
7 BEGIN
8     UPDATE MOVIE_SHOW SET MOVIE_ID = NULL WHERE SCREEN_ID IN
9         (SELECT SCREEN_ID FROM SCREEN WHERE THEATRE_ID=theatre_no);
10
11    DELETE FROM SHOW_SEAT WHERE SHOW_ID IN
12        (SELECT SHOW_ID FROM MOVIE_SHOW WHERE SCREEN_ID IN
13            (SELECT SCREEN_ID FROM SCREEN WHERE THEATRE_ID = theatre_no));
14
15    DELETE FROM PAYMENT WHERE BOOKING_ID IN
16        (SELECT BOOKING_ID FROM BOOKING WHERE SHOW_ID IN
17            (SELECT SHOW_ID FROM MOVIE_SHOW WHERE SCREEN_ID IN
18                (SELECT SCREEN_ID FROM SCREEN WHERE THEATRE_ID = theatre_no)));
19
20    DELETE FROM BOOKING WHERE SHOW_ID IN
21        (SELECT SHOW_ID FROM MOVIE_SHOW WHERE SCREEN_ID IN
22            (SELECT SCREEN_ID FROM SCREEN WHERE THEATRE_ID = theatre_no));
23
24    DELETE FROM MOVIE_SHOW WHERE SCREEN_ID IN
25        (SELECT SCREEN_ID FROM SCREEN WHERE THEATRE_ID= theatre_no);
26
27    DELETE FROM SEAT WHERE SCREEN_ID IN
28        (SELECT SCREEN_ID FROM SCREEN WHERE THEATRE_ID= theatre_no);
29

```

The 'Object Info' tab at the bottom shows the table structure for 'PAYMENT' and the columns defined. The 'Session' tab shows the current connection details.

Before calling the procedure

The screenshot shows the MySQL Workbench interface with the 'Schemas' tree on the left. The 'CUSTOMER' table is selected. A query editor window displays a series of SELECT statements against various tables including PAYMENT, SHOW_SEAT, BOOKING, MOVIE_SHOW, SEAT, SCREEN, EMPLOYEE, and THEATRE. The results grid shows data for multiple rows, including Theatre IDs, names, addresses, and pincodes.

Theatre_ID	Theatre_Name	Theatre.Contact	Theatre.Address	Theatre.Pincode
500	PVR	7144171441	Street no 152, Delhi	400025
501	Inox	7144171442	Street no 281, Chennai	620015
502	ABC	7144171443	No 111 xxxx street	90128

The 'Object Info' tab at the bottom shows the table structure for 'CUSTOMER' and the columns defined. The 'Session' tab shows the current connection details.

Movie Ticket Booking System with Social Distancing

Term project report

Local Instance 3306 MySQL Model* EER Diagram

Administration Schemas Query 24 SHOW_SEAT Booking_seat Make_Payment project_table_creation Theatre_removal Context Help Snippets

Limit to 1000 rows

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

SCHEMAS

MOVIE_SOCIAL_DISTAN... (selected)

Tables: BOOKING, CUSTOMER, EMPLOYEE, MOVIE, MOVIE_SHOW, PAYMENT, SCREEN, SEAT, SHOW_SEAT, THEATRE

Views

Stored Procedures

Functions

MOVIE_SOCIAL_DISTANC...

MOVIE_SOCIAL_DISTANC...

SaleCo_Products

SolarSystem

SPORTS

Object Info Session

Table: CUSTOMER

Action Output

EMP_ID	EMP_LastName	EMP_FirstName	EMP_Age	EMP.Designation	EMP_Email	EMP.Contact	Theatre_ID
1	George	Kohnycz	50	Chairman	lgeorge@tht.com	8221182211	500
2	Rhonda	Lewis	52	Chairman	lrhonda@tht.com	8221182212	501
3	Rhett	Vandam	45	Manager	vrhett@tht.com	8221182213	500
4	Anne	Jones	42	Manager	janne@tht.com	8221182214	501
5	John	Lange	35	Associate Manager	ljohn@tht.com	8221182215	501
6	Drooge	Melanie	29	Associate Manager	mdrooge@tht.com	8221182216	502

BOOKING 26 PAYMENT 27 SHOW_SEAT 28 MOVIE_SHOW 29 SEAT 30 SCREEN 31 EMPLOYEE 32 > Apply

Action Output

Time	Action	Response	Duration / Fetch Time
206 03:51:10	SELECT * FROM SEAT LIMIT 1 0, 1000	360 row(s) returned	UUUU48 sec / UUUU...
207 03:51:10	SELECT * FROM SCREEN LIMIT 0, 1000	6 row(s) returned	0.00022 sec / 0.0000...
208 03:51:10	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.0020 sec / 0.0000...
209 03:51:16	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00032 sec / 0.0000...
210 03:51:16	SELECT * FROM PAYMENT LIMIT 0, 1000	42 row(s) returned	0.00035 sec / 0.0000...
211 03:51:16	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	45 row(s) returned	0.00028 sec / 0.0000...
212 03:51:16	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	11 row(s) returned	0.00035 sec / 0.0000...
213 03:51:16	SELECT * FROM SEAT LIMIT 0, 1000	360 row(s) returned	0.00040 sec / 0.0000...
214 03:51:16	SELECT * FROM SCREEN LIMIT 0, 1000	6 row(s) returned	0.00031 sec / 0.0000...
215 03:51:16	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.00032 sec / 0.0000...
216 03:51:16	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.00029 sec / 0.0000...

Query Completed

Local Instance 3306 MySQL Model* EER Diagram

Administration Schemas Query 24 SHOW_SEAT Booking_seat Make_Payment project_table_creation Theatre_removal Context Help Snippets

Limit to 1000 rows

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

SCHEMAS

MOVIE_SOCIAL_DISTAN... (selected)

Tables: BOOKING, CUSTOMER, EMPLOYEE, MOVIE, MOVIE_SHOW, PAYMENT, SCREEN, SEAT, SHOW_SEAT, THEATRE

Views

Stored Procedures

Functions

MOVIE_SOCIAL_DISTANC...

MOVIE_SOCIAL_DISTANC...

SaleCo_Products

SolarSystem

SPORTS

Object Info Session

Table: CUSTOMER

Action Output

SCREEN_ID	SCREEN_Name	Total_Seats	Theatre_ID
1000	S1	60	500
1001	S2	60	500
1002	S3	60	500
1003	S1	60	500
1004	S2	60	501
1005	S1	60	502

BOOKING 26 PAYMENT 27 SHOW_SEAT 28 MOVIE_SHOW 29 SEAT 30 SCREEN 31 EMPLOYEE 32 > Apply

Action Output

Time	Action	Response	Duration / Fetch Time
206 03:51:10	SELECT * FROM SEAT LIMIT 1 0, 1000	360 row(s) returned	UUUU48 sec / UUUU...
207 03:51:10	SELECT * FROM SCREEN LIMIT 0, 1000	6 row(s) returned	0.00022 sec / 0.0000...
208 03:51:10	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.0020 sec / 0.0000...
209 03:51:16	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00032 sec / 0.0000...
210 03:51:16	SELECT * FROM PAYMENT LIMIT 0, 1000	42 row(s) returned	0.00035 sec / 0.0000...
211 03:51:16	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	45 row(s) returned	0.00028 sec / 0.0000...
212 03:51:16	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	11 row(s) returned	0.00035 sec / 0.0000...
213 03:51:16	SELECT * FROM SEAT LIMIT 0, 1000	360 row(s) returned	0.00040 sec / 0.0000...
214 03:51:16	SELECT * FROM SCREEN LIMIT 0, 1000	6 row(s) returned	0.00031 sec / 0.0000...
215 03:51:16	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.00032 sec / 0.0000...
216 03:51:16	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.00029 sec / 0.0000...

Query Completed

Movie Ticket Booking System with Social Distancing

Term project report

Schemas

```

2 •  SELECT * FROM PAYMENT;
3 •  SELECT * FROM SHOW_SEAT;
4 •  SELECT * FROM MOVIE_SHOW;
5 •  SELECT * FROM SEAT;
6 •  SELECT * FROM SCREEN;
7 •  SELECT * FROM EMPLOYEE;
8 •  SELECT * FROM THEATRE;
9

```

Result Grid

SEAT_ID	ROW_No	SEAT_No	Is_Available	SEAT_Price	SCREEN_ID
10340	E	7	1	10.00	1000
10341	E	8	1	10.00	1005
10348	E	9	1	10.00	1005
10349	E	10	1	10.00	1005
10350	F	1	0	10.00	1005
10351	F	2	0	10.00	1005
10352	F	3	0	10.00	1005
10353	F	4	0	10.00	1005
10354	F	5	0	10.00	1005
10355	F	6	0	10.00	1005
10356	F	7	0	10.00	1005
10357	F	8	0	10.00	1005
10358	F	9	0	10.00	1005
10359	F	10	0	10.00	1005

Action Output

Time	Action	Response	Duration / Fetch Time
2020-03-11T00:51:10	SELECT * FROM SEAT LIMIT 0, 1000	300 row(s) returned	0.000048 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SCREEN LIMIT 0, 1000	6 row(s) returned	0.00022 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.00020 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00032 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	42 row(s) returned	0.00035 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	45 row(s) returned	0.00028 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SEAT LIMIT 0, 1000	11 row(s) returned	0.00035 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SCREEN LIMIT 0, 1000	360 row(s) returned	0.00040 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SEAT LIMIT 0, 1000	6 row(s) returned	0.00031 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.00032 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.00029 sec / 0.000...

Object Info

Session

Table: CUSTOMER

Columns:

- CUST_ID int AI PK
- CUST_LastName varchar(100)
- CUST_FirstName varchar(100)
- CUST_Age int
- CUST_Username varchar(50)
- CUST_Pwd varchar(50)
- CUST_Email varchar(100)
- CUST_Contact varchar(15)
- CUST DOJ date

Action Output

Time	Action	Response	Duration / Fetch Time
2020-03-11T00:51:10	SELECT * FROM SEAT LIMIT 0, 1000	300 row(s) returned	0.000048 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SCREEN LIMIT 0, 1000	6 row(s) returned	0.00022 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.00020 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00032 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	42 row(s) returned	0.00035 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	45 row(s) returned	0.00028 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SEAT LIMIT 0, 1000	11 row(s) returned	0.00035 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SCREEN LIMIT 0, 1000	360 row(s) returned	0.00040 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SEAT LIMIT 0, 1000	6 row(s) returned	0.00031 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.00032 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.00029 sec / 0.000...

Query Completed

Schemas

```

2 •  SELECT * FROM PAYMENT;
3 •  SELECT * FROM SHOW_SEAT;
4 •  SELECT * FROM MOVIE_SHOW;
5 •  SELECT * FROM SEAT;
6 •  SELECT * FROM SCREEN;
7 •  SELECT * FROM EMPLOYEE;
8 •  SELECT * FROM THEATRE;
9

```

Result Grid

SHOW_ID	SHOW_Date	SHOW_Starttime	SHOW_Endtime	Is_Sanitized	SCREEN_ID	MOVIE_ID
50000	2020-12-09	09:00:00	12:00:00	1	1000	7000
50001	2020-12-09	18:00:00	21:00:00	1	1000	7000
50002	2020-12-09	19:00:00	12:00:00	1	1001	7001
50003	2020-12-05	09:00:00	21:00:00	1	1001	7002
50004	2020-12-06	09:00:00	12:00:00	1	1002	7004
50005	2020-12-07	18:00:00	21:00:00	1	1002	7003
50006	2020-12-08	15:00:00	18:00:00	1	1003	7002
50007	2020-12-09	19:00:00	22:00:00	1	1003	7004
50008	2020-12-10	15:00:00	18:00:00	1	1004	7004
50009	2020-12-11	19:00:00	22:00:00	1	1004	7003
50010	2020-12-12	15:00:00	18:00:00	1	1005	7004

Action Output

Time	Action	Response	Duration / Fetch Time
2020-03-11T00:51:10	SELECT * FROM SEAT LIMIT 0, 1000	360 row(s) returned	0.000048 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SCREEN LIMIT 0, 1000	6 row(s) returned	0.00022 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.00020 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00032 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	42 row(s) returned	0.00035 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	45 row(s) returned	0.00028 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SEAT LIMIT 0, 1000	11 row(s) returned	0.00035 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SCREEN LIMIT 0, 1000	360 row(s) returned	0.00040 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SEAT LIMIT 0, 1000	6 row(s) returned	0.00031 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.00032 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.00029 sec / 0.000...

Object Info

Session

Table: CUSTOMER

Columns:

- CUST_ID int AI PK
- CUST_LastName varchar(100)
- CUST_FirstName varchar(100)
- CUST_Age int
- CUST_Username varchar(50)
- CUST_Pwd varchar(50)
- CUST_Email varchar(100)
- CUST_Contact varchar(15)
- CUST DOJ date

Action Output

Time	Action	Response	Duration / Fetch Time
2020-03-11T00:51:10	SELECT * FROM SEAT LIMIT 0, 1000	360 row(s) returned	0.000048 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SCREEN LIMIT 0, 1000	6 row(s) returned	0.00022 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.00020 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00032 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	42 row(s) returned	0.00035 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	45 row(s) returned	0.00028 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SEAT LIMIT 0, 1000	11 row(s) returned	0.00035 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SCREEN LIMIT 0, 1000	360 row(s) returned	0.00040 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM SEAT LIMIT 0, 1000	6 row(s) returned	0.00031 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.00032 sec / 0.000...
2020-03-11T00:51:10	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.00029 sec / 0.000...

Query Completed

Movie Ticket Booking System with Social Distancing

Term project report

Local instance 3306 MySQL Model* EER Diagram

Schemas

- MOVIE_SOCIAL_DISTANCING
 - Tables
 - CUSTOMER

Query 24

```

2 •  SELECT * FROM PAYMENT;
3 •  SELECT * FROM SHOW_SEAT;
4 •  SELECT * FROM MOVIE_SHOW;
5 •  SELECT * FROM SEAT;
6 •  SELECT * FROM SCREEN;
7 •  SELECT * FROM EMPLOYEE;
8 •  SELECT * FROM THEATRE;
9

```

Limit to 1000 rows

Result Grid

SHOW_SEAT_ID	SHOW_SEAT_Status	SEAT_ID	SHOW_ID	BOOKING_ID
70031	U	70031	50001	30043
70032	0	10032	50003	30032
70033	0	10033	50004	30033
70034	0	10034	50004	30034
70035	0	10035	50005	30035
70036	0	10036	50006	30036
70037	0	10037	50006	30037
70038	0	10038	50004	30038
70039	0	10039	50004	30039
70040	0	10040	50004	30040
70041	0	10300	50010	30043
70042	0	10301	50010	30043
70043	0	10302	50010	30043
70044	0	10303	50010	30043

Action Output

Time	Action	Response	Duration / Fetch Time
202 03:51:10	SELECT * FROM SEAT LIMIT 0, 1000	360 row(s) returned	0.000048 sec / 0.0000...
207 03:51:10	SELECT * FROM SCREEN LIMIT 0, 1000	6 row(s) returned	0.000022 sec / 0.0000...
208 03:51:10	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.000020 sec / 0.0000...
209 03:51:10	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.000032 sec / 0.0000...
210 03:51:16	SELECT * FROM PAYMENT LIMIT 0, 1000	42 row(s) returned	0.000035 sec / 0.0000...
211 03:51:16	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	45 row(s) returned	0.000028 sec / 0.0000...
212 03:51:16	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	11 row(s) returned	0.000035 sec / 0.0000...
213 03:51:16	SELECT * FROM SEAT LIMIT 0, 1000	360 row(s) returned	0.000040 sec / 0.0000...
214 03:51:16	SELECT * FROM SCREEN LIMIT 0, 1000	6 row(s) returned	0.000031 sec / 0.0000...
215 03:51:16	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.000032 sec / 0.0000...
216 03:51:16	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.000029 sec / 0.0000...

Query Completed

Local instance 3306 MySQL Model* EER Diagram

Schemas

- MOVIE_SOCIAL_DISTANCING
 - Tables
 - CUSTOMER

Query 24

```

1 •  SELECT * FROM BOOKING;
2 •  SELECT * FROM PAYMENT;
3 •  SELECT * FROM SHOW_SEAT;
4
5

```

Limit to 1000 rows

Result Grid

PAYMENT_ID	PAYMENT_Mode	PAYMENT_Sub_Total	PAYMENT_Status	Total_With_Tax	PAYMENT_Time	Remote_Trans_ID	BOOKING_ID
73	Wallet	20.00	Success	20.00	2020-05-05 12:04:00	P4A00023	30023
74	Wallet	10.00	Success	10.00	2020-05-05 12:04:00	P4A00024	30024
75	Wallet	20.00	Success	20.00	2020-05-05 12:04:00	P4A00025	30025
76	Wallet	20.00	Success	20.00	2020-05-05 12:04:00	P4A00026	30026
77	Credit	10.00	Success	10.00	2020-05-05 12:04:00	P4A00027	30027
78	Wallet	20.00	Success	20.00	2020-05-10 12:04:00	P4A00028	30028
79	Net Banking	30.00	Success	30.00	2020-05-11 12:04:00	P4A00029	30029
80	Debit	40.00	Success	40.00	2020-05-12 12:04:00	P4A00030	30030
81	Credit	20.00	Success	20.00	2020-05-13 12:04:00	P4A00031	30031
82	Wallet	10.00	Success	10.00	2020-05-14 12:04:00	P4A00032	30032
83	Net Banking	20.00	Success	20.00	2020-05-15 12:04:00	P4A00033	30033
84	Net Banking	20.00	Success	20.00	2020-05-16 12:04:00	P4A00034	30034
85	Net Banking	10.00	Success	10.00	2020-05-17 12:04:00	P4A00035	30035
86	Net Banking	20.00	Success	20.00	2020-05-18 12:04:00	P4A00036	30036
87	Wallet	30.00	Success	30.00	2020-05-19 12:04:00	P4A00037	30037
88	Net Banking	40.00	Success	40.00	2020-05-20 12:04:00	P4A00038	30038
89	Debit	20.00	Success	20.00	2020-05-21 12:04:00	P4A00039	30039
90	Debit	10.00	Success	10.00	2020-05-22 12:04:00	P4A00040	30040

Action Output

Time	Action	Response	Duration / Fetch Time
166 03:19:24	CALL View_Booked_seats(\$0000)	1 row(s) returned	0.000041 sec / 0.0000...
167 03:19:36	SELECT * FROM MOVIE_SOCIAL_DISTANCE.SHOW_SEAT_SEAT_LIMIT 0, 1000	41 row(s) returned	0.000030 sec / 0.0000...
168 03:19:51	DROP PROCEDURE IF EXISTS View_Booked_seats	0 row(s) affected	0.0017 sec
169 03:19:51	CREATE PROCEDURE View_Booked_seats (IN shoid INTEGER) BEGIN DECLARE...	0 row(s) affected	0.0049 sec
170 03:19:55	CALL View_Booked_seats(5000)	1 row(s) returned	0.00073 sec / 0.0000...
171 03:21:03	DROP PROCEDURE IF EXISTS View_Booked_seats	0 row(s) affected	0.0013 sec
172 03:21:03	CREATE PROCEDURE View_Booked_seats (IN shoid INTEGER) BEGIN SELECT...	0 row(s) affected	0.00096 sec
173 03:30:32	SELECT * FROM MOVIE_SOCIAL_DISTANCE.THEATRE LIMIT 0, 1000	3 row(s) returned	0.0016 sec / 0.0000...
174 03:32:25	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.0000...
175 03:32:25	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00057 sec / 0.0000...
176 03:32:25	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00031 sec / 0.0000...

Query Completed

Movie Ticket Booking System with Social Distancing

Term project report

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** MOVIE_SOCIAL_DISTANCING
- Tables:** CUSTOMER, BOOKING, PAYMENT, SHOW_SEAT
- Query Editor:**

```

1 •  SELECT * FROM BOOKING;
2 •  SELECT * FROM PAYMENT;
3 •  SELECT * FROM SHOW_SEAT;
4
5
    
```
- Result Grid:** Displays booking data with columns: BOOKING_ID, No.Of_Seats, BOOKING_Status, BOOKING_Price, BOOKING_Time, SHOW_ID, CUST_ID.
- Action Output:** Shows the execution history of the procedure with 167 rows, including time, action, response, and duration.

Time	Action	Response	Duration / Fetch Time
166 - 03:19:24	LALL VIEW_BOOKED_SEATS(buuuu)	1 row(s) returned	UUUUU41 sec / UUUUU...
167 03:19:36	SELECT * FROM MOVIE_SOCIAL_DISTANCE.SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
168 03:19:51	DROP PROCEDURE IF EXISTS View_Blocked_seats	0 row(s) affected	0.0017 sec
169 03:19:51	CREATE PROCEDURE View_Blocked_seats (IN showid INTEGER) BEGIN DECLARE...	0 row(s) affected	0.0049 sec
170 03:19:55	CALL View_Blocked_seats(50000)	1 row(s) returned	0.00073 sec / 0.000...
171 03:21:03	DROP PROCEDURE IF EXISTS View_Blocked_seats	0 row(s) affected	0.0013 sec
172 03:21:03	CREATE PROCEDURE View_Blocked_seats (IN showid INTEGER) BEGIN SELECT...	0 row(s) affected	0.00096 sec
173 03:30:32	SELECT * FROM BOOKING LIMIT 0, 1000	3 row(s) returned	0.0016 sec / 0.0000...
174 03:32:25	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
175 03:32:25	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00057 sec / 0.0000...
176 03:32:25	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00031 sec / 0.0000...

After calling the procedure

Movie Ticket Booking System with Social Distancing

Term project report

Local instance 3306 MySQL Model* EER Diagram

Administration Schemas Query 24 SHOW_SEAT Booking_seat Make_Payment project_table_creation Theatre_removal Context Help Snippets

Schemas

MOVIE_SOCIAL_DISTANCING Tables

- BOOKING
- CUSTOMER
- EMPLOYEE
- MOVIE
- MOVIE_SHOW
- PAYMENT
- SCREEN
- SEAT
- SHOW_SEAT
- THEATRE

Views

Stored Procedures

Functions

Object Info Session

Table: CUSTOMER

Action Output

Time	Action	Response	Duration / Fetch Time
Z15 03:51:56	SELECT * FROM EMPLOYEE LIMIT 0, 1000	0 row(s) returned	0.00002 sec / 0.000...
Z16 03:51:56	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.00029 sec / 0.000...
Z17 04:00:30	CALL THEATRE_REMOVAL(502)	1 row(s) affected	0.018 sec
Z18 04:00:30	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00027 sec / 0.000...
Z19 04:00:30	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00033 sec / 0.000...
Z20 04:00:30	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
Z21 04:00:30	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	10 row(s) returned	0.00034 sec / 0.000...
Z22 04:00:30	SELECT * FROM SEAT LIMIT 0, 1000	300 row(s) returned	0.00037 sec / 0.000...
Z23 04:00:30	SELECT * FROM SCREEN LIMIT 0, 1000	5 row(s) returned	0.00020 sec / 0.000...
Z24 04:00:30	SELECT * FROM EMPLOYEE LIMIT 0, 1000	5 row(s) returned	0.00031 sec / 0.000...
Z25 04:00:30	SELECT * FROM THEATRE LIMIT 0, 1000	2 row(s) returned	0.00031 sec / 0.000...

Query Completed

Local instance 3306 MySQL Model* EER Diagram

Administration Schemas Query 24 SHOW_SEAT Booking_seat Make_Payment project_table_creation Theatre_removal Context Help Snippets

Schemas

MOVIE_SOCIAL_DISTANCING Tables

- BOOKING
- CUSTOMER
- EMPLOYEE
- MOVIE
- MOVIE_SHOW
- PAYMENT
- SCREEN
- SEAT
- SHOW_SEAT
- THEATRE

Views

Stored Procedures

Functions

Object Info Session

Table: CUSTOMER

Action Output

Time	Action	Response	Duration / Fetch Time
Z15 03:51:56	SELECT * FROM EMPLOYEE LIMIT 0, 1000	0 row(s) returned	0.00002 sec / 0.000...
Z16 03:51:56	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.00029 sec / 0.000...
Z17 04:00:30	CALL THEATRE_REMOVAL(502)	1 row(s) affected	0.018 sec
Z18 04:00:30	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00027 sec / 0.000...
Z19 04:00:30	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00033 sec / 0.000...
Z20 04:00:30	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
Z21 04:00:30	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	10 row(s) returned	0.00034 sec / 0.000...
Z22 04:00:30	SELECT * FROM SEAT LIMIT 0, 1000	300 row(s) returned	0.00037 sec / 0.000...
Z23 04:00:30	SELECT * FROM SCREEN LIMIT 0, 1000	5 row(s) returned	0.00020 sec / 0.000...
Z24 04:00:30	SELECT * FROM EMPLOYEE LIMIT 0, 1000	5 row(s) returned	0.00031 sec / 0.000...
Z25 04:00:30	SELECT * FROM THEATRE LIMIT 0, 1000	2 row(s) returned	0.00031 sec / 0.000...

Query Completed

Movie Ticket Booking System with Social Distancing

Term project report

Query 24

```

1 • CALL THEATRE_REMOVAL(502);
2 • SELECT * FROM BOOKING;
3 • SELECT * FROM PAYMENT;
4 • SELECT * FROM SHOW_SEAT;
5 • SELECT * FROM MOVIE_SHOW;
6 • SELECT * FROM SEAT;
7 • SELECT * FROM SCREEN;
8 • SELECT * FROM EMPLOYEE;
9 • SELECT * FROM THEATRE;

```

Result Grid

SCREEN_ID	SCREEN_Name	Total_Seats	Theatre_ID
1000	S1	60	500
1001	S2	60	500
1002	S3	60	500
1003	S1	60	501
1004	S2	60	501

Action Output

Time	Action	Response	Duration / Fetch Time
Z15 03:51:56	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.00032 sec / 0.000...
Z16 03:51:56	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.00029 sec / 0.000...
Z17 04:00:30	CALL THEATRE_REMOVAL(502)	1 row(s) affected	0.018 sec
Z18 04:00:30	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00027 sec / 0.000...
Z19 04:00:30	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00033 sec / 0.000...
Z20 04:00:30	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
Z21 04:00:30	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	10 row(s) returned	0.00034 sec / 0.000...
Z22 04:00:30	SELECT * FROM SEAT LIMIT 0, 1000	300 row(s) returned	0.00037 sec / 0.000...
Z23 04:00:30	SELECT * FROM SCREEN LIMIT 0, 1000	5 row(s) returned	0.00020 sec / 0.000...
Z24 04:00:30	SELECT * FROM EMPLOYEE LIMIT 0, 1000	5 row(s) returned	0.00031 sec / 0.000...
Z25 04:00:30	SELECT * FROM THEATRE LIMIT 0, 1000	2 row(s) returned	0.00031 sec / 0.000...

Query Completed

Query 24

```

1 • CALL THEATRE_REMOVAL(502);
2 • SELECT * FROM BOOKING;
3 • SELECT * FROM PAYMENT;
4 • SELECT * FROM SHOW_SEAT;
5 • SELECT * FROM MOVIE_SHOW;
6 • SELECT * FROM SEAT;
7 • SELECT * FROM SCREEN;
8 • SELECT * FROM EMPLOYEE;
9 • SELECT * FROM THEATRE;

```

Result Grid

SEAT_ID	ROW_No	SEAT_No	Is_Available	SEAT_Price	SCREEN_ID
10288	E	9	1	10.00	1004
10289	E	10	1	10.00	1004
10290	F	1	0	10.00	1004
10291	F	2	0	10.00	1004
10292	F	3	0	10.00	1004
10293	F	4	0	10.00	1004
10294	F	5	0	10.00	1004
10295	F	6	0	10.00	1004
10296	F	7	0	10.00	1004
10297	F	8	0	10.00	1004
10298	F	9	0	10.00	1004
10299	F	10	0	10.00	1004

Action Output

Time	Action	Response	Duration / Fetch Time
Z15 03:51:56	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.00032 sec / 0.000...
Z16 03:51:56	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.00029 sec / 0.000...
Z17 04:00:30	CALL THEATRE_REMOVAL(502)	1 row(s) affected	0.018 sec
Z18 04:00:30	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00027 sec / 0.000...
Z19 04:00:30	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00032 sec / 0.000...
Z20 04:00:30	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
Z21 04:00:30	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	10 row(s) returned	0.00034 sec / 0.000...
Z22 04:00:30	SELECT * FROM SEAT LIMIT 0, 1000	300 row(s) returned	0.00037 sec / 0.000...
Z23 04:00:30	SELECT * FROM SCREEN LIMIT 0, 1000	5 row(s) returned	0.00020 sec / 0.000...
Z24 04:00:30	SELECT * FROM EMPLOYEE LIMIT 0, 1000	5 row(s) returned	0.00031 sec / 0.000...
Z25 04:00:30	SELECT * FROM THEATRE LIMIT 0, 1000	2 row(s) returned	0.00031 sec / 0.000...

Query Completed

Movie Ticket Booking System with Social Distancing

Term project report

Query 24: SHOW_SEAT, Booking_seat, Make_Payment, project_table_creation, Theatre_removal

```

1 • CALL THEATRE_REMOVAL(502);
2 • SELECT * FROM BOOKING;
3 • SELECT * FROM PAYMENT;
4 • SELECT * FROM SHOW_SEAT;
5 • SELECT * FROM MOVIE_SHOW;
6 • SELECT * FROM SEAT;
7 • SELECT * FROM SCREEN;
8 • SELECT * FROM EMPLOYEE;
9 • SELECT * FROM THEATRE;

```

Result Grid (Limit to 1000 rows)

SHOW_ID	SHOW_Date	SHOW_Starttime	SHOW_Endtime	Is_Sanitized	SCREEN_ID	MOVIE_ID
50000	2020-12-02	08:00:00	12:00:00	1	1000	7000
50001	2020-12-03	18:00:00	21:00:00	1	1000	7000
50002	2020-12-04	09:00:00	12:00:00	1	1001	7001
50004	2020-12-06	09:00:00	12:00:00	1	1002	7004
50005	2020-12-07	18:00:00	21:00:00	1	1002	7003
50006	2020-12-08	15:00:00	18:00:00	1	1003	7002
50007	2020-12-09	19:00:00	22:00:00	1	1003	7004
50008	2020-12-10	15:00:00	18:00:00	1	1004	7004
50009	2020-12-11	19:00:00	22:00:00	1	1004	7003

Action Output

Time	Action	Response	Duration / Fetch Time
215 03:51:56	SELECT * FROM EMPLOYEE LIMIT 0, 1000	0 row(s) returned	0.00032 sec / 0.000...
216 03:51:56	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.00029 sec / 0.000...
217 04:00:30	CALL THEATRE_REMOVAL(502)	1 row(s) affected	0.018 sec
218 04:00:30	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00027 sec / 0.000...
219 04:00:30	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00033 sec / 0.000...
220 04:00:30	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
221 04:00:30	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	10 row(s) returned	0.00034 sec / 0.000...
222 04:00:30	SELECT * FROM SEAT LIMIT 0, 1000	300 row(s) returned	0.00037 sec / 0.000...
223 04:00:30	SELECT * FROM SCREEN LIMIT 0, 1000	5 row(s) returned	0.00020 sec / 0.000...
224 04:00:30	SELECT * FROM EMPLOYEE LIMIT 0, 1000	5 row(s) returned	0.00031 sec / 0.000...
225 04:00:30	SELECT * FROM THEATRE LIMIT 0, 1000	2 row(s) returned	0.00031 sec / 0.000...

Query Completed

Query 24: SHOW_SEAT, Booking_seat, Make_Payment, project_table_creation, Theatre_removal

```

1 • CALL THEATRE_REMOVAL(502);
2 • SELECT * FROM BOOKING;
3 • SELECT * FROM PAYMENT;
4 • SELECT * FROM SHOW_SEAT;
5 • SELECT * FROM MOVIE_SHOW;
6 • SELECT * FROM SEAT;
7 • SELECT * FROM SCREEN;
8 • SELECT * FROM EMPLOYEE;
9 • SELECT * FROM THEATRE;

```

Result Grid (Limit to 1000 rows)

SHOW_SEAT_ID	SHOW_SEAT_Status	SEAT_ID	SHOW_ID	BOOKING_ID
70029	0	10029	50003	30029
70030	0	10030	50003	30030
70031	0	10031	50003	30031
70032	0	10032	50003	30032
70033	0	10033	50004	30033
70034	0	10034	50004	30034
70035	0	10035	50004	30035
70036	0	10036	50004	30036
70037	0	10037	50004	30037
70038	0	10038	50004	30038
70039	0	10039	50004	30039
70040	0	10040	50004	30040

Action Output

Time	Action	Response	Duration / Fetch Time
215 03:51:56	SELECT * FROM EMPLOYEE LIMIT 0, 1000	0 row(s) returned	0.00032 sec / 0.000...
216 03:51:56	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.00029 sec / 0.000...
217 04:00:30	CALL THEATRE_REMOVAL(502)	1 row(s) affected	0.018 sec
218 04:00:30	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00027 sec / 0.000...
219 04:00:30	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00033 sec / 0.000...
220 04:00:30	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
221 04:00:30	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	10 row(s) returned	0.00034 sec / 0.000...
222 04:00:30	SELECT * FROM SEAT LIMIT 0, 1000	300 row(s) returned	0.00037 sec / 0.000...
223 04:00:30	SELECT * FROM SCREEN LIMIT 0, 1000	5 row(s) returned	0.00020 sec / 0.000...
224 04:00:30	SELECT * FROM EMPLOYEE LIMIT 0, 1000	5 row(s) returned	0.00031 sec / 0.000...
225 04:00:30	SELECT * FROM THEATRE LIMIT 0, 1000	2 row(s) returned	0.00031 sec / 0.000...

Query Completed

Movie Ticket Booking System with Social Distancing

Term project report

Query 24: SHOW_SEAT

```

1 • CALL THEATRE_REMOVAL(502);
2 • SELECT * FROM BOOKING;
3 • SELECT * FROM PAYMENT;
4 • SELECT * FROM SHOW_SEAT;
5 • SELECT * FROM MOVIE_SHOW;
6 • SELECT * FROM SEAT;
7 • SELECT * FROM SCREEN;
8 • SELECT * FROM EMPLOYEE;
9 • SELECT * FROM THEATRE;
10

```

Limit to 1000 rows

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

PAYMENT_ID	PAYMENT_Mode	PAYMENT_Sub_Total	PAYMENT_Status	Total_With_Tax	PAYMENT_Time	Remote_Trans_ID	BOOKING_ID
79	Net Banking	30.00	Success	30.00	2020-05-11 12:04:00	P4AA0029	30029
80	Debit	40.00	Success	40.00	2020-05-12 12:04:00	P4AA0030	30030
81	Credit	20.00	Success	20.00	2020-05-13 12:04:00	P4AA0031	30031
82	Wallet	10.00	Success	10.00	2020-05-14 12:04:00	P4AA0032	30032
83	Net Banking	20.00	Success	20.00	2020-05-15 12:04:00	P4AA0033	30033
84	Net Banking	20.00	Success	20.00	2020-05-16 12:04:00	P4AA0034	30034
85	Net Banking	10.00	Success	10.00	2020-05-17 12:04:00	P4AA0035	30035
86	Net Banking	20.00	Success	20.00	2020-05-18 12:04:00	P4AA0036	30036
87	Wallet	30.00	Success	30.00	2020-05-19 12:04:00	P4AA0037	30037
88	Net Banking	40.00	Success	40.00	2020-05-20 12:04:00	P4AA0038	30038
89	Debit	20.00	Success	20.00	2020-05-21 12:04:00	P4AA0039	30039
90	Debit	10.00	Success	10.00	2020-05-22 12:04:00	P4AA0040	30040

Result Grid | Filter Rows: | Search | Edit: | Export/Import: |

Action Output

Time	Action	Response	Duration / Fetch Time
2020-05-19 03:51:56	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.000029 sec / 0.000...
2020-05-19 03:51:56	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.000029 sec / 0.000...
2020-05-19 04:00:30	CALL THEATRE_REMOVAL(502)	1 row(s) affected	0.018 sec
2020-05-19 04:00:30	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00027 sec / 0.0000...
2020-05-19 04:00:30	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00033 sec / 0.000...
2020-05-19 04:00:30	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
2020-05-19 04:00:30	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	10 row(s) returned	0.00034 sec / 0.000...
2020-05-19 04:00:30	SELECT * FROM SEAT LIMIT 0, 1000	300 row(s) returned	0.00037 sec / 0.0000...
2020-05-19 04:00:30	SELECT * FROM SCREEN LIMIT 0, 1000	5 row(s) returned	0.00020 sec / 0.000...
2020-05-19 04:00:30	SELECT * FROM EMPLOYEE LIMIT 0, 1000	5 row(s) returned	0.00031 sec / 0.0000...
2020-05-19 04:00:30	SELECT * FROM THEATRE LIMIT 0, 1000	2 row(s) returned	0.00031 sec / 0.0000...

Query Completed

Query 24: BOOKING

```

1 • CALL THEATRE_REMOVAL(502);
2 • SELECT * FROM BOOKING;
3 • SELECT * FROM PAYMENT;
4 • SELECT * FROM SHOW_SEAT;
5 • SELECT * FROM MOVIE_SHOW;
6 • SELECT * FROM SEAT;
7 • SELECT * FROM SCREEN;
8 • SELECT * FROM EMPLOYEE;
9 • SELECT * FROM THEATRE;
10

```

Limit to 1000 rows

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

BOOKING_ID	No_Of_Seats	BOOKING_Status	BOOKING_Price	BOOKING_Time	SHOW_ID	CUST_ID
30029	3	Success	30.00	2020-05-11 12:00:00	50007	203
30030	4	Success	40.00	2020-05-12 12:00:00	50007	204
30031	2	Success	20.00	2020-05-13 12:00:00	50007	205
30032	1	Success	10.00	2020-05-14 12:00:00	50008	200
30033	2	Success	20.00	2020-05-15 12:00:00	50008	201
30034	2	Success	20.00	2020-05-16 12:00:00	50008	202
30035	1	Success	10.00	2020-05-17 12:00:00	50008	203
30036	2	Success	20.00	2020-05-18 12:00:00	50008	204
30037	3	Success	30.00	2020-05-19 12:00:00	50008	205
30038	4	Success	40.00	2020-05-20 12:00:00	50009	203
30039	2	Success	20.00	2020-05-21 12:00:00	50009	204
30040	1	Success	10.00	2020-05-22 12:00:00	50009	205

Result Grid | Filter Rows: | Search | Edit: | Export/Import: |

Action Output

Time	Action	Response	Duration / Fetch Time
2020-05-19 03:51:56	SELECT * FROM EMPLOYEE LIMIT 0, 1000	6 row(s) returned	0.000032 sec / 0.000...
2020-05-19 03:51:56	SELECT * FROM THEATRE LIMIT 0, 1000	3 row(s) returned	0.000029 sec / 0.000...
2020-05-19 04:00:30	CALL THEATRE_REMOVAL(502)	1 row(s) affected	0.018 sec
2020-05-19 04:00:30	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00027 sec / 0.0000...
2020-05-19 04:00:30	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00033 sec / 0.000...
2020-05-19 04:00:30	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
2020-05-19 04:00:30	SELECT * FROM MOVIE_SHOW LIMIT 0, 1000	10 row(s) returned	0.00034 sec / 0.000...
2020-05-19 04:00:30	SELECT * FROM SEAT LIMIT 0, 1000	300 row(s) returned	0.00037 sec / 0.0000...
2020-05-19 04:00:30	SELECT * FROM SCREEN LIMIT 0, 1000	5 row(s) returned	0.00020 sec / 0.000...
2020-05-19 04:00:30	SELECT * FROM EMPLOYEE LIMIT 0, 1000	5 row(s) returned	0.00031 sec / 0.0000...
2020-05-19 04:00:30	SELECT * FROM THEATRE LIMIT 0, 1000	2 row(s) returned	0.00031 sec / 0.0000...

Query Completed

Movie Ticket Booking System with Social Distancing
Term project report

View_Booked_seats

Execution of the Stored Procedure:

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Local instance 3306, MySQL Model*, EER Diagram.
- Tables:** Airline, BOOK_STORE, MOVIE_SOCIAL_DISTANC..., MOVIE, PAYMENT, SCREEN, SEAT, SHOW_SEAT, THEATRE, Views, Stored Procedures, Functions.
- Query Editor:** Query 24, titled "View_Booked_tickets". It contains the code for the stored procedure:

```

1 • DROP PROCEDURE IF EXISTS View_Booked_seats;
2
3 DELIMITER $$ 
4
5 • CREATE PROCEDURE View_Booked_seats (IN showid INTEGER)
6
7 BEGIN
8
9     SELECT movie_show.SHOW_ID, count(show_seat.SHOW_SEAT_Status) as booked_seats
10    FROM seat
11   JOIN show_seat ON seat.seat_id = show_seat.seat_id
12   JOIN movie_show ON movie_show.show_id=show_seat.show_id
13      WHERE show_seat.SHOW_ID=showid
14      GROUP BY movie_show.show_id;
15
16 END $$ 
17
18 DELIMITER ;

```

- Action Output:** Shows the history of actions taken on the database, including the creation of the procedure and its execution.

Time	Action	Response	Duration / Fetch Time
162 03:19:59	CALL View_Booked_seats(50010)	Error Code: 1054. Unknown column 'e.show_id' in 'where clause'	0.0015 sec
163 03:18:57	DROP PROCEDURE IF EXISTS View_Booked_seats	0 row(s) affected	0.0017 sec
164 03:18:57	CREATE PROCEDURE View_Booked_seats (IN showid INTEGER) BEGIN DECLARE...	0 row(s) affected	0.0017 sec
165 03:19:00	CALL View_Booked_seats(50010)	0 row(s) returned	0.0019 sec / 0.0000...
166 03:19:24	CALL View_Booked_seats(50000)	1 row(s) returned	0.00041 sec / 0.0000...
167 03:19:36	SELECT * FROM MOVIE_SOCIAL_DISTANCE.SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
168 03:19:51	DROP PROCEDURE IF EXISTS View_Booked_seats	0 row(s) affected	0.0017 sec
169 03:19:51	CREATE PROCEDURE View_Booked_seats (IN showid INTEGER) BEGIN DECLARE...	0 row(s) affected	0.0049 sec
170 03:19:55	CALL View_Booked_seats(50000)	1 row(s) returned	0.00073 sec / 0.0000...
171 03:21:03	DROP PROCEDURE IF EXISTS View_Booked_seats	0 row(s) affected	0.0013 sec
172 03:21:03	CREATE PROCEDURE View_Booked_seats (IN showid INTEGER) BEGIN SELECT...	0 row(s) affected	0.00096 sec

After calling the procedure:

Movie Ticket Booking System with Social Distancing

Term project report

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** MOVIE_SOCIAL_DISTAN... (selected)
- Tables:** CUSTOMER (selected)
- Query Grid:** Result of the query `CALL View_Booked_seats(50000);`. The result grid shows one row with value 50000.
- Action Output:** Shows the execution log with 172 entries, detailing the creation and execution of the stored procedure.
- Message Bar:** "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

MAKE_PAYMENT

Execution of the Stored Procedure:

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** PAYMENT (selected)
- Stored Procedures:** MAKE_PAYMENT (selected)
- Query Grid:** Result of the query `DROP PROCEDURE IF EXISTS MAKE_PAYMENT;`
- Action Output:** Shows the execution log with 261 entries, detailing the creation of the stored procedure.
- Message Bar:** "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

Before calling the procedure

Movie Ticket Booking System with Social Distancing

Term project report

```

1 • SELECT * FROM BOOKING;
2 • SELECT * FROM PAYMENT;
3 • SELECT * FROM SHOW_SEAT;
4
5

```

PAYMENT_ID	PAYMENT_Mode	PAYMENT_Sub_Total	PAYMENT_Status	Total_With_Tax	PAYMENT_Time	Remote_Trans_ID	BOOKING_ID
73	Wallet	20.00	Success	20.00	2020-05-05 12:04:00	P4A00023	30023
74	Wallet	10.00	Success	10.00	2020-05-06 12:04:00	P4A00024	30024
75	Wallet	20.00	Success	20.00	2020-05-07 12:04:00	P4A00025	30025
76	Wallet	20.00	Success	20.00	2020-05-08 12:04:00	P4A00026	30026
77	Credit	10.00	Success	10.00	2020-05-09 12:04:00	P4A00027	30027
78	Wallet	20.00	Success	20.00	2020-05-10 12:04:00	P4A00028	30028
79	Net Banking	30.00	Success	30.00	2020-05-11 12:04:00	P4A00029	30029
80	Debit	40.00	Success	40.00	2020-05-12 12:04:00	P4A00030	30030
81	Debit	20.00	Success	20.00	2020-05-13 12:04:00	P4A00031	30031
82	Wallet	10.00	Success	10.00	2020-05-14 12:04:00	P4A00032	30032
83	Net Banking	20.00	Success	20.00	2020-05-15 12:04:00	P4A00033	30033
84	Net Banking	20.00	Success	20.00	2020-05-16 12:04:00	P4A00034	30034
85	Net Banking	10.00	Success	10.00	2020-05-17 12:04:00	P4A00035	30035
86	Net Banking	20.00	Success	20.00	2020-05-18 12:04:00	P4A00036	30036
87	Wallet	30.00	Success	30.00	2020-05-19 12:04:00	P4A00037	30037
88	Net Banking	40.00	Success	40.00	2020-05-20 12:04:00	P4A00038	30038
89	Debit	20.00	Success	20.00	2020-05-21 12:04:00	P4A00039	30039
90	Debit	10.00	Success	10.00	2020-05-22 12:04:00	P4A00040	30040

Action Output

Time	Action	Response	Duration / Fetch Time
165 - 03:19:24	CALL VIEW_BOOKED_SEATS(5000)	1 row(s) returned	0.00041 sec / 0.000...
167 03:19:36	SELECT * FROM MOVIE_SOCIAL_DISTANCE.SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
168 03:19:51	DROP PROCEDURE IF EXISTS View_B booked_seats	0 row(s) affected	0.0017 sec
169 03:19:51	CREATE PROCEDURE View_B booked_seats (IN shovid INTEGER) BEGIN DECLARE...	0 row(s) affected	0.0049 sec
170 03:19:55	CALL View_B booked_seats(5000)	1 row(s) returned	0.00073 sec / 0.000...
171 03:21:03	DROP PROCEDURE IF EXISTS View_B booked_seats	0 row(s) affected	0.0013 sec
172 03:21:03	CREATE PROCEDURE View_B booked_seats (IN shovid INTEGER) BEGIN SELECT...	0 row(s) affected	0.00096 sec
173 03:30:32	SELECT * FROM MOVIE_SOCIAL_DISTANCE.THEATRE LIMIT 0, 1000	3 row(s) returned	0.0016 sec / 0.000...
174 03:32:25	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
175 03:32:25	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00057 sec / 0.000...
176 03:32:25	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00031 sec / 0.000...

Query Completed

After calling the procedure

```

1 • CALL SEAT_BOOKING ('2','A1,A2,50010,502,204');
2 • SELECT * FROM BOOKING;
3 • SELECT * FROM PAYMENT;
4 • SELECT * FROM SHOW_SEAT;
5

```

PAYMENT_ID	PAYMENT_Mode	PAYMENT_Sub_Total	PAYMENT_Status	Total_With_Tax	PAYMENT_Time	Remote_Trans_ID	BOOKING_ID
74	Wallet	10.00	Success	10.00	2020-05-06 12:04:00	P4A00024	30024
75	Wallet	20.00	Success	20.00	2020-05-07 12:04:00	P4A00025	30025
76	Wallet	20.00	Success	20.00	2020-05-08 12:04:00	P4A00026	30026
77	Credit	10.00	Success	10.00	2020-05-09 12:04:00	P4A00027	30027
78	Wallet	20.00	Success	20.00	2020-05-10 12:04:00	P4A00028	30028
79	Net Banking	30.00	Success	30.00	2020-05-11 12:04:00	P4A00029	30029
80	Debit	40.00	Success	40.00	2020-05-12 12:04:00	P4A00030	30030
81	Credit	20.00	Success	20.00	2020-05-13 12:04:00	P4A00031	30031
82	Wallet	10.00	Success	10.00	2020-05-14 12:04:00	P4A00032	30032
83	Net Banking	20.00	Success	20.00	2020-05-15 12:04:00	P4A00033	30033
84	Net Banking	20.00	Success	20.00	2020-05-16 12:04:00	P4A00034	30034
85	Net Banking	10.00	Success	10.00	2020-05-17 12:04:00	P4A00035	30035
86	Net Banking	20.00	Success	20.00	2020-05-18 12:04:00	P4A00036	30036
87	Wallet	30.00	Success	30.00	2020-05-19 12:04:00	P4A00037	30037
88	Net Banking	40.00	Success	40.00	2020-05-20 12:04:00	P4A00038	30038
89	Debit	20.00	Success	20.00	2020-05-21 12:04:00	P4A00039	30039
90	Debit	10.00	Success	10.00	2020-05-22 12:04:00	P4A00040	30040

Action Output

Time	Action	Response	Duration / Fetch Time
191 03:39:23	CALL SEAT_BOOKING ('2','A1,A2,50010,502,204')	Error Code: 1364. Field 'PAYMENT_ID' doesn't have a default value.	0.0023 sec
192 03:43:00	ALTER TABLE PAYMENT MODIFY PAYMENT_ID INTEGER PRIMARY KEY AUTO_INCREMENT	Error Code: 1068. Multiple primary key defined	0.013 sec
193 03:43:12	ALTER TABLE PAYMENT MODIFY PAYMENT_ID INTEGER AUTO_INCREMENT	41 row(s) affected Records: 41 Duplicates: 0 Warnings: 0	0.0050 sec
194 03:43:22	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.0029 sec / 0.000...
195 03:43:45	DELETE FROM BOOKING WHERE SHOW_ID = 50010	1 row(s) affected	0.0015 sec
196 03:43:49	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00031 sec / 0.000...
197 03:43:55	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00028 sec / 0.000...
198 03:44:04	CALL SEAT_BOOKING ('2','A1,A2,50010,502,204')	1 row(s) affected	0.0049 sec
199 03:44:04	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00025 sec / 0.000...
200 03:44:04	SELECT * FROM PAYMENT LIMIT 0, 1000	42 row(s) returned	0.00027 sec / 0.000...
201 03:44:04	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	45 row(s) returned	0.00020 sec / 0.000...

Query Completed

Movie Ticket Booking System with Social Distancing

Term project report

SEAT BOOKING

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Local Instance 3306
- Administration:** Schemas
- Query 17:** PAYMENT, Booking_seat
- Limit to 1000 rows**
- Context Help:** Snippets
- Object Info:** Session
- Action Output:** Table: PAYMENT
- Columns:**
 - PAYMENT_ID: int PK
 - PAYMENT_Mode: varchar(15)
 - PAYMENT_Sub_Tot al: decimal(8,2)
 - PAYMENT_Status: varchar(10)
 - Total_Wth_Tax: decimal(8,2)
 - PAYMENT_Time: timestamp
 - Remote_Trans_ID: varchar(21)
 - BOOKING_ID: int
- Action Output Data:**

Time	Action	Response	Duration / Fetch Time
19/17:36	DROP PROCEDURE IF EXISTS `CREATEMOVIESHOW`	0 row(s) affected	0.032 sec
19/17:36	CREATE PROCEDURE `createMovieShow` (IN `s_SHOW_DATE` DATE , IN `s_SHOW_ST_`...	0 row(s) affected	0.019 sec
19/21:30	DROP PROCEDURE IF EXISTS `THEATRE_REMOVAL`	0 row(s) affected, 1 warning(s): 1305 PROCEDURE m...	0.00053 sec
19/21:30	CREATE PROCEDURE `THEATRE_REMOVAL` (IN `theatre_no` INTEGER) BEGIN UPD...	0 row(s) affected	0.0027 sec
19/32:15	DROP PROCEDURE IF EXISTS `MAKE_PAYMENT`	0 row(s) affected, 1 warning(s): 1305 PROCEDURE m...	0.0029 sec
19/32:15	#Insert the payment total with tax for the booking id, in order to start the payment...	0 row(s) affected	0.0030 sec
19/32:36	SELECT * FROM `MOVIE_SOCIAL_DISTANCES` PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00045 sec / 0.000...
19/33:28	DROP PROCEDURE IF EXISTS `MAKE_PAYMENT`	0 row(s) affected	0.0012 sec
19/33:28	#Insert the payment total with tax for the booking id, in order to start the payment...	0 row(s) affected	0.0017 sec
19/37:20	DROP PROCEDURE IF EXISTS `SEAT_BOOKING`	0 row(s) affected, 1 warning(s): 1305 PROCEDURE m...	0.0013 sec
19/37:20	# Assumption that only available seats are visible to the customer. CREATE PROC...	0 row(s) affected	0.016 sec
- Query Completed**

Before calling the procedure

The screenshot shows the MySQL Modeler interface with the following details:

- Top Bar:** Local instance 3306, MySQL Model*, EER Diagram, Administration, Schemas, Query 24, SHOW_SEAT.
- Left Panel (Administration):** Filtered objects include AIRLINE, BOOK_STORE, MOVIE_SOCIAL_DISTANC..., and Tables (BOOKING, CUSTOMER, EMPLOYEE, MOVIE, MOVIE_SHOW, PAYMENT, SCREEN, SEAT, SHOW_SEAT, THEATRE). Views, Stored Procedures, Functions, and other categories like MOVIE_SOCIAL_DISTANC... are also listed.
- Center Panel (Query Editor):** A query window displays:

```
1 • SELECT * FROM BOOKING;
2 • SELECT * FROM PAYMENT;
3 • SELECT * FROM SHOW_SEAT;
```

A note says "Limit to 1000 rows".
- Result Grid:** Shows the results of the query. The columns are BOOKING_ID, No.Of_Seats, BOOKING_Status, BOOKING_Price, BOOKING_Time, SHOW_ID, CUST_ID. The data includes multiple rows for each booking type.
- Bottom Panel (Object Info):** Action Output table with rows for BOOKING 4, PAYMENT 5, and SHOW_SEAT 6.
- Right Panel (Context Help):** Automatic context help is disabled. It provides links to Context Help, Snippets, Result Grid, Form Editor, Field Types, and Query Stats.

Movie Ticket Booking System with Social Distancing

Term project report

Local instance 3306 MySQL Model* EER Diagram

Administration Schemas Query 24 SHOW_SEAT

SCHEMAS Filter objects

- Airline
- BOOK_STORE
- MOVIE_SOCIAL_DISTAN...
- ▼ Tables
 - BOOKING
 - CUSTOMER
 - EMPLOYEE
 - MOVIE
 - MOVIE_SHOW
 - PAYMENT
 - SCREEN
 - SEAT
 - SHOW_SEAT
 - THEATRE
 - Views
- Stored Procedures
- Functions
- MOVIE_SOCIAL_DISTANC...
- MOVIE_SOCIAL_DISTANC...
- SaleCo_Products
- SolarSystem
- SPORTS

Result Grid Filter Rows: Search Edit Export/Import:

```

1 • SELECT * FROM BOOKING;
2 • SELECT * FROM PAYMENT;
3 • SELECT * FROM SHOW_SEAT;
4
5
    
```

PAYMENT_ID	PAYMENT_Mode	PAYMENT_Sub_Total	PAYMENT_Status	Total_With_Tax	PAYMENT_Time	Remote_Trans_ID	BOOKING_ID
73	Wallet	20.00	Success	20.00	2020-05-12 04:00	PAA00203	30023
74	Wallet	10.00	Success	10.00	2020-05-06 12:04:00	PAA00204	30024
75	Wallet	20.00	Success	20.00	2020-05-07 12:04:00	PAA00205	30025
76	Wallet	20.00	Success	20.00	2020-05-08 12:04:00	PAA00206	30026
77	Credit	10.00	Success	10.00	2020-05-09 12:04:00	PAA00207	30027
78	Wallet	20.00	Success	20.00	2020-05-10 12:04:00	PAA00208	30028
79	Net Banking	30.00	Success	30.00	2020-05-11 12:04:00	PAA00209	30029
80	Debit	40.00	Success	40.00	2020-05-12 12:04:00	PAA00300	30030
81	Credit	20.00	Success	20.00	2020-05-13 12:04:00	PAA00301	30031
82	Wallet	10.00	Success	10.00	2020-05-14 12:04:00	PAA00302	30032
83	Net Banking	20.00	Success	20.00	2020-05-15 12:04:00	PAA00303	30033
84	Net Banking	20.00	Success	20.00	2020-05-16 12:04:00	PAA00304	30034
85	Net Banking	10.00	Success	10.00	2020-05-17 12:04:00	PAA00305	30035
86	Net Banking	20.00	Success	20.00	2020-05-18 12:04:00	PAA00306	30036
87	Wallet	30.00	Success	30.00	2020-05-19 12:04:00	PAA00307	30037
88	Net Banking	40.00	Success	40.00	2020-05-20 12:04:00	PAA00308	30038
89	Debit	20.00	Success	20.00	2020-05-21 12:04:00	PAA00309	30039
90	Debit	10.00	Success	10.00	2020-05-22 12:04:00	PAA00400	30040

Action Output

Time	Action	Response	Duration / Fetch Time
166 03:19:24	CALL VIEW_BOOKED_SEATS(\$UUUU)	1 row(s) returned	0.000041 sec / 0.0000...
167 03:19:36	SELECT * FROM MOVIE_SOCIAL_DISTANCE.SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.000030 sec / 0.000...
168 03:19:51	DROP PROCEDURE IF EXISTS View_Booked_seats	0 row(s) affected	0.0017 sec
169 03:19:51	CREATE PROCEDURE View_Booked_seats (IN showid INTEGER) BEGIN DECLARE...	0 row(s) affected	0.0049 sec
170 03:19:55	CALL View_Booked_seats(50000)	1 row(s) returned	0.00073 sec / 0.0000...
171 03:21:03	DROP PROCEDURE IF EXISTS View_Booked_seats	0 row(s) affected	0.0013 sec
172 03:21:03	CREATE PROCEDURE View_Booked_seats (IN showid INTEGER) BEGIN SELECT...	0 row(s) affected	0.00096 sec
173 03:30:32	SELECT * FROM MOVIE_SOCIAL_DISTANCE.SHOW_SEAT LIMIT 0, 1000	3 row(s) returned	0.0016 sec / 0.0000...
174 03:32:25	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
175 03:32:25	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00057 sec / 0.0000...
176 03:32:25	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00031 sec / 0.0000...

Query Completed

Local instance 3306 MySQL Model* EER Diagram

Administration Schemas Query 24 SHOW_SEAT

SCHEMAS Filter objects

- Airline
- BOOK_STORE
- MOVIE_SOCIAL_DISTAN...
- ▼ Tables
 - BOOKING
 - CUSTOMER
 - EMPLOYEE
 - MOVIE
 - MOVIE_SHOW
 - PAYMENT
 - SCREEN
 - SEAT
 - SHOW_SEAT
 - THEATRE
 - Views
- Stored Procedures
- Functions
- MOVIE_SOCIAL_DISTANC...
- MOVIE_SOCIAL_DISTANC...
- SaleCo_Products
- SolarSystem
- SPORTS

Result Grid Filter Rows: Search Edit Export/Import:

```

1 • SELECT * FROM BOOKING;
2 • SELECT * FROM PAYMENT;
3 • SELECT * FROM SHOW_SEAT;
4
5
    
```

SHOW_SEAT_ID	SHOW_SEAT_Status	SEAT_ID	SHOW_ID	BOOKING_ID
70023	0	10032	50002	30023
70024	0	10004	50002	30024
70025	0	10026	50003	30026
70026	0	10026	50003	30026
70027	0	10027	50003	30027
70028	0	10028	50003	30028
70029	0	10029	50003	30029
70030	0	10030	50003	30030
70031	0	10031	50003	30031
70032	0	10032	50003	30032
70033	0	10033	50004	30033
70034	0	10034	50004	30034
70035	0	10035	50004	30035
70036	0	10036	50004	30036
70037	0	10037	50004	30037
70038	0	10038	50004	30038
70039	0	10039	50003	30039
70040	0	10040	50003	30040

Action Output

Time	Action	Response	Duration / Fetch Time
166 03:19:24	CALL VIEW_BOOKED_SEATS(\$UUUU)	1 row(s) returned	0.000041 sec / 0.0000...
167 03:19:36	SELECT * FROM MOVIE_SOCIAL_DISTANCE.SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.000030 sec / 0.000...
168 03:19:51	DROP PROCEDURE IF EXISTS View_Booked_seats	0 row(s) affected	0.0017 sec
169 03:19:51	CREATE PROCEDURE View_Booked_seats (IN showid INTEGER) BEGIN DECLARE...	0 row(s) affected	0.0049 sec
170 03:19:55	CALL View_Booked_seats(50000)	1 row(s) returned	0.00073 sec / 0.0000...
171 03:21:03	DROP PROCEDURE IF EXISTS View_Booked_seats	0 row(s) affected	0.0013 sec
172 03:21:03	CREATE PROCEDURE View_Booked_seats (IN showid INTEGER) BEGIN SELECT...	0 row(s) affected	0.00096 sec
173 03:30:32	SELECT * FROM MOVIE_SOCIAL_DISTANCE.SHOW_SEAT LIMIT 0, 1000	3 row(s) returned	0.0016 sec / 0.0000...
174 03:32:25	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00030 sec / 0.000...
175 03:32:25	SELECT * FROM PAYMENT LIMIT 0, 1000	41 row(s) returned	0.00057 sec / 0.0000...
176 03:32:25	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00031 sec / 0.0000...

Query Completed

Movie Ticket Booking System with Social Distancing

Term project report

After calling the procedure

Schemas

```

1 • CALL SEAT_BOOKING ('2','A:1,A:2',50010,502,204);
2 • SELECT * FROM BOOKING;
3 • SELECT * FROM PAYMENT;
4 • SELECT * FROM SHOW_SEAT;
5

```

Result Grid

BOOKING_ID	No_Of_Seats	BOOKING_Status	BOOKING_Price	BOOKING_Time	SHOW_ID	CUST_ID
30024	1	Success	10.00	2020-05-06 12:00:00	50006	204
30025	2	Success	20.00	2020-05-07 12:00:00	50006	205
30026	2	Success	20.00	2020-05-08 12:00:00	50006	206
30027	1	Success	10.00	2020-05-09 12:00:00	50006	200
30028	2	Success	20.00	2020-05-10 12:00:00	50006	201
30029	3	Success	30.00	2020-05-11 12:00:00	50007	203
30030	4	Success	40.00	2020-05-12 12:00:00	50007	204
30031	2	Success	20.00	2020-05-13 12:00:00	50007	205
30032	1	Success	10.00	2020-05-14 12:00:00	50008	200
30033	2	Success	20.00	2020-05-15 12:00:00	50008	201
30034	2	Success	20.00	2020-05-16 12:00:00	50008	202
30035	1	Success	10.00	2020-05-17 12:00:00	50008	203
30036	2	Success	20.00	2020-05-18 12:00:00	50008	204
30037	3	Success	30.00	2020-05-19 12:00:00	50008	205
30038	4	Success	40.00	2020-05-20 12:00:00	50009	203
30039	2	Success	20.00	2020-05-21 12:00:00	50009	204
30040	1	Success	10.00	2020-05-22 12:00:00	50009	205
30043	2	Success	40.00	2020-05-27 03:44:04	50010	204

Action Output

Time	Action	Response	Duration / Fetch Time
191 03:39:23	CALL SEAT_BOOKING ('2','A:1,A:2',50010,502,204)	Error Code: 1364, Field 'PAYMENT_ID' doesn't have a...	0.0023 sec
192 03:43:00	ALTER TABLE PAYMENT MODIFY PAYMENT_ID INTEGER PRIMARY KEY AUTO_INCREMENT	Error Code: 1068. Multiple primary key defined	0.013 sec
193 03:43:12	ALTER TABLE PAYMENT MODIFY PAYMENT_ID INTEGER AUTO_INCREMENT	41 row(s) affected Records: 41 Duplicates: 0 Warning...	0.060 sec
194 03:43:22	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00029 sec / 0.000...
195 03:43:45	DELETE FROM BOOKING WHERE SHOW_ID = 50010	1 row(s) affected	0.0015 sec
196 03:43:49	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00031 sec / 0.000...
197 03:43:55	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00028 sec / 0.000...
198 03:44:04	CALL SEAT_BOOKING ('2','A:1,A:2',50010,502,204)	1 row(s) affected	0.0049 sec
199 03:44:04	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00025 sec / 0.000...
200 03:44:04	SELECT * FROM PAYMENT LIMIT 0, 1000	42 row(s) returned	0.00027 sec / 0.000...
201 03:44:04	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	45 row(s) returned	0.00020 sec / 0.000...

Schemas

```

1 • CALL SEAT_BOOKING ('2','A:1,A:2',50010,502,204);
2 • SELECT * FROM BOOKING;
3 • SELECT * FROM PAYMENT;
4 • SELECT * FROM SHOW_SEAT;
5

```

Result Grid

PAYMENT_ID	PAYMENT_Mode	PAYMENT_Sub_Total	PAYMENT_Status	Total_With_Tax	PAYMENT_Time	Remote_Trans_ID	BOOKING_ID
74	Wallet	10.00	Success	10.00	2020-05-06 12:04:00	P4400024	30024
75	Wallet	20.00	Success	20.00	2020-05-07 12:04:00	P4400025	30025
76	Wallet	20.00	Success	20.00	2020-05-08 12:04:00	P4400026	30026
77	Credit	10.00	Success	10.00	2020-05-09 12:04:00	P4400027	30027
78	Wallet	20.00	Success	20.00	2020-05-10 12:04:00	P4400028	30028
79	Net Banking	30.00	Success	30.00	2020-05-11 12:04:00	P4400029	30029
80	Debit	40.00	Success	40.00	2020-05-12 12:04:00	P4400030	30030
81	Credit	20.00	Success	20.00	2020-05-13 12:04:00	P4400031	30031
82	Wallet	10.00	Success	10.00	2020-05-14 12:04:00	P4400032	30032
83	Net Banking	20.00	Success	20.00	2020-05-15 12:04:00	P4400033	30033
84	Net Banking	20.00	Success	20.00	2020-05-16 12:04:00	P4400034	30034
85	Net Banking	10.00	Success	10.00	2020-05-17 12:04:00	P4400035	30035
86	Net Banking	20.00	Success	20.00	2020-05-18 12:04:00	P4400036	30036
87	Wallet	30.00	Success	30.00	2020-05-19 12:04:00	P4400037	30037
88	Net Banking	40.00	Success	40.00	2020-05-20 12:04:00	P4400038	30038
89	Debit	20.00	Success	20.00	2020-05-21 12:04:00	P4400039	30039
90	Debit	10.00	Success	10.00	2020-05-22 12:04:00	P4400040	30040
91		40.00		87.20			30043

Action Output

Time	Action	Response	Duration / Fetch Time
191 03:39:23	CALL SEAT_BOOKING ('2','A:1,A:2',50010,502,204)	Error Code: 1364, Field 'PAYMENT_ID' doesn't have a...	0.0023 sec
192 03:43:00	ALTER TABLE PAYMENT MODIFY PAYMENT_ID INTEGER PRIMARY KEY AUTO_INCREMENT	Error Code: 1068. Multiple primary key defined	0.013 sec
193 03:43:12	ALTER TABLE PAYMENT MODIFY PAYMENT_ID INTEGER AUTO_INCREMENT	41 row(s) affected Records: 41 Duplicates: 0 Warning...	0.060 sec
194 03:43:22	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00029 sec / 0.000...
195 03:43:45	DELETE FROM BOOKING WHERE SHOW_ID = 50010	1 row(s) affected	0.0015 sec
196 03:43:49	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00031 sec / 0.000...
197 03:43:55	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) affected	0.00028 sec / 0.000...
198 03:44:04	CALL SEAT_BOOKING ('2','A:1,A:2',50010,502,204)	1 row(s) affected	0.0049 sec
199 03:44:04	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00025 sec / 0.000...
200 03:44:04	SELECT * FROM PAYMENT LIMIT 0, 1000	42 row(s) returned	0.00027 sec / 0.000...
201 03:44:04	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	45 row(s) returned	0.00020 sec / 0.000...

Movie Ticket Booking System with Social Distancing

Term project report

The screenshot shows the MySQL Workbench interface. In the top navigation bar, it says "Local instance 3306" and "MySQL Model". Below the toolbar, there are tabs for "Administration", "Schemas", "Query 24", "SHOW_SEAT", "Booking_seat", "Make_Payment", and "project_table_creation". The "Schemas" tab is selected, showing the database structure with tables like AIRLINE, BOOK_STORE, and MOVIE_SOCIAL_DISTANCING. The "Query 24" tab contains the following SQL code:

```

1 • CALL SEAT_BOOKING ('2','A:1,A:2',50010,502,204);
2 • SELECT * FROM BOOKING;
3 • SELECT * FROM PAYMENT;
4 • SELECT * FROM SHOW_SEAT;
5

```

The "Result Grid" tab displays the results of the last query, showing columns: SHOW_SEAT_ID, SHOW_SEAT_Status, SEAT_ID, SHOW_ID, and BOOKING_ID. The results are as follows:

SHOW_SEAT_ID	SHOW_SEAT_Status	SEAT_ID	SHOW_ID	BOOKING_ID
70027	0	10027	50003	30027
70028	0	10028	50003	30028
70029	0	10029	50003	30029
70030	0	10030	50003	30030
70031	0	10031	50003	30031
70032	0	10032	50003	30032
70033	0	10033	50004	30033
70034	0	10034	50004	30034
70035	0	10035	50004	30035
70036	0	10036	50004	30036
70037	0	10037	50004	30037
70038	0	10038	50004	30038
70039	0	10039	50004	30039
70040	0	10040	50004	30040
70041	0	10300	50010	30043
70042	0	10301	50010	30043
70043	0	10302	50010	30043
70044	0	10303	50010	30043

The "Timeline" section at the bottom shows the sequence of events:

Time	Action	Response	Duration / Fetch Time
191 03:43:23	CALL SEAT_BOOKING ('2','A:1,A:2',50010,502,204)	Error Code: 1364, Field 'PAYMENT_ID' doesn't have a...	0.0023 sec
192 03:43:00	ALTER TABLE PAYMENT MODIFY PAYMENT_ID INTEGER PRIMARY KEY AUTO_INCREMENT	Error Code: 1068. Multiple primary key defined	0.013 sec
193 03:43:12	ALTER TABLE PAYMENT MODIFY PAYMENT_ID INTEGER AUTO_INCREMENT	41 row(s) affected Records: 41 Duplicates: 0 Warnings: 0	0.060 sec
194 03:43:22	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00029 sec / 0.000...
195 03:43:45	DELETE FROM BOOKING WHERE SHOW_ID = 50010	1 row(s) affected	0.0015 sec
196 03:43:49	SELECT * FROM BOOKING LIMIT 0, 1000	41 row(s) returned	0.00031 sec / 0.000...
197 03:43:55	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	41 row(s) returned	0.00028 sec / 0.000...
198 03:44:04	CALL SEAT_BOOKING ('2','A:1,A:2',50010,502,204)	1 row(s) affected	0.0049 sec
199 03:44:04	SELECT * FROM BOOKING LIMIT 0, 1000	42 row(s) returned	0.00025 sec / 0.000...
200 03:44:04	SELECT * FROM PAYMENT LIMIT 0, 1000	42 row(s) returned	0.00027 sec / 0.000...
201 03:44:04	SELECT * FROM SHOW_SEAT LIMIT 0, 1000	45 row(s) returned	0.00020 sec / 0.000...

Query Description

The following are the description of all the queries, triggers, stored procedures and events that are being used in this database.

Event

i) Sanitize_Show

There is only one event in the database. This event updates the Is_Sanitized field in the movie_show table to 0 after the end of each show. This indicates to the employee that the show has ended and that they have to start the sanitization process.

```
DROP EVENT IF EXISTS SANITIZE_SHOW;
```

```
SET GLOBAL event_scheduler = ON;
```

```
CREATE EVENT SANITIZE_SHOW
    ON SCHEDULE EVERY 1 MINUTE
    STARTS CURRENT_TIMESTAMP + INTERVAL 1 MINUTE
DO
    UPDATE MOVIE_SHOW SET Is_Sanitized = 0 WHERE
    SHOW_Date = DATE_FORMAT(NOW(), '%Y-%m-%d')
    AND DATE_FORMAT(SHOW_Endtime, '%h:%i') =
    DATE_FORMAT(NOW(), '%h:%i');
```

Triggers

i) **BOOK_SHOW_BYSTATUS**

If the payment is failed for a booking id then this trigger updates the booking status to failure in the booking table and deletes the blocked seats for the booking id in the show_seat table.

```
DROP TRIGGER IF EXISTS BOOK_SHOW_BYSTATUS;
```

```
DELIMITER $$
```

```
CREATE TRIGGER BOOK_SHOW_BYSTATUS BEFORE UPDATE ON
PAYMENT FOR EACH ROW
```

```
BEGIN
```

```
    IF NEW.PAYMENT_Status = 'Failure' THEN
        UPDATE BOOKING SET BOOKING_Status = 'Failure'
        WHERE BOOKING_ID = NEW.BOOKING_ID;
        DELETE FROM SHOW_SEAT WHERE BOOKING_ID =
        NEW.BOOKING_ID;
```

```
    END IF;
```

```
END$$
```

```
DELIMITER ;
```

ii) **Upd_user**

The password field should not be null. In order to ensure this, this trigger updates the password of the customer if the password field is null or empty.

```
DROP TRIGGER IF EXISTS upd_user;
```

```
DELIMITER $$  
CREATE TRIGGER upd_user BEFORE UPDATE ON customer  
    FOR EACH ROW  
BEGIN  
    IF (NEW.cust_pwd IS NULL OR NEW.cust_pwd = '') THEN  
        SET NEW.cust_pwd = OLD.cust_pwd;  
    ELSE  
        SET NEW.cust_pwd = NEW.cust_pwd;  
    END IF;  
END$$  
  
DELIMITER ;
```

iii) Upd_seat_price

Since there might be a price range for seats inside the screen. This trigger sets the price for the seats that are allotted for the price range after the seats are inserted.

```
DROP TRIGGER IF EXISTS upd_seat_price;  
  
DELIMITER $$  
  
CREATE TRIGGER upd_seat_price BEFORE INSERT ON SEAT FOR  
    EACH ROW  
BEGIN  
  
    IF NEW.ROW_No = 'A' OR NEW.ROW_No = 'B' THEN  
        SET NEW.SEAT_price = 20;  
    ELSE  
        IF NEW.ROW_No = 'C' OR NEW.ROW_No = 'D' THEN  
            SET NEW.SEAT_price = 15;  
        ELSE  
            SET NEW.SEAT_price = 10;  
        END IF;  
    END IF;  
  
END$$
```

DELIMITER ;

Stored Procedures

i) BLOCK_ALTERNATE_ROWS()

To follow the social distancing inside the screens, we have to mandatorily block alternate rows. This is a temporary change we have a separate procedure to perform this action

```
DROP PROCEDURE IF EXISTS BLOCK_ALTERNATE_ROWS;
```

```
DELIMITER $$
```

```
CREATE PROCEDURE BLOCK_ALTERNATE_ROWS ()
```

```
BEGIN
```

```
    UPDATE SEAT SET Is_Available = 0 WHERE ROW_No = 'B' OR  
    ROW_No = 'D' OR ROW_No = 'F';  
END $$
```

DELIMITER ;

ii) createEmployee

Procedure to insert employee details

```
DROP PROCEDURE IF EXISTS createEmployee;
```

```
DELIMITER $$
```

```
CREATE PROCEDURE createEmployee(  
    e_EMP_LastName VARCHAR(100),  
    e_EMP_FirstName    VARCHAR(100),  
    e_EMP_Age INTEGER,  
    e_EMP_Designation VARCHAR(50),  
    e_EMP_Email      VARCHAR(100),  
    e_EMP_Contact    VARCHAR(15) ,  
    e_Theatre_ID INTEGER)
```

```
BEGIN
```

```
INSERT INTO
EMPLOYEE(EMP_LastName,EMP_FirstName,EMP_Age,EMP_DE
SIGNATION,EMP_Email,EMP_Contact,Theatre_ID)
VALUES
(e_EMP_LastName,e_EMP_FirstName,e_EMP_Age,e_EMP_Design
ation,e_EMP_Email,e_EMP_Contact,e_Theatre_ID);

END$$
```

```
DELIMITER ;
```

iii) createCustomer

Procedure to insert customer details.

```
DROP PROCEDURE IF EXISTS createCustomer;
```

```
DELIMITER $$
```

```
CREATE PROCEDURE createCustomer(
    IN c_CUST_LastName VARCHAR(50),
    IN c_CUST_FirstName VARCHAR(50),
    IN c_CUST_Age INTEGER ,
    IN c_CUST_Username VARCHAR(50),
    IN c_CUST_Pwd VARCHAR(15),
    IN c_CUST_Email VARCHAR(50),
    IN c_CUST_Contact VARCHAR(10)
)
```

```
BEGIN
```

```
    INSERT INTO
CUSTOMER(CUST_LastName,CUST_FirstName,CUST_Age,CUST
_Username,CUST_Pwd,CUST_Email,CUST_Contact,
CUST_DOJ)
VALUES(c_CUST_LastName,c_CUST_FirstName,c_CUST_Age,c_
CUST_Username,c_CUST_Pwd,c_CUST_Email,
c_CUST_Contact,DATE_FORMAT(NOW(),'%Y-%m-%d'));
```

```
END$$
```

DELIMITER ;

iv) createMovie

Procedure to insert movie details

```
DROP PROCEDURE IF EXISTS createMovie;
```

```
DELIMITER $$
```

```
CREATE PROCEDURE createMovie(
```

```
    IN    m_MOVIE_Title  VARCHAR(50) ,  
    IN    m_MOVIE_Desc   VARCHAR(400),  
    IN    m_MOVIE_Genre VARCHAR(50),  
    IN    m_MOVIE_Language  VARCHAR(50),  
    IN    m_MOVIE_Duration TIME,  
    IN    m_MOVIE_Release   DATE,  
    IN    m_MOVIE_Rating    DECIMAL(5,2)
```

```
)
```

```
BEGIN
```

```
    INSERT          INTO          MOVIE(MOVIE_Title  
,MOVIE_Desc,MOVIE_Genre,MOVIE_Language,MOVIE_Duration  
,MOVIE_Release,MOVIE_Rating)  
    VALUES(m_MOVIE_Title,m_MOVIE_Desc,m_MOVIE_Genre,m_  
MOVIE_Language,m_MOVIE_Duration,m_MOVIE_Release,m_MO  
VIE_Rating);
```

```
END$$
```

DELIMITER ;

v) createMovieShow

Procedure to enter movie show details

```
DROP PROCEDURE IF EXISTS createMovieShow;
```

```
DELIMITER $$
```

```
CREATE PROCEDURE createMovieShow(
    IN s_SHOW_Date DATE ,
    IN s_SHOW_Starttime      TIME,
    IN s_SHOW_Endtime      TIME,
    IN s_Is_Sanitized  BOOL,
    IN s_SCREEN_ID INTEGER,
    IN s_MOVIE_ID  INTEGER
)
BEGIN
    INSERT INTO MOVIE_SHOW(SHOW_Date,SHOW_Starttime
    ,SHOW_Endtime,Is_Sanitized,SCREEN_ID,MOVIE_ID)
    VALUES(s_SHOW_Date,s_SHOW_Starttime,s_SHOW_Endtime,s_I
    s_Sanitized,s_SCREEN_ID,s_MOVIE_ID);
END$$

DELIMITER ;
```

vi) **MAKE_PAYMENT**

Procedure to update the payment table with the details of the booking.

```
DROP PROCEDURE IF EXISTS MAKE_PAYMENT;

DELIMITER $$
```

#Insert the payment total with tax for the booking id, in order to start the payment process.

```
CREATE PROCEDURE MAKE_PAYMENT (IN book_id INTEGER, IN
subtotal DECIMAL(8,2))
```

```
BEGIN
    DECLARE total_with_tax DECIMAL(8,2);
    SET total_with_tax = subtotal + (subtotal * 1.18);
```

Movie Ticket Booking System with Social Distancing
Term project report

```
INSERT INTO PAYMENT
(PAYMENT_Sub_Total,Total_With_Tax,BOOKING_ID) VALUES
(subtotal,total_with_tax, book_id);
END $$

DELIMITER ;
```

vii) View_Booked_seats

Returns the booked seats for a particular show.

```
DROP PROCEDURE IF EXISTS View_Booked_seats;
```

```
DELIMITER $$
```

```
CREATE PROCEDURE View_Booked_seats (IN showid INTEGER)
```

```
BEGIN
```

```
SELECT movie_show.SHOW_ID,
       count(show_seat.SHOW_SEAT_Status) as booked_seats
  FROM seat
 JOIN show_seat on seat.seat_id = show_seat.seat_id
 JOIN movie_show on movie_show.show_id=show_seat.show_id
 WHERE show_seat.SHOW_ID=showid
 GROUP BY movie_show.show_id;
```

```
END $$
```

```
DELIMITER ;
```

viii) THEATRE_REMOVAL

Procedure to remove a theatre.

```
DROP PROCEDURE IF EXISTS THEATRE_REMOVAL;
```

```
DELIMITER $$
```

Movie Ticket Booking System with Social Distancing
Term project report

```
CREATE PROCEDURE THEATRE_REMOVAL (IN theatre_no
INTEGER)

BEGIN
    UPDATE MOVIE_SHOW SET MOVIE_ID = NULL WHERE
SCREEN_ID IN
(SELECT SCREEN_ID FROM SCREEN WHERE
THEATRE_ID=theatre_no);

    DELETE FROM SHOW_SEAT WHERE SHOW_ID IN
(SELECT SHOW_ID FROM MOVIE_SHOW WHERE
SCREEN_ID IN
(SELECT SCREEN_ID FROM SCREEN WHERE
THEATRE_ID = theatre_no));

    DELETE FROM PAYMENT WHERE BOOKING_ID IN
(SELECT BOOKING_ID FROM BOOKING WHERE
SHOW_ID IN
(SELECT SHOW_ID FROM MOVIE_SHOW WHERE
SCREEN_ID IN
(SELECT SCREEN_ID FROM SCREEN WHERE
THEATRE_ID = theatre_no)));

    DELETE FROM BOOKING WHERE SHOW_ID IN
(SELECT SHOW_ID FROM MOVIE_SHOW WHERE
SCREEN_ID IN
(SELECT SCREEN_ID FROM SCREEN WHERE
THEATRE_ID = theatre_no));

    DELETE FROM MOVIE_SHOW WHERE SCREEN_ID IN
(SELECT SCREEN_ID FROM SCREEN WHERE
THEATRE_ID= theatre_no);

    DELETE FROM SEAT WHERE SCREEN_ID IN
(SELECT SCREEN_ID FROM SCREEN WHERE
THEATRE_ID= theatre_no);

    DELETE FROM SCREEN WHERE THEATRE_ID=
theatre_no;
```

Movie Ticket Booking System with Social Distancing
Term project report

```
DELETE FROM EMPLOYEE WHERE THEATRE_ID=
theatre_no;
```

```
DELETE FROM THEATRE WHERE THEATRE_ID =
theatre_no;
```

```
END$$
```

```
DELIMITER ;
```

ix) SEAT_BOOKING

Procedure to book the seats with social distancing. This procedure books the seats given by the customer. It also books 2 seats on either side of the original booking, so that social distancing is followed between two bookings. It calculates the price for the original booking and updates the total amount to be paid for the booking by the customer in the payment table using the MAKE_PAYMENT procedure which will be further taken for remote transaction for the payment to happen.

```
DROP PROCEDURE IF EXISTS SEAT_BOOKING;
```

```
DELIMITER $$
```

```
# Assumption that only available seats are visible to the customer
```

```
CREATE PROCEDURE SEAT_BOOKING (
    IN seats INTEGER,
    IN seat_num VARCHAR(50),
    IN showid INTEGER,
    IN theatreid INTEGER,
    IN custid INTEGER
)
```

```
BEGIN
```

```
    DECLARE counter1 INTEGER;
    DECLARE counter2 INTEGER;
    DECLARE seatprice DECIMAL(8,2);
    DECLARE book_price DECIMAL(8,2);
    DECLARE rowno VARCHAR(20);
    DECLARE seatno INTEGER;
```

Movie Ticket Booking System with Social Distancing
Term project report

```
DECLARE seat VARCHAR(20);
DECLARE seatid INTEGER;
DECLARE bookid INTEGER;
DECLARE screenid INTEGER;
DECLARE rowno_seat1 VARCHAR(20);
DECLARE old_rowno VARCHAR(20);
DECLARE old_seatno INTEGER;
DECLARE old_seatid INTEGER;

SET counter1 = 1;
SET counter2 = 1;
SET book_price = 0;

# Fetch the screen_id for the given show_id

SET screenid = (SELECT SCREEN_ID FROM MOVIE_SHOW
WHERE SHOW_ID = showid);

# Calculate the price for the given seats

seatloop: LOOP
    IF counter1 <= seats THEN

        SET seat =
        SUBSTRING_INDEX(SUBSTRING_INDEX(seat_num,',',cou
nter1), ',', -1);

        SET rowno =
        SUBSTRING_INDEX(SUBSTRING_INDEX(seat,':',1), ':', -1);

        SET seatno =
        SUBSTRING_INDEX(SUBSTRING_INDEX(seat,':',2), ':', -1);

        # Fetch the seat id for the given combination of row no and seat no

        SET seatid = (SELECT SEAT_ID FROM SEAT WHERE
ROW_No = rowno AND SEAT_No = seatno AND
SCREEN_ID = screenid);

        # Fetch the price for each price and add to the total booking price
```

Movie Ticket Booking System with Social Distancing
Term project report

```
SET seatprice = (SELECT SEAT_Price FROM SEAT WHERE
SEAT_ID = seatid);

SET book_price = book_price + seatprice;

ELSE
    LEAVE seatloop;

END IF;

SET counter1 = counter1 + 1;

END LOOP;

# Insert the booking details for the given number of seats

        INSERT           INTO           BOOKING
(No_of_Seats,BOOKING_STATUS,BOOKING_Price,BOOKING_Time,S
HOW_ID,CUST_ID) VALUES
(seats,'Success', book_price, CURRENT_TIMESTAMP, showid, custid);

# Fetch the book_id for the inserted booking details

SET bookid = (SELECT BOOKING_ID FROM BOOKING WHERE
CUST_ID = custid AND DATE_FORMAT(BOOKING_Time,'%y-$m-$d
%h:%i') = DATE_FORMAT(NOW(),'%y-$m-$d %h:%i') AND
No_of_Seats = seats AND SHOW_ID = showid);

# update the payment table with price and the booking id to process the
payment

CALL MAKE_PAYMENT(bookid,book_price);
```

Loop is used to split the seat_num to the necessary fields in the tables.
seat_num will be in the combination of format of 'rowno:seatno'

seatloop: LOOP
IF counter2 <= seats THEN
 SET seat =

Movie Ticket Booking System with Social Distancing
Term project report

```
SUBSTRING_INDEX(SUBSTRING_INDEX(seat_num,',',counter2), ',', -1);
```

```
SET rowno =  
SUBSTRING_INDEX(SUBSTRING_INDEX(seat,':',1), ':', -1);
```

```
SET seatno =  
SUBSTRING_INDEX(SUBSTRING_INDEX(seat,':',2), ':', -1);
```

```
# Fetch the seat id for the given combination of row no and seat no
```

```
SET seatid = (SELECT SEAT_ID FROM SEAT WHERE  
ROW_No = rowno AND SEAT_No = seatno AND  
SCREEN_ID = screenid);
```

```
# Insert the entries into show_seat table to block the given seats for the  
corressponding show
```

```
INSERT INTO SHOW_SEAT  
(SHOW_SEAT_STATUS,SEAT_ID,SHOW_ID,BOOKING_ID  
) VALUES (0,seatid,showid,bookid);
```

```
# Logic to implement blocking of adjacent seats for the given booking  
starts...
```

```
# Assume all the given seats are booked for the rowno of the first given seat.
```

```
IF counter2 = 1 THEN  
    SET rowno_seat1 = rowno;  
END IF;
```

```
IF rowno = rowno_seat1 THEN
```

```
# Block the adjacent seats on the left side of the first seat of the given  
booking
```

```
IF counter2 = 1 THEN  
    IF seatno <> 1 AND seatno <> 2 THEN  
        INSERT INTO SHOW_SEAT  
(SHOW_SEAT_STATUS,SEAT_ID,SHOW  
_ID,BOOKING_ID) VALUES
```

Movie Ticket Booking System with Social Distancing
Term project report

```
(0,seatid-1,showid,bookid);

        INSERT      INTO      SHOW_SEAT
        (SHOW_SEAT_STATUS,SEAT_ID,SHOW
         _ID,BOOKING_ID) VALUES
        (0,seatid-2,showid,bookid);

    ELSE
        IF seatno = 2 THEN
            INSERT      INTO      SHOW_SEAT
            (SHOW_SEAT_STATUS,SEAT_ID,S
             HOW_ID,BOOKING_ID)  VALUES
            (0,seatid-1,showid,bookid);
        END IF;
    END IF;
END IF;

# Block the adjacent seats on the right side of the last seat of the given
booking

IF counter2 = seats THEN
    IF seatno <> 9 AND seatno <> 10 THEN
        INSERT      INTO      SHOW_SEAT
        (SHOW_SEAT_STATUS,SEAT_ID,SHOW
         _ID,BOOKING_ID) VALUES
        (0,seatid+1,showid,bookid);

        INSERT      INTO      SHOW_SEAT
        (SHOW_SEAT_STATUS,SEAT_ID,SHOW
         _ID,BOOKING_ID) VALUES
        (0,seatid+2,showid,bookid);

    ELSE
        IF seatno = 9 THEN
            INSERT      INTO      SHOW_SEAT
            (SHOW_SEAT_STATUS,SEAT_ID,S
             HOW_ID,BOOKING_ID)
            VALUES(0,seatid+1,showid,bookid);
        END IF;
    END IF;
END IF;
```

Movie Ticket Booking System with Social Distancing
Term project report

Store the previous row-seat combination for futher processing if required

```
SET old_rowno = rowno;  
SET old_seatno = seatno;  
SET old_seatid = seatid;
```

ELSE

Logic to implement if the customer books seats in different rows.

Set the new rowno to the temporary variable

```
SET rowno_seat1 = rowno;
```

Block the adjacent seats on the right side of the previous row

```
IF old_seatno <> 9 AND old_seatno <> 10 THEN  
    INSERT      INTO      SHOW_SEAT  
    (SHOW_SEAT_STATUS,SEAT_ID,SHOW_ID,B  
     OOKING_ID)          VALUES  
    (0,old_seatid+1,showid,bookid);
```

```
    INSERT      INTO      SHOW_SEAT  
    (SHOW_SEAT_STATUS,SEAT_ID,SHOW_ID,B  
     OOKING_ID)          VALUES  
    (0,old_seatid+2,showid,bookid);
```

ELSE

```
    IF old_seatno = 9 THEN  
        INSERT      INTO      SHOW_SEAT  
        (SHOW_SEAT_STATUS,SEAT_ID,SHOW  
         _ID,BOOKING_ID) VALUES  
        (0,old_seatid+1,showid,bookid);
```

END IF;

END IF;

Block the adjacent seats on the left side of the seat in the new row

```
IF seatno <> 1 AND seatno <> 2 THEN
```

```
    INSERT      INTO      SHOW_SEAT  
    (SHOW_SEAT_STATUS,SEAT_ID,SHOW_ID,B  
     OOKING_ID)          VALUES  
    (0,seatid-1,showid,bookid);
```

Movie Ticket Booking System with Social Distancing
Term project report

```
        INSERT      INTO      SHOW_SEAT
        (SHOW_SEAT_STATUS,SEAT_ID,SHOW_ID,B
         OOKING_ID)           VALUES
        (0,seatid-2,showid,bookid);

    ELSE
        IF seatno = 2 THEN
            INSERT      INTO      SHOW_SEAT
            (SHOW_SEAT_STATUS,SEAT_ID,SHOW
             _ID,BOOKING_ID) VALUES
            (0,seatid-1,showid,bookid);
        END IF;
    END IF;
```

If this is the only seat in the new row, then block the adjacent seats on the right side of the seat.

```
IF counter2 = seats THEN
    IF seatno <> 9 AND seatno <> 10 THEN
        INSERT      INTO      SHOW_SEAT
        (SHOW_SEAT_STATUS,SEAT_ID,SHOW
         _ID,BOOKING_ID) VALUES
        (0,seatid+1,showid,bookid);

        INSERT      INTO      SHOW_SEAT
        (SHOW_SEAT_STATUS,SEAT_ID,SHOW
         _ID,BOOKING_ID) VALUES
        (0,seatid+2,showid,bookid);

    ELSE
        IF seatno = 9 THEN
            INSERT      INTO      SHOW_SEAT
            (SHOW_SEAT_STATUS,SEAT_ID,S
             HOW_ID,BOOKING_ID)
            VALUES(0,seatid+1,showid,bookid);
        END IF;
    END IF;
END IF;
```

END IF;

```
ELSE
    LEAVE seatloop;

END IF;

SET counter2 = counter2 + 1;
END LOOP;

END$$
DELIMITER ;
```

Summary & Conclusion

A movie ticketing system database design has been developed in such a way that social distancing is maintained for each booking. To ensure this, alternate rows are blocked and adjacent seats (2 seats on each side of the booking) per booking are blocked in a manner that social distance of 6ft (which is considered to be safe in public places) is maintained. In this way pandemic guidelines will be followed within the screens in the theatre. Along with this, a sanitization check is introduced for each show, which informs the theatre management that a particular show is over, and they have to start their sanitization process. It is implemented via events in MySQL. This event runs every minute and checks for end time of the show and if yes, it will automatically update the Boolean field `Is_Sanitized` for the corresponding screen. Only a user who has access to write the database can update the `Is_sanitized` value for each screen. Through this we can have the theatre management track the exclusive sanitization process through which they maintain the cleanliness and abide by the government in this pandemic.

Future work

As each movie will have reviews, trailers, ratings, these data will increase as the movie increases. This data can be considered as big data. Hence in this case we need to use NoSQL. Cassandra can be used.

An automatic report generation of the show bookings can be sent to the theatre management on a weekly/monthly basis which will be helpful to improve their business.

- An automatic report generation of the show bookings can be sent to the theatre management on a weekly/monthly basis which will be helpful to improve their business.
- An automatic report generation of the show bookings can be sent to the theatre management on a weekly/monthly basis which will be helpful to improve their business.
- An automatic report generation of the show bookings can be sent to the theatre management on a weekly/monthly basis which will be helpful to improve their business.

In regard with the covid situation we are dealing with, it is always necessary to be cautious and take responsibility to reduce the spread of the disease. With the same thought We want to introduce a follow-up from our side with the customer, where 4-5 days after customer watches the show, he would receive a mail asking whether he has been infected with covid after watching the show or if he has been experiencing any symptoms lately., if yes then a warning mail would be issued to customers who have been part of that show saying that it is better if they get themselves checked up if they are experiencing any symptoms.

While booking a ticket from the customer's end, following questions would be asked to the customer,

- Customer is fine with taking the temperature check before entering the theatre and if the temperature is more than the average temperature, he would not be allowed to enter the theatre even if he has a valid ticket
- Customer should wear the mask at all times in the theatre
- Customers are allowed to sit only in the place they have booked and not in any seats which are available but blocked due to covid situation

They can book a ticket only if they are willing to abide by the above-mentioned clause.