MOVIE TICKET BOOKING SYSTEM WITH SOCIAL DISTANCING

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

Table of Contents

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.

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.

- 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

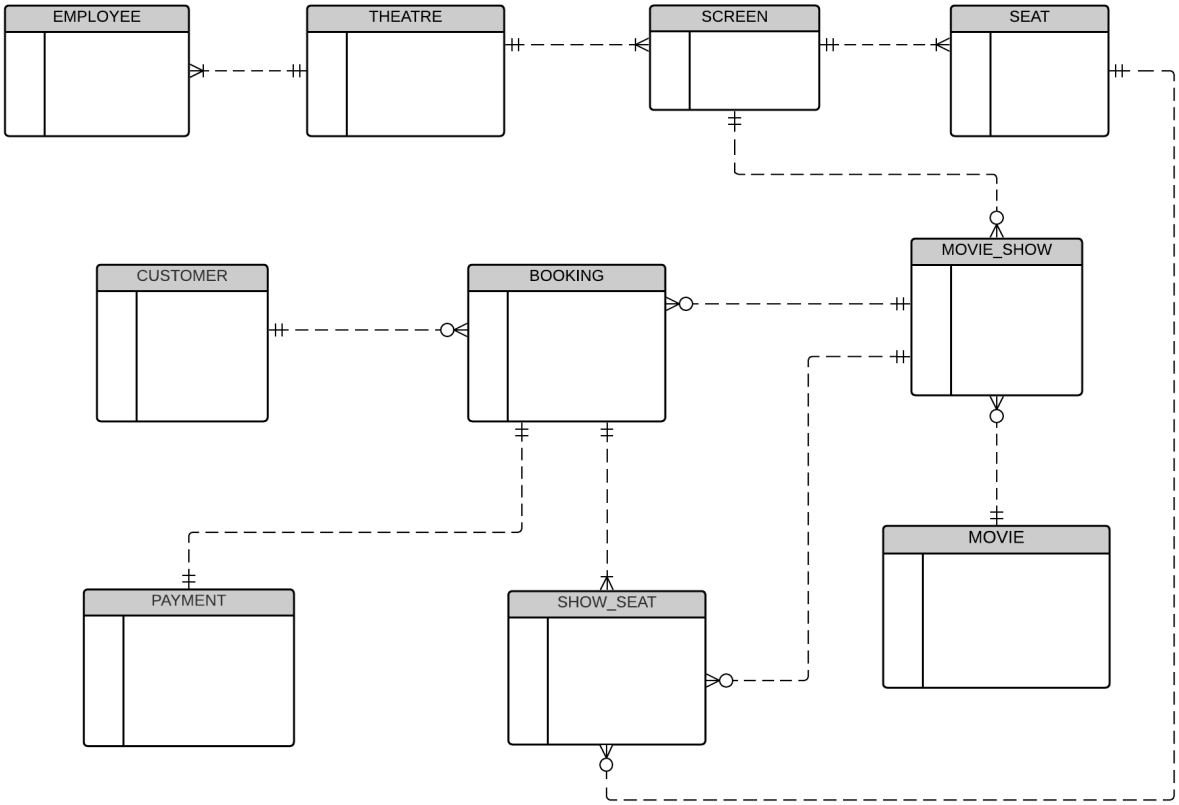
Relationship

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.

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.

Figure: 4.1: Logical ER-Diagram

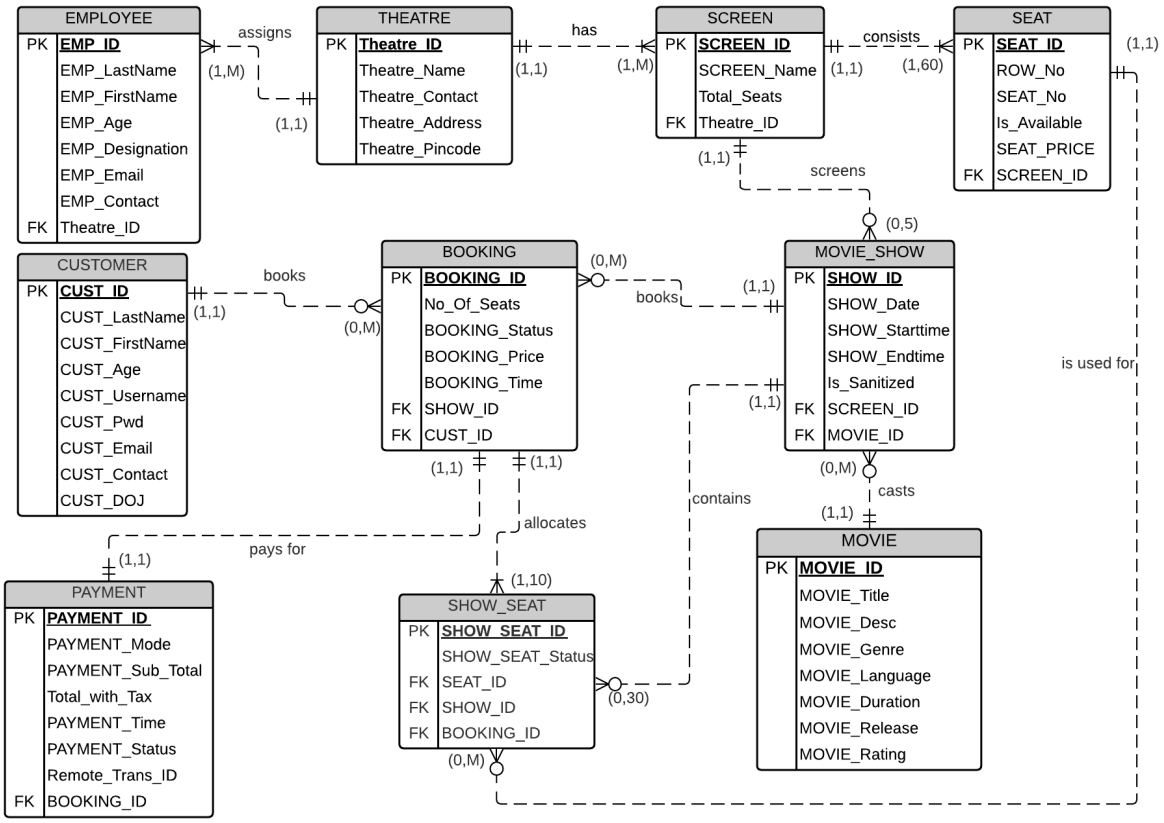
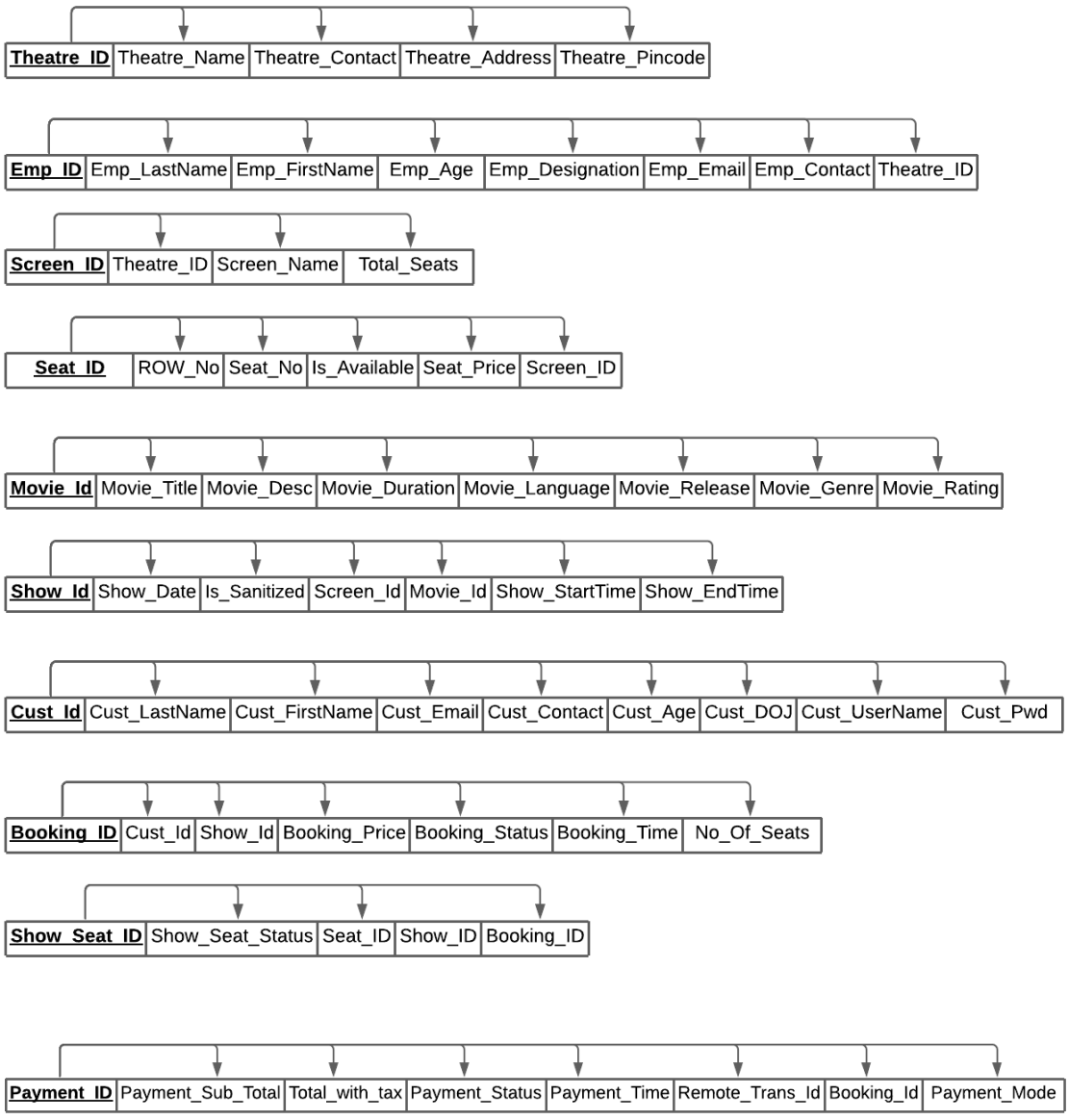


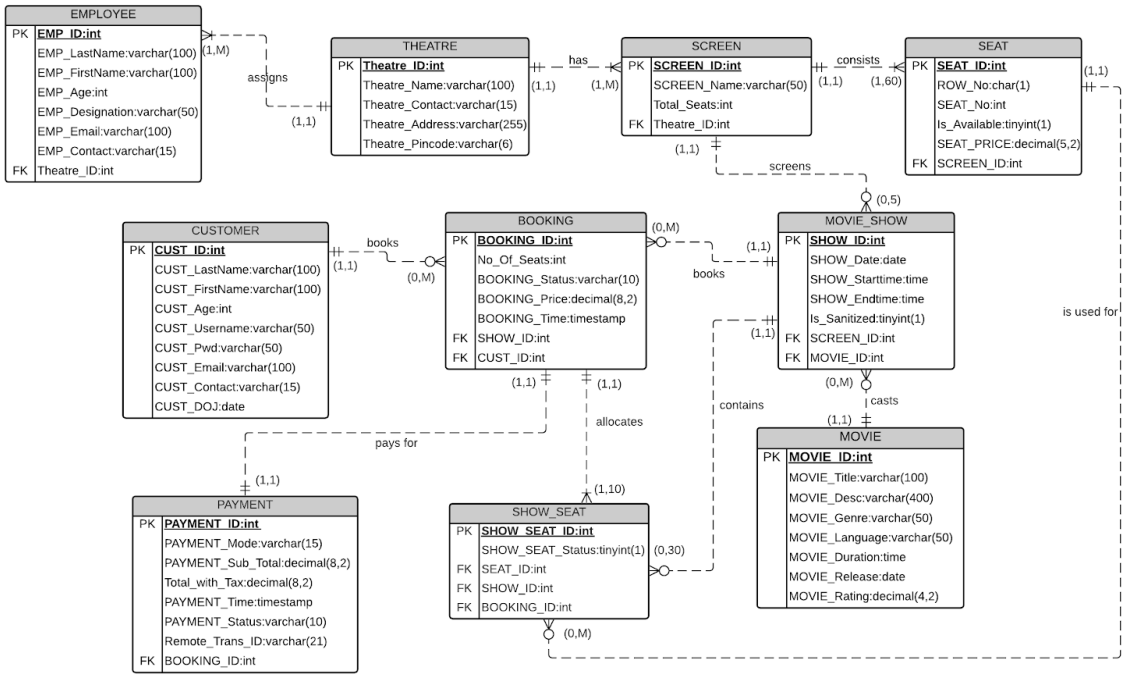
Figure: 4.2: Normalization process



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;

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';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Grants for admin@localhost
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`

Result 17 × Read Only

Output

Action Output

#	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 form 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

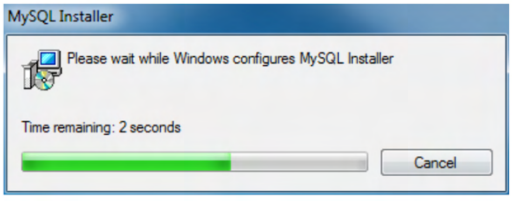
Software Requirements

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

Implementation & Loading

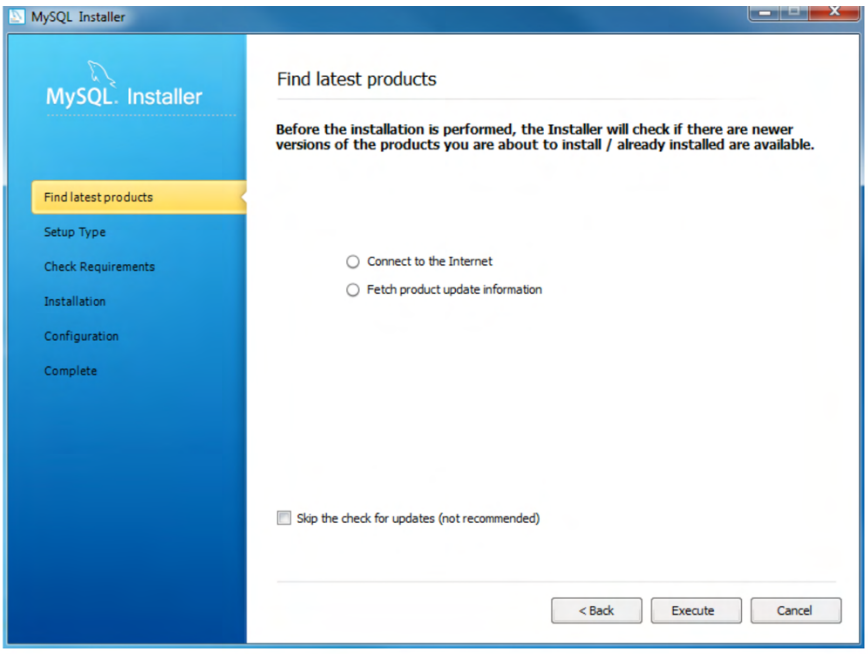
DBMS Installation

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

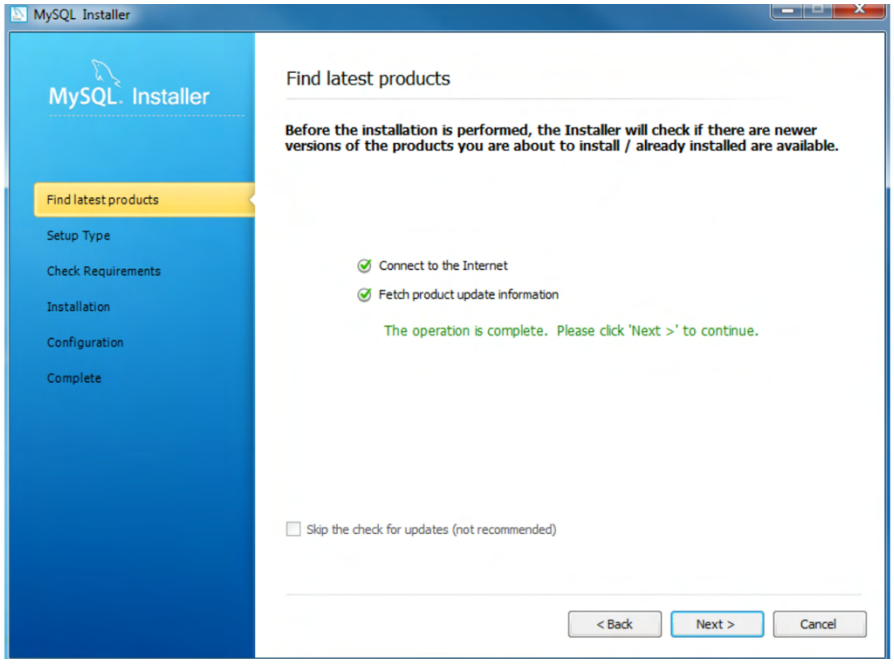


Install MySQL Step 1: Windows configures MySQL Installer

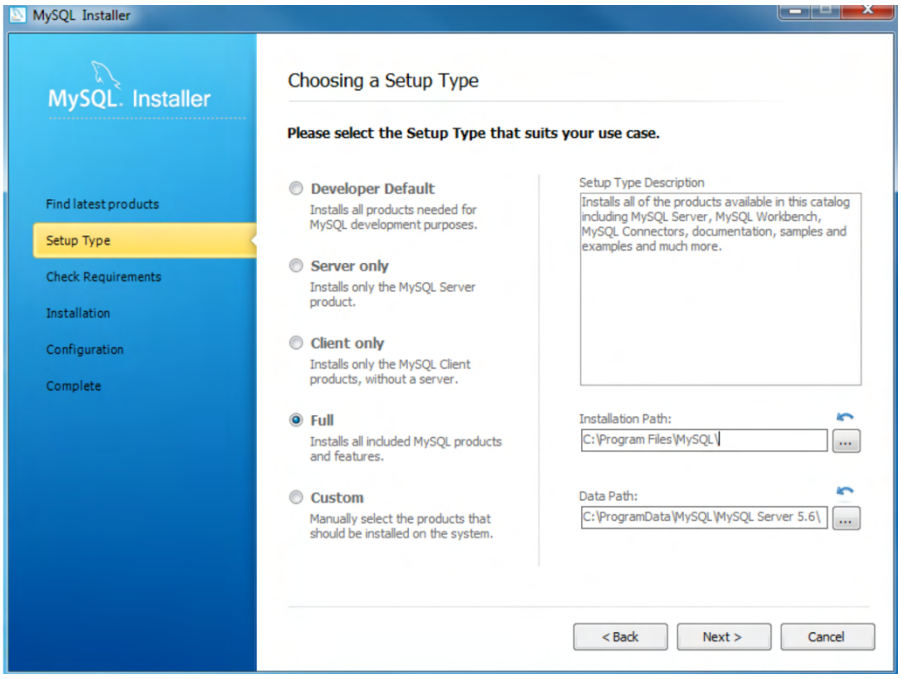
Install MySQL Step 2 – Welcome Screen: A welcome screen provides several options. Choose the first option: Install MySQL Products



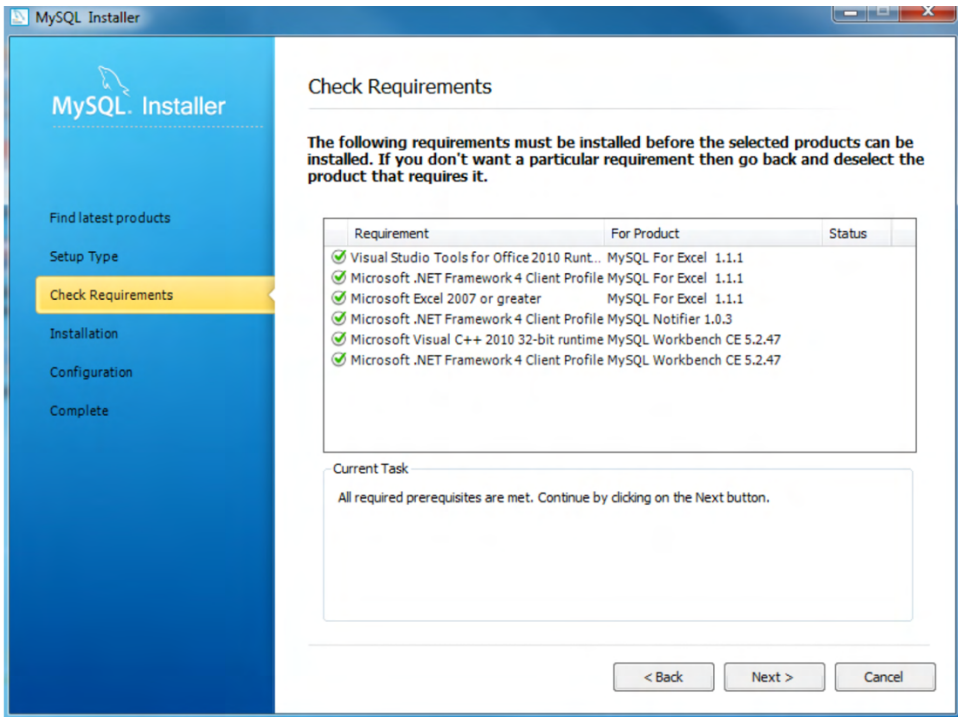
Install MySQL Step 3 – Download the latest MySQL products: MySQL installer checks and downloads the latest MySQL products including MySQL server, MySQL Workbench, etc.



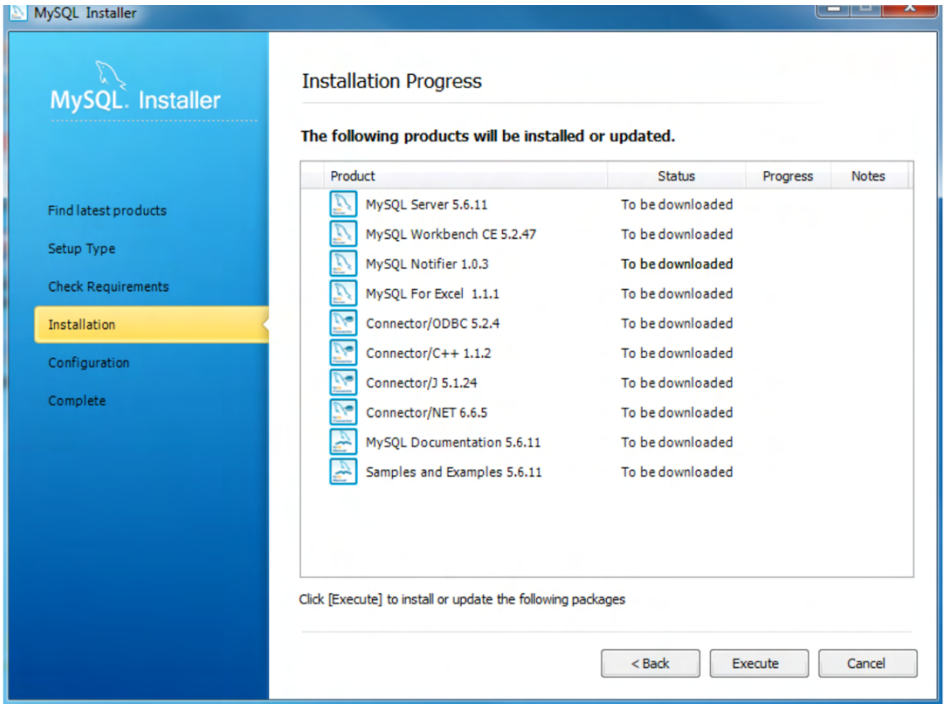
Install MySQL Step 4: Click the Next button to continue



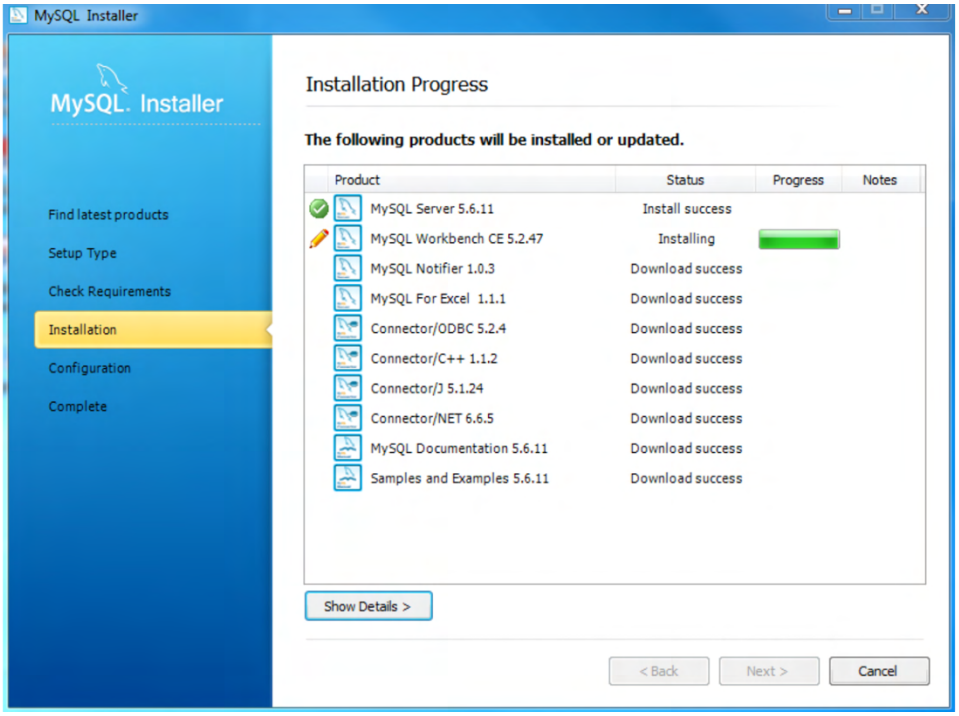
Install MySQL Step 5 – Choosing a Setup Type: there are several setup types available. Choose the Full option to install all MySQL products and features.



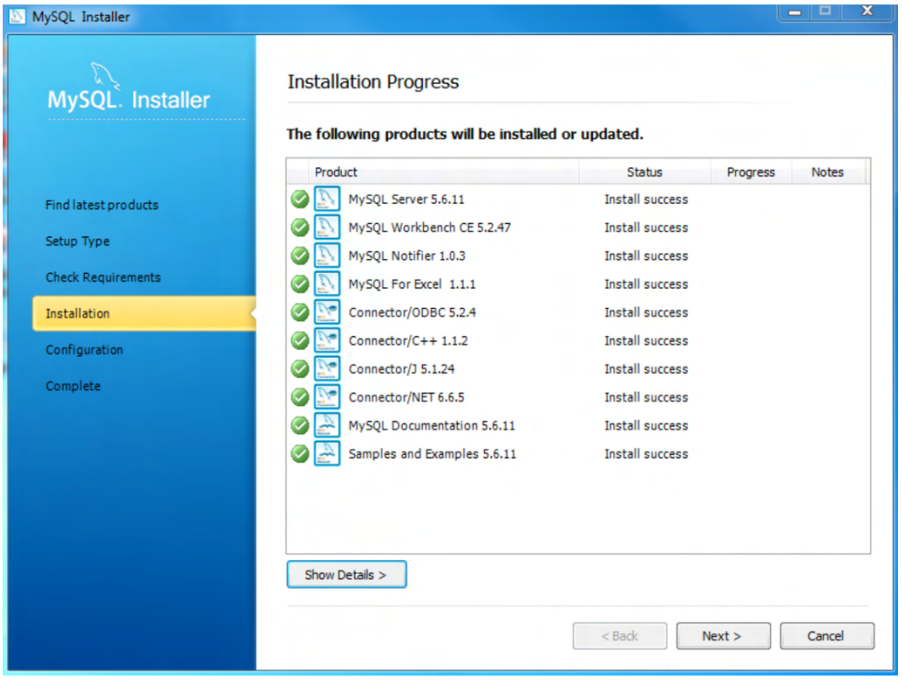
Install MySQL Step 6 – Checking Requirements



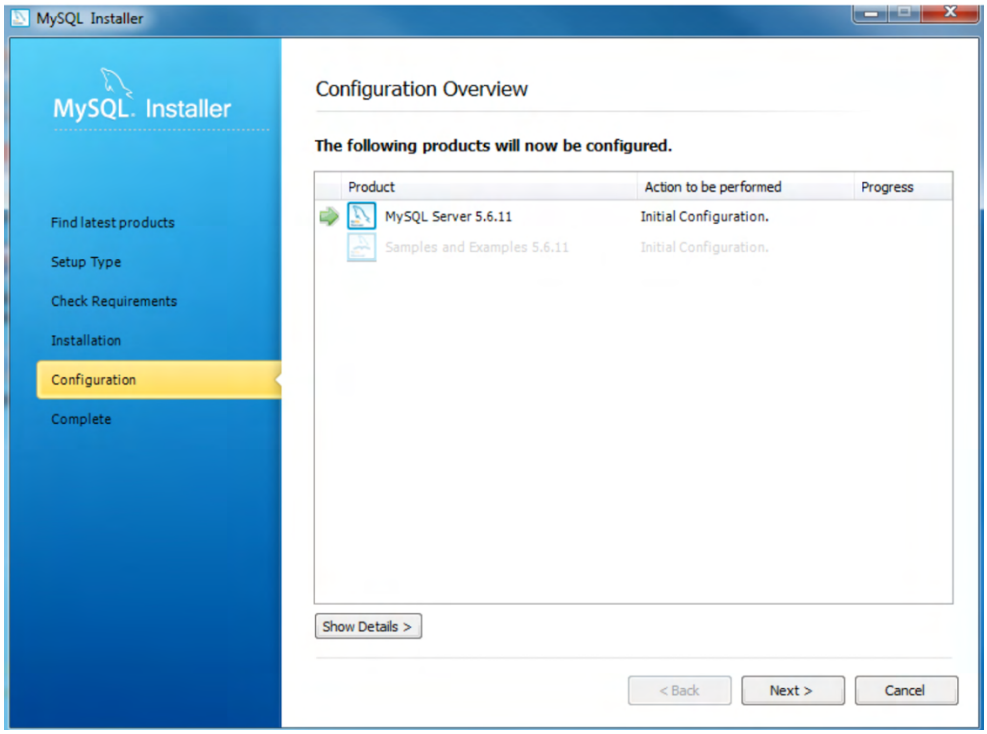
Install MySQL Step 7 – Installation Progress: MySQL Installer downloads all selected products. It will take a while, depending on which products you selected and the speed of your internet connection.



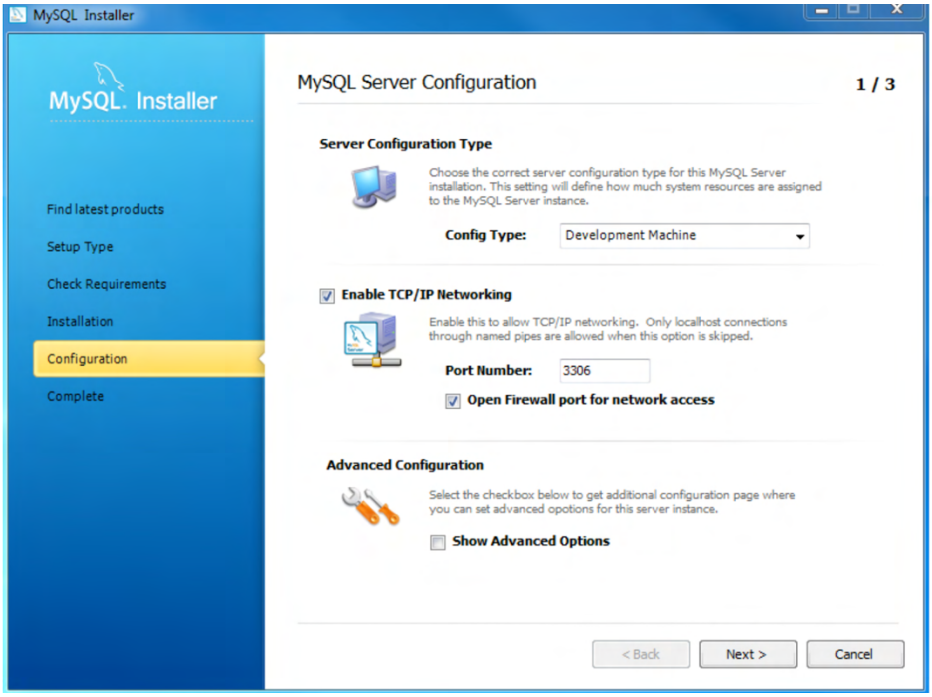
Install MySQL Step 7 – Installation Progress: downloading Products in progress.



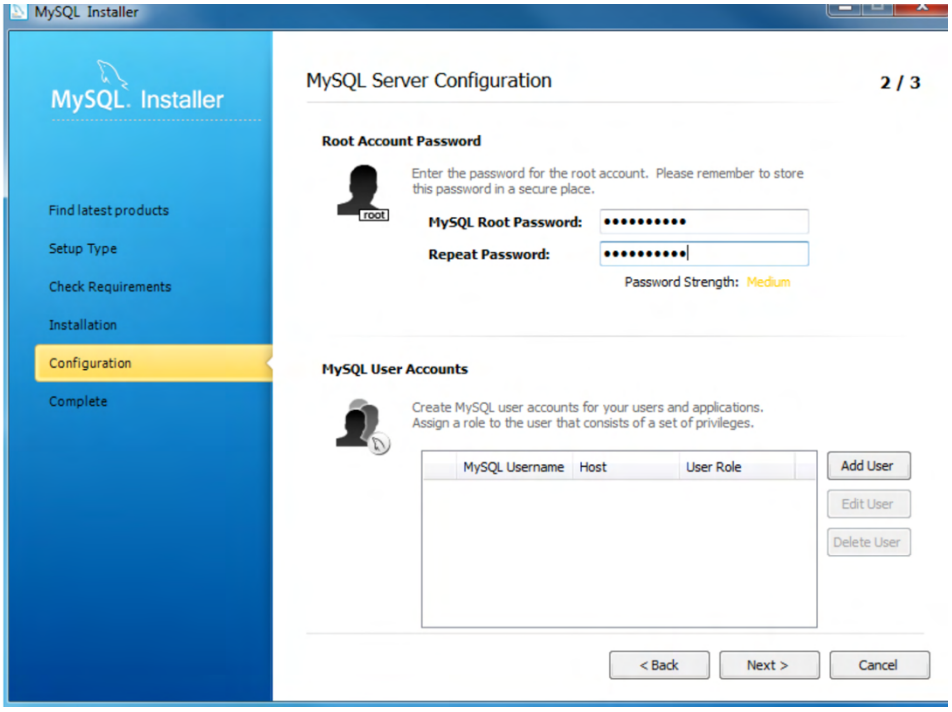
Install MySQL Step 7 – Installation Progress: Complete Downloading. Click the **Next** button to continue…



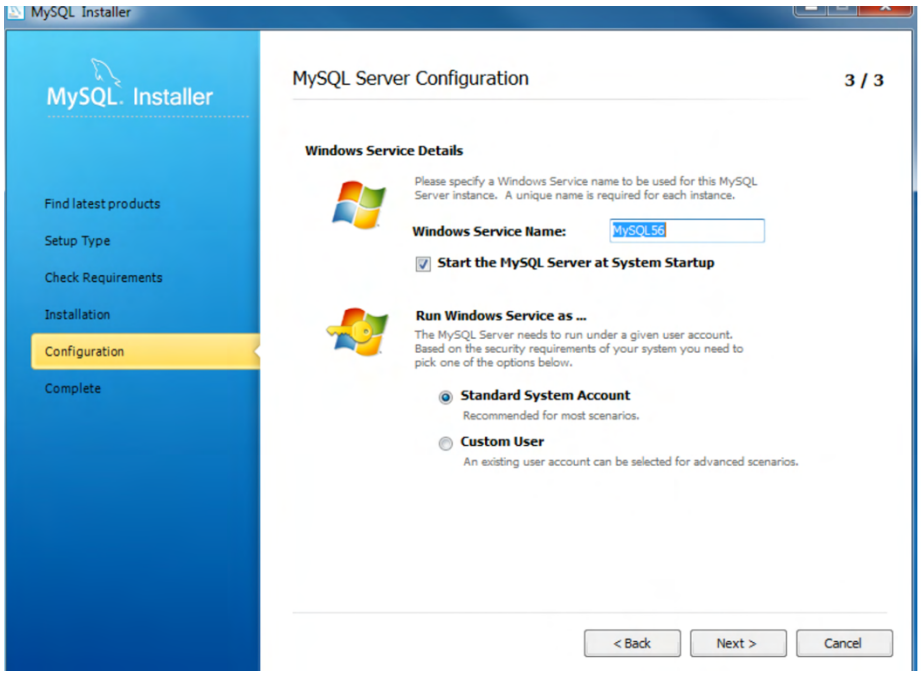
Install MySQL Step 8 – Configuration Overview. Click the Next button to configure MySQL Database Server



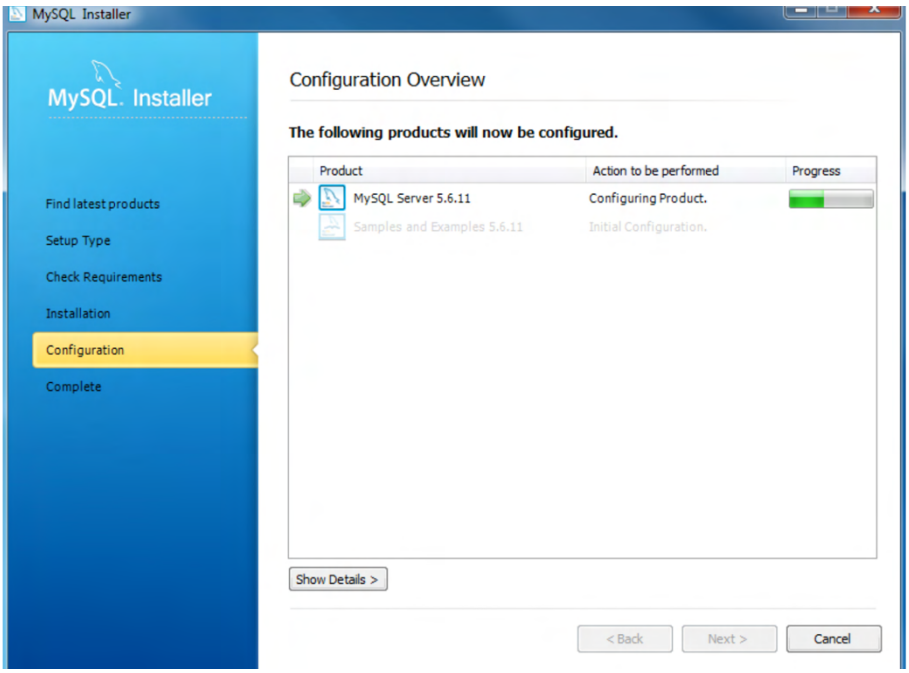
Install MySQL Step 8.1 – MySQL Server Configuration: choose Config Type and MySQL port (3006 by default) and click Next button to continue.



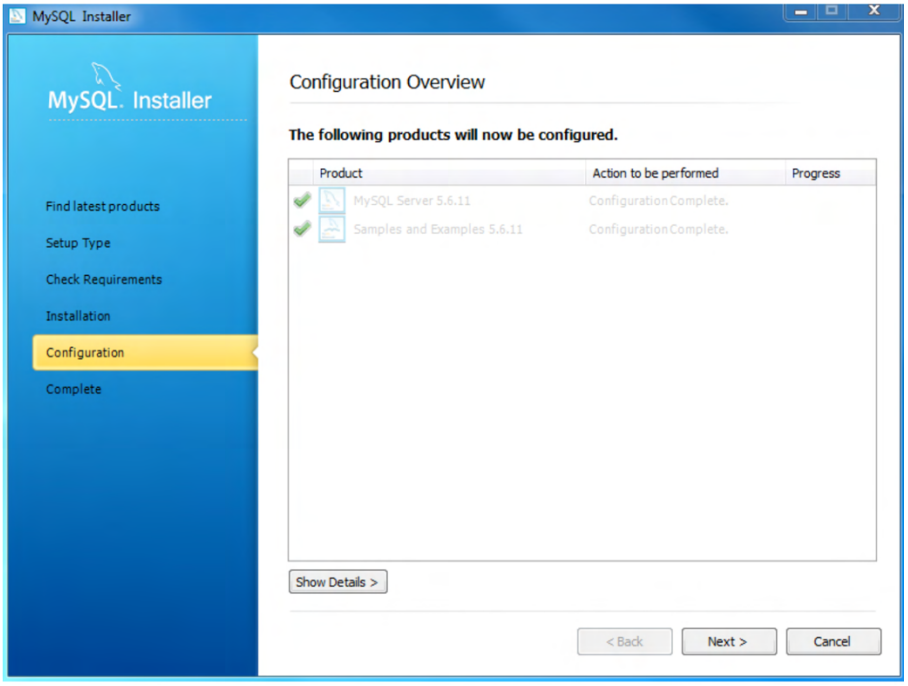
Install MySQL Step 8.1 – MySQL Server Configuration: choose a password for the root account. Please note the password download and keep it securely if you are installing MySQL database server on a production server. If you want to add a more MySQL user, you can do it in this step



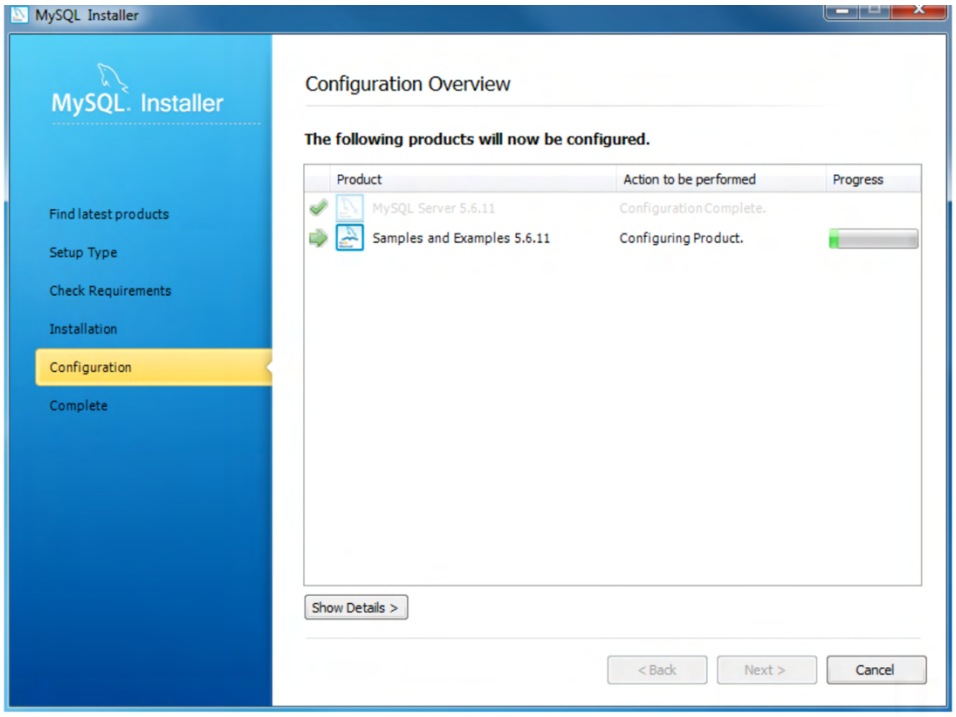
Install MySQL Step 8.1 – MySQL Server Configuration: choose Windows service details including Windows Service Name and account type, then click Next button to continue



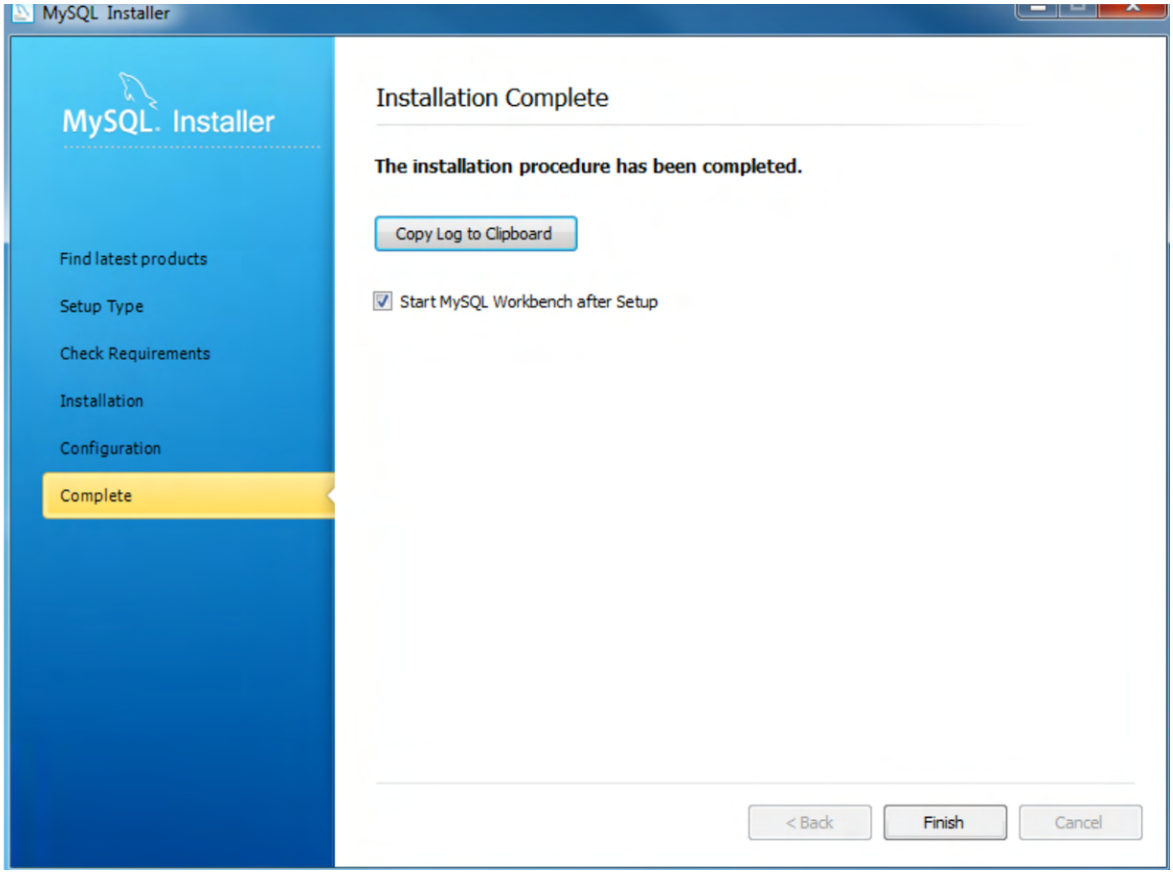
Install MySQL Step 8.1 – MySQL Server Configuration – In Progress: MySQL Installer is configuring MySQL database server. Wait until it is done and click the Next button to continue



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.



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;
```

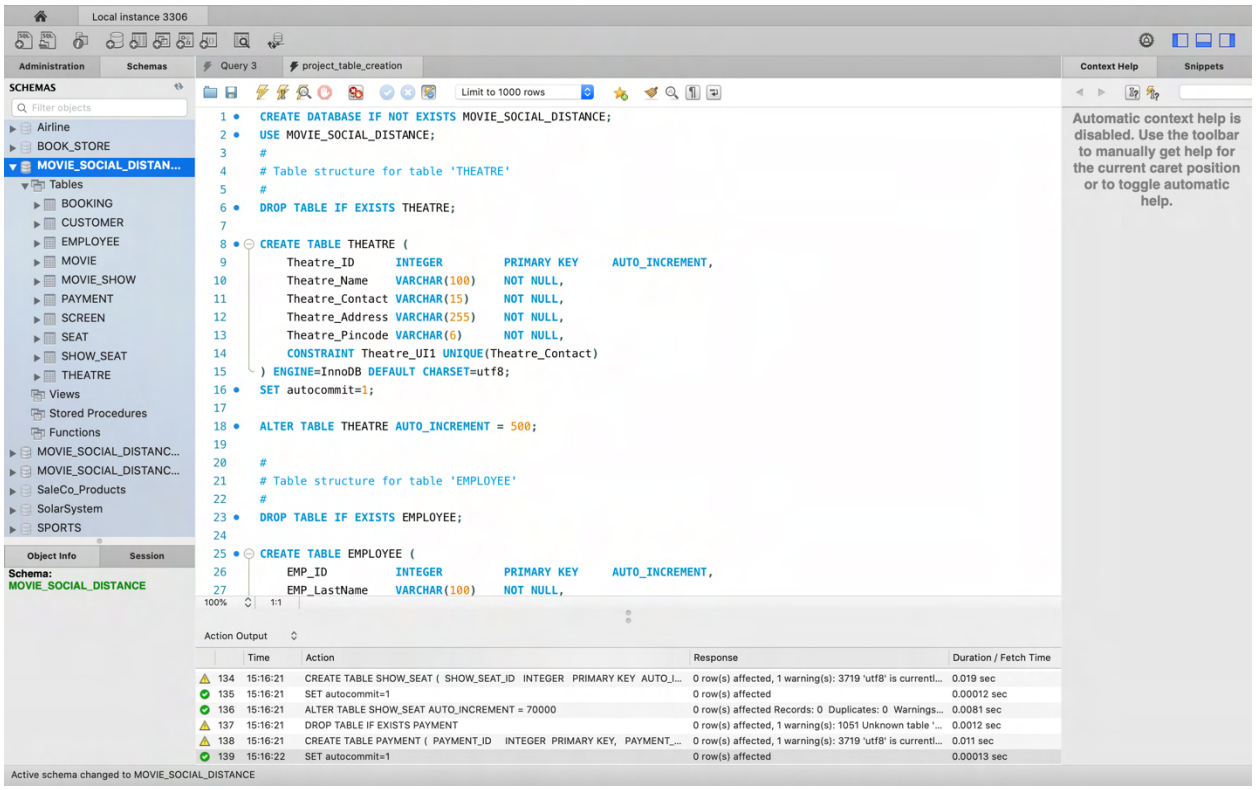
```
SET autocommit=1;

#
# 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_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,
```

```
  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,
```

```
        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;
```

Below is the screenshot of the above code after execution.



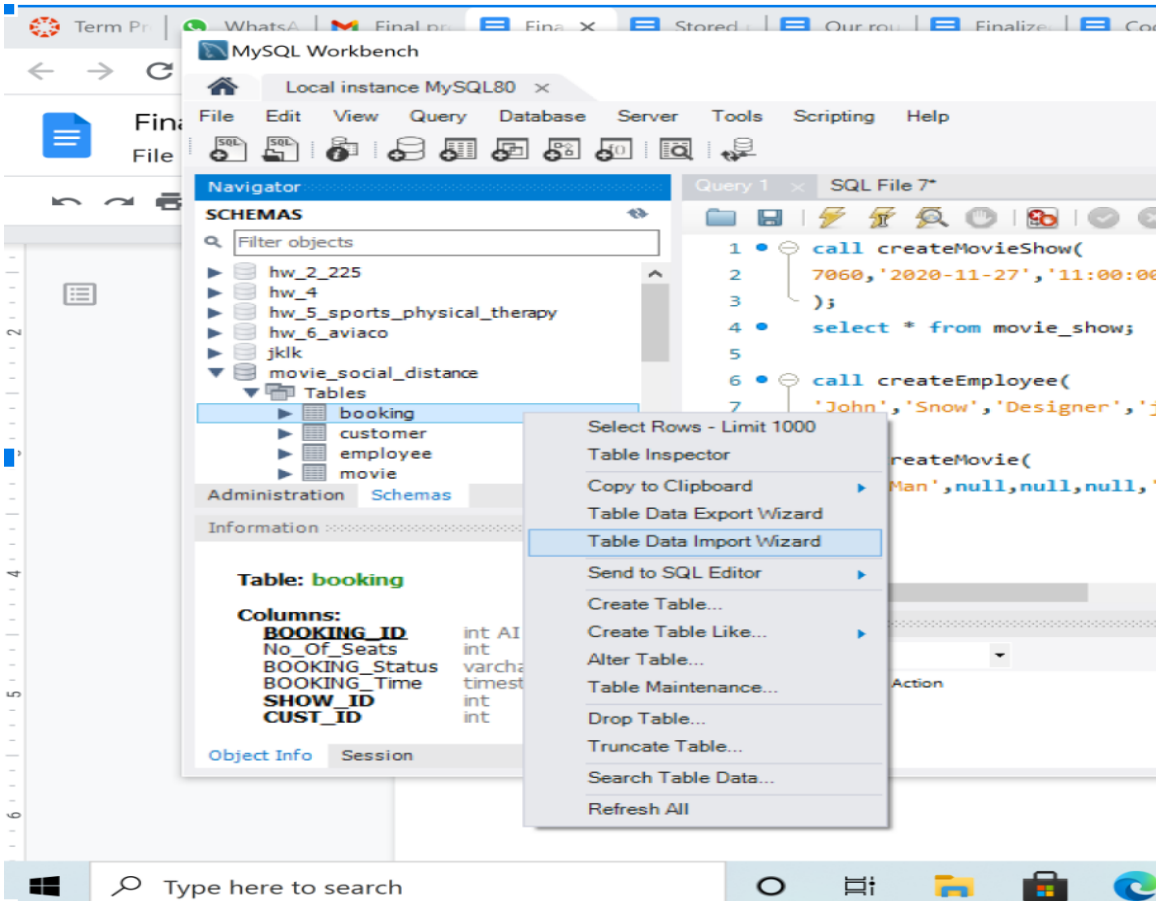
Data Loading & Conversion

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.

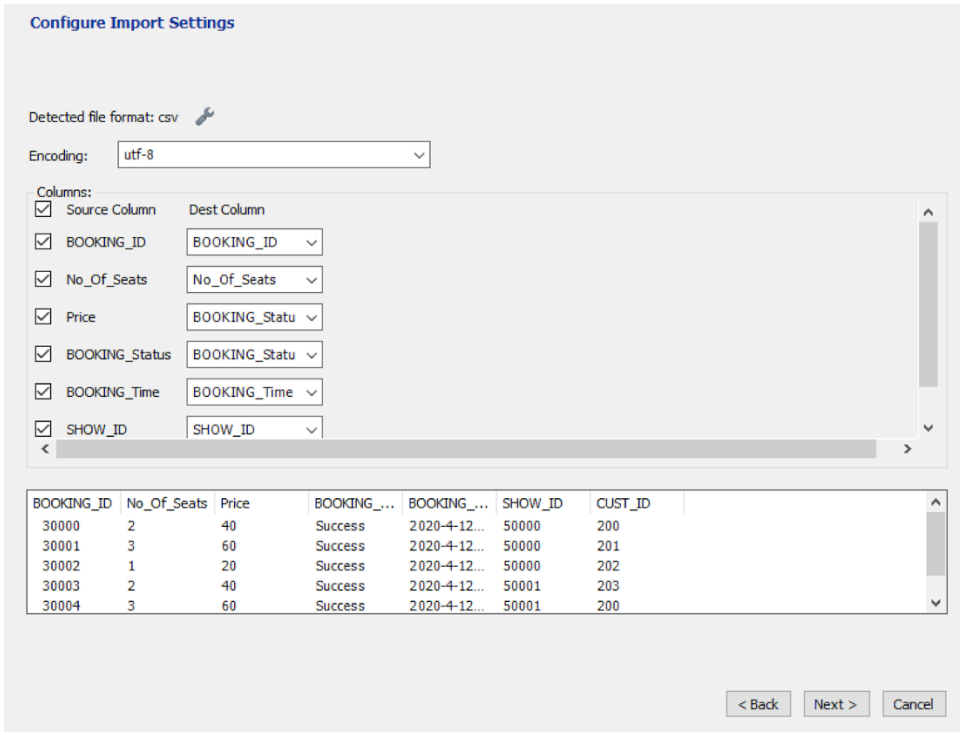
CSV File:

1	BOOKING	No_Of_Se	Price	BOOKING_	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

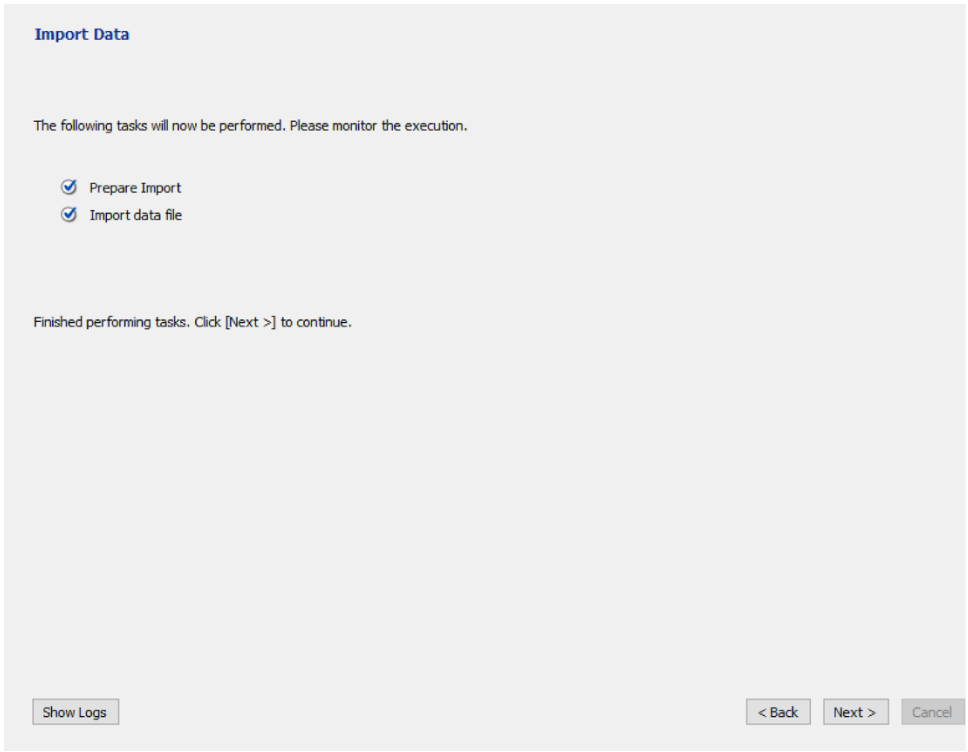
This wizard can be accessed from the table's context menu by right-clicking on a table and choose **Table Data Import Wizard**, as the below figure shows and choose the path from which the csv file has to be imported.



And it maps the columns with table schema as shown



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



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_BYSTATUS	If the 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

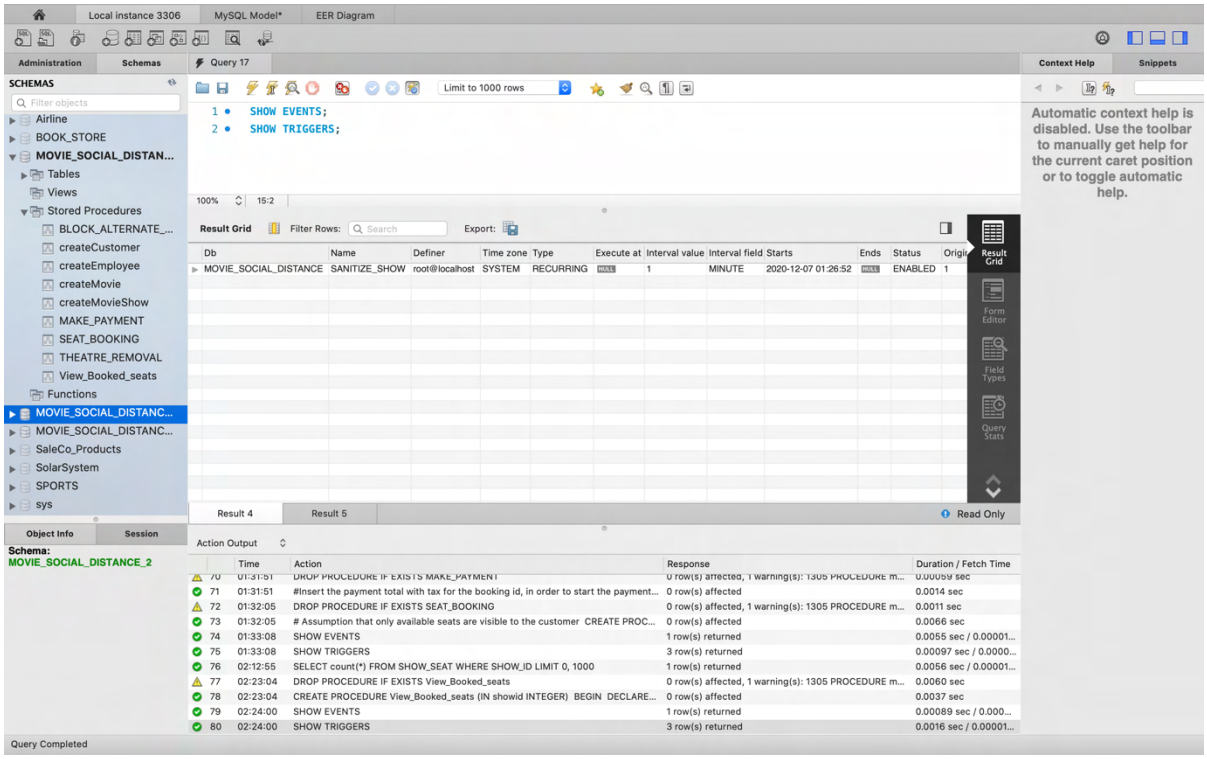
			row number in the SEAT table. The price range is default and set at the time of design.	range based on the row numbers	
v)	Stored Procedure	BLOCK_ALTERNATE_ROWS	Update the Is_Available field in the SEAT table to 0 for alternate rows	The IS_Available field is set to 0 for alternate fields	Attached
vi)	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_REMOVAL	To delete a THEATRE_ID from the THEATRE table, ● Update the movie_id to NULL in the MOVIE_SHOW table ● Delete the entries in the SHOW_SEAT table which	● The movie_id is updated to NULL in the MOVIE_SHOW table ● The SHOW_SEAT entries for the theatre_id is getting deleted.	Attached

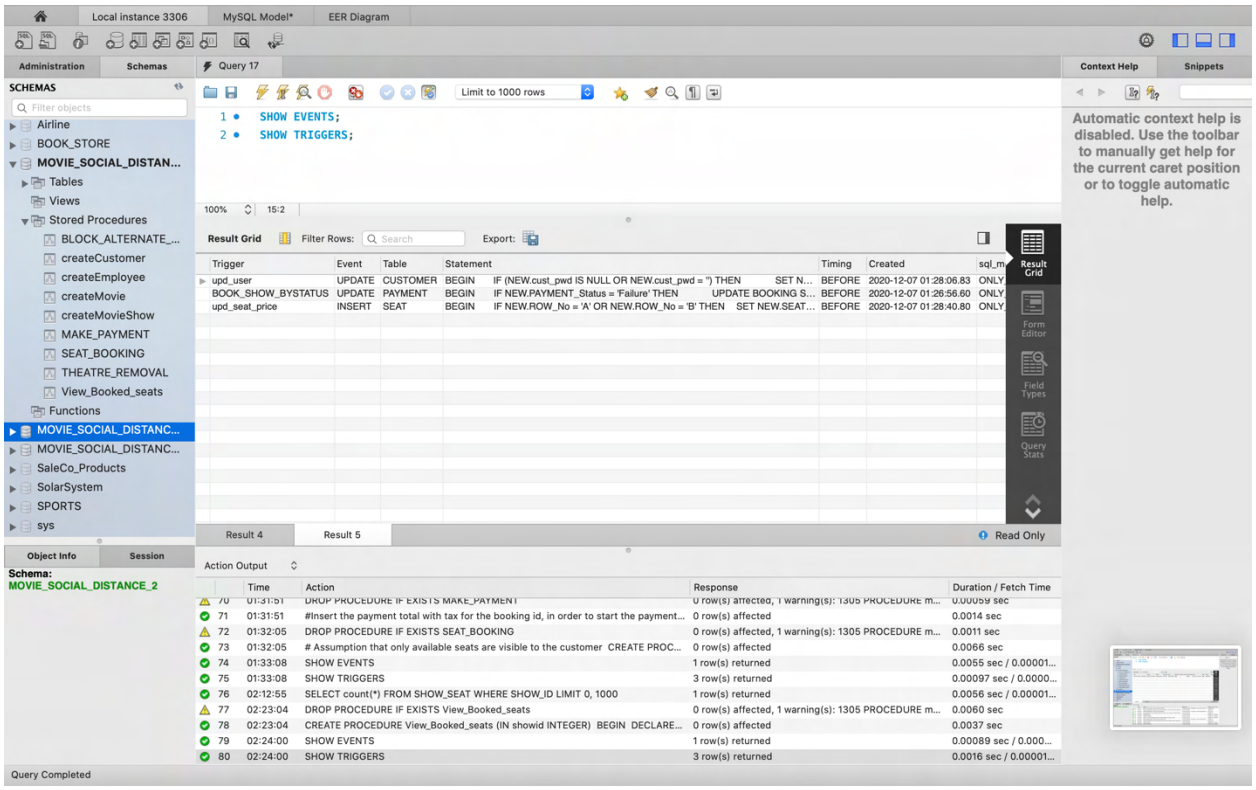
			are linked to the show_id that are running on the screen_id in the given theatre_id ● Delete the entries in the PAYMENT table that are linked to the theatre_id ● Delete the entries in the BOOKING table mapped to the given theatre_id ● Delete the entry from MOVIE_SHOW that are mapped to the theatre_id 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	● 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_SHOW 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	

			the given theatre id ● 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,total_with_tax,booking_id	The PAYMENT table is updated with the values of payment_subtotal,total_with_tax,booking_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_STATUS as 0	The booking details are entered into BOOKING details and the SHOW_SEAT table is updated with the SHOW_SEAT_STATUS as 0 for the corresponding booking_id	Attached

Screenshots

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

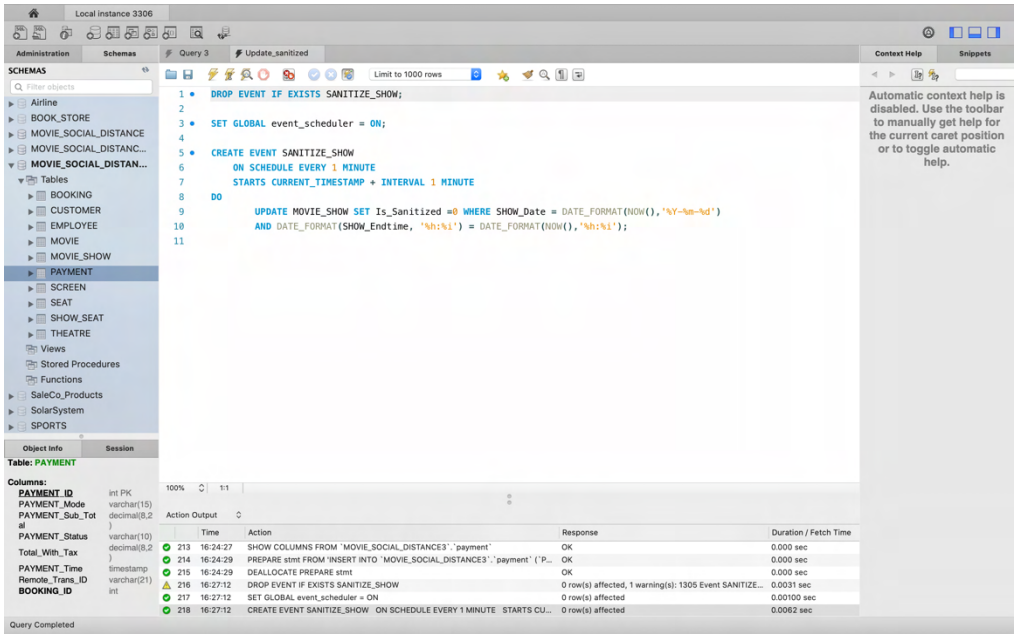




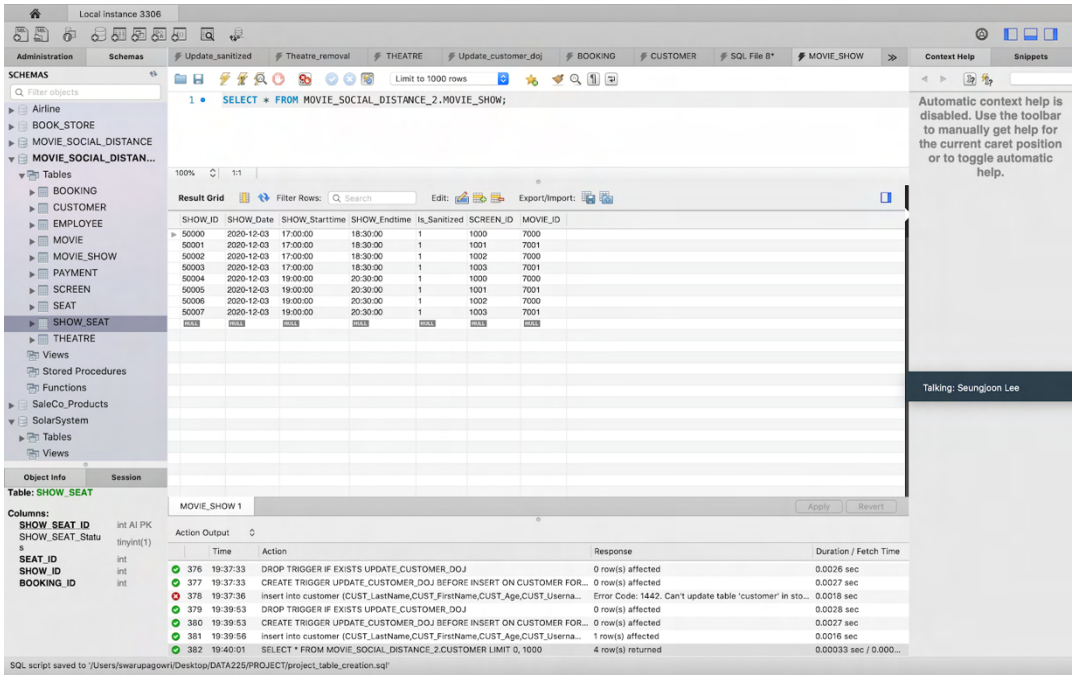
Screenshots of the testing of the above actions.

SANITIZE_Show

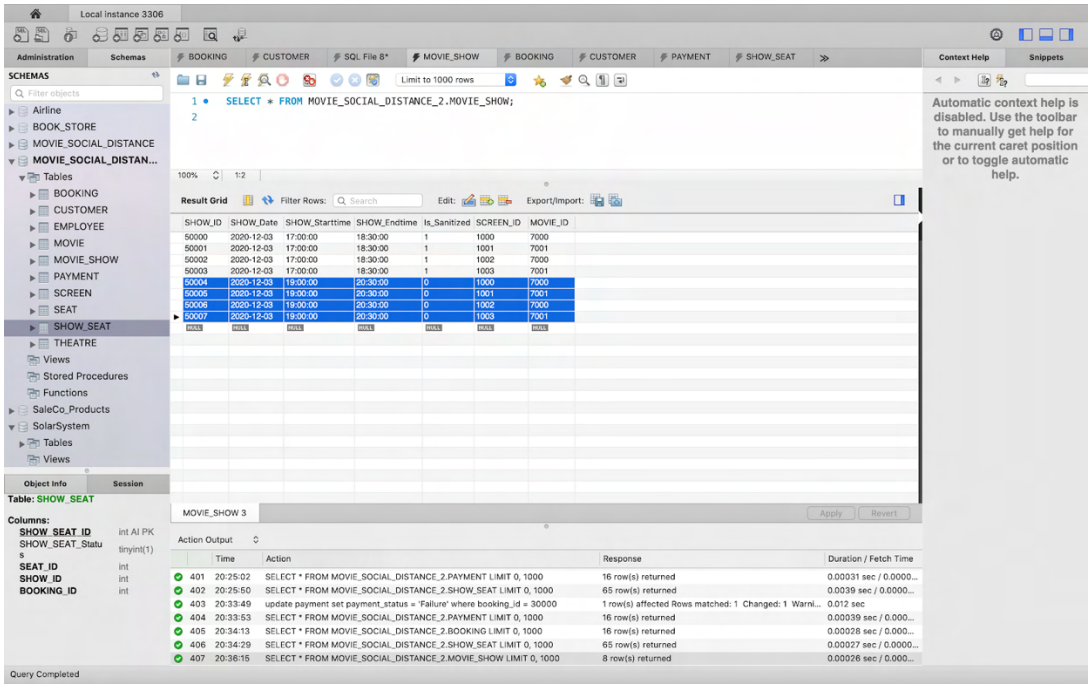
Execution of the event:



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.

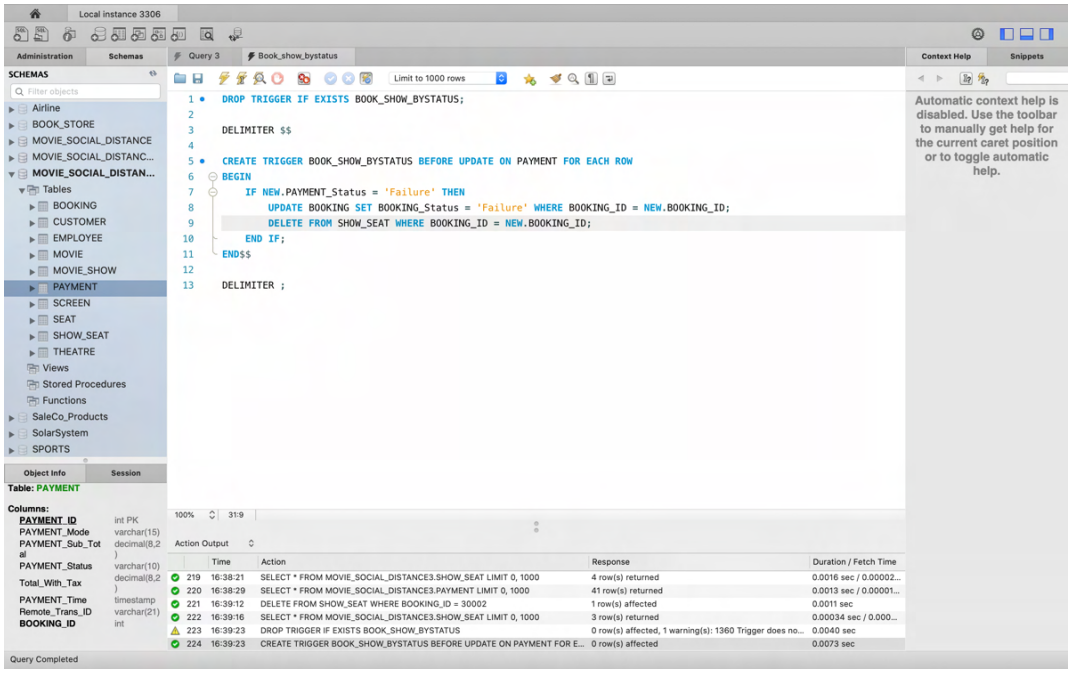


Screen shot taken at local time is 20:30:00 on 2020-12-03. The Is_Sanitized has been updated to 0.

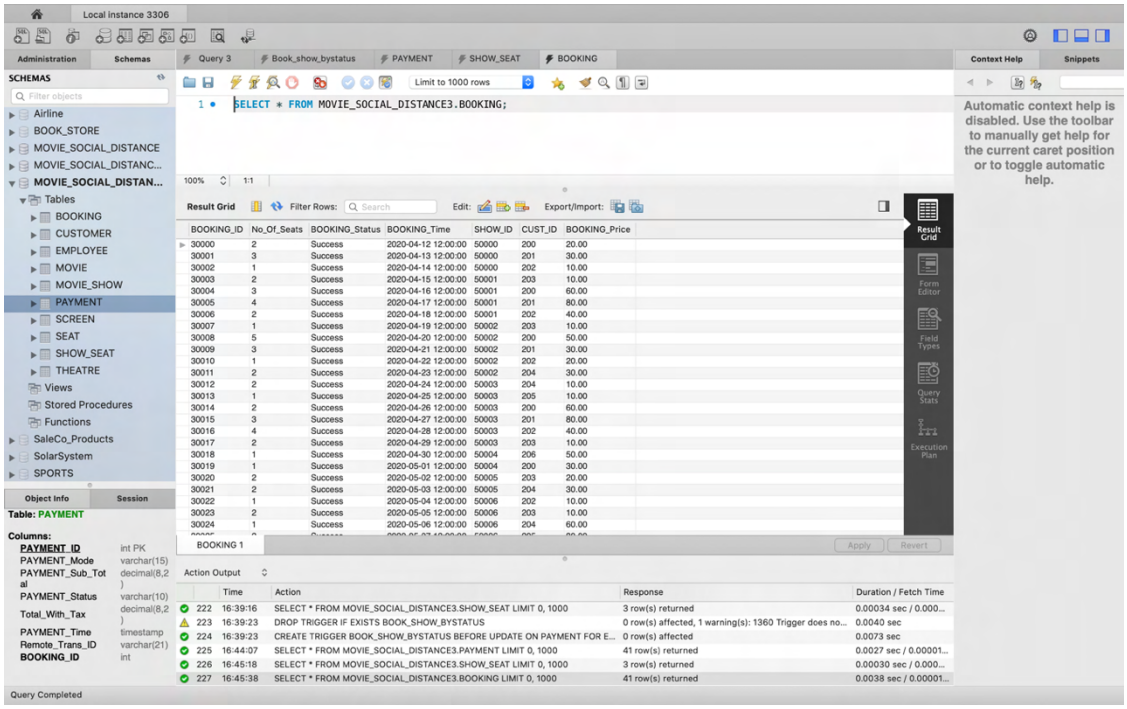


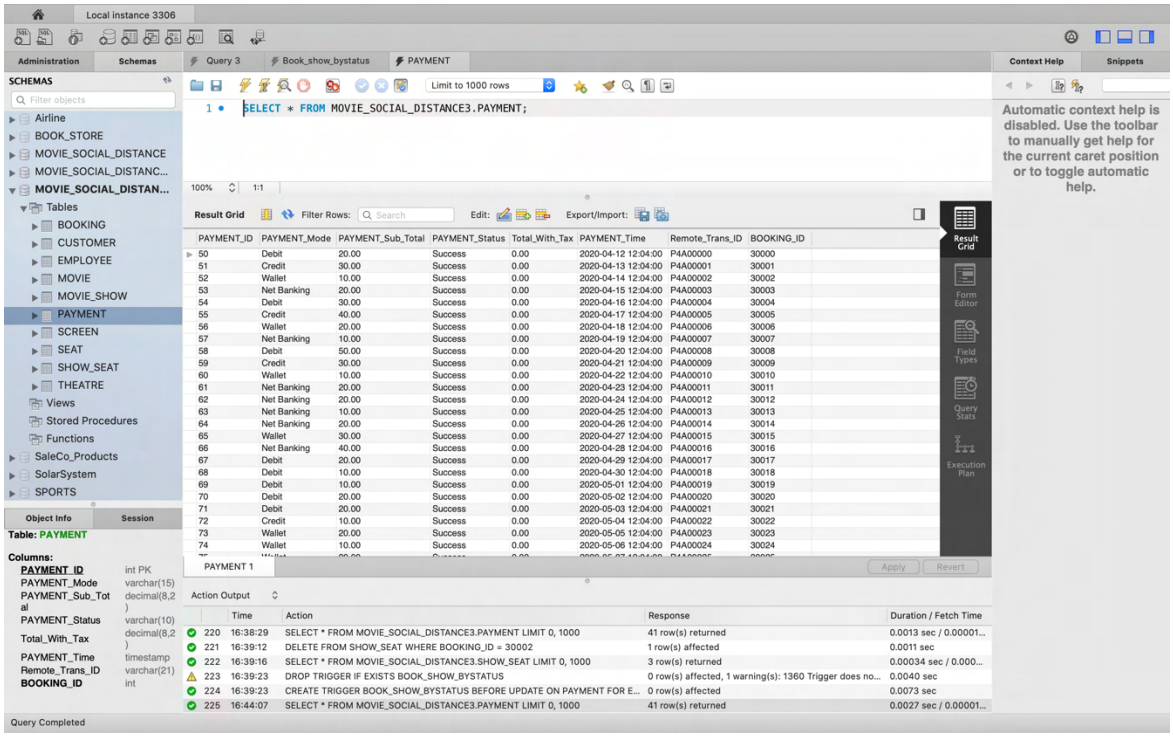
BOOK_SHOW_BYSTATUS

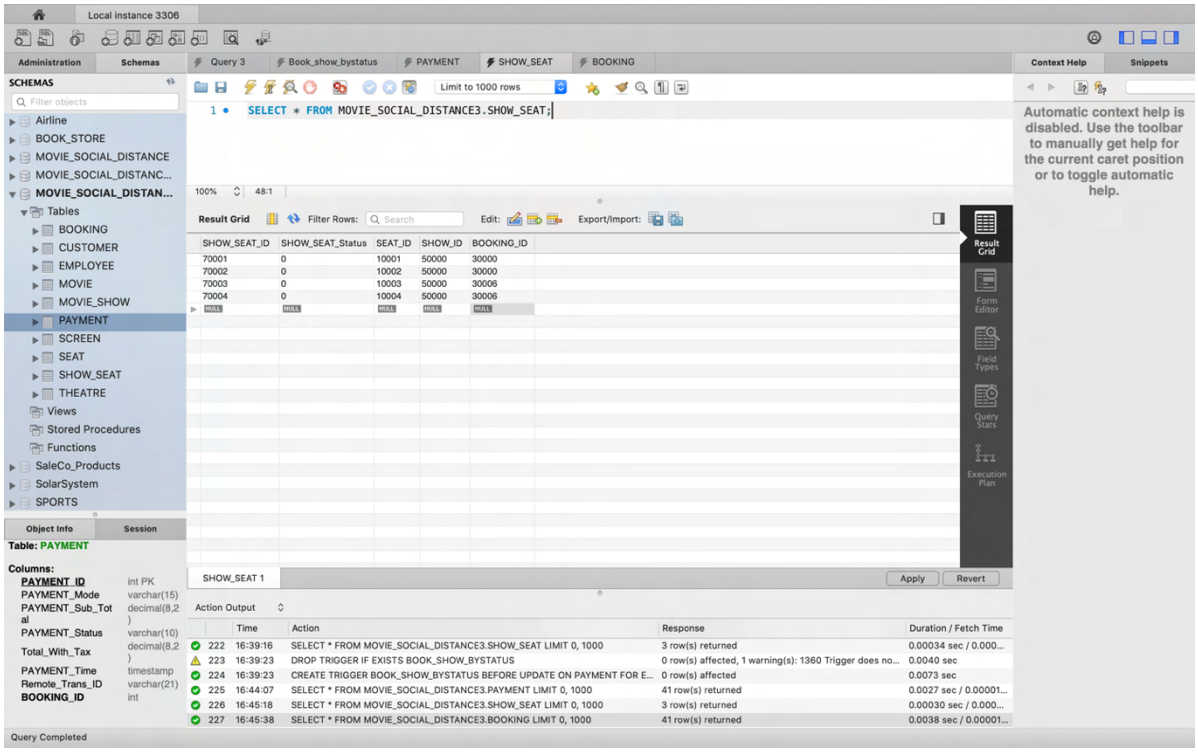
Execution of the trigger:



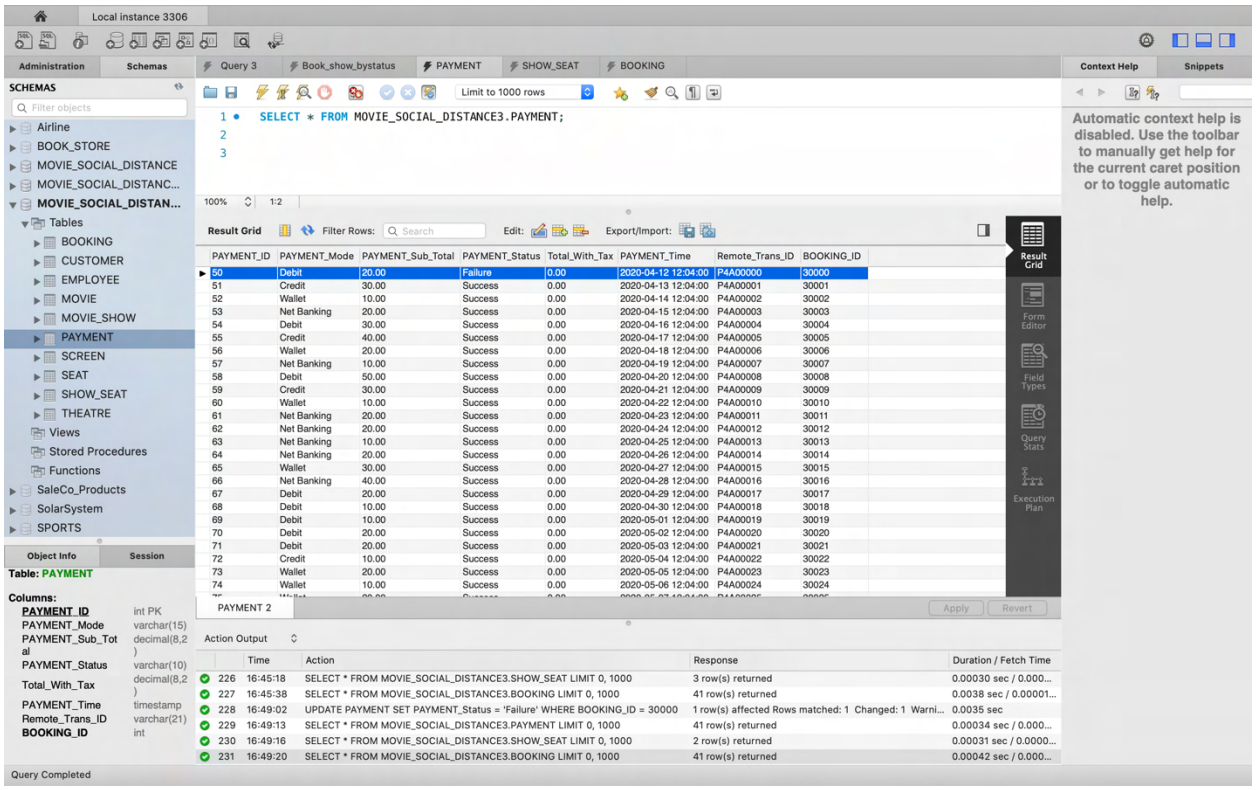
Entries in tables payment, booking and show_seat before a payment failure.

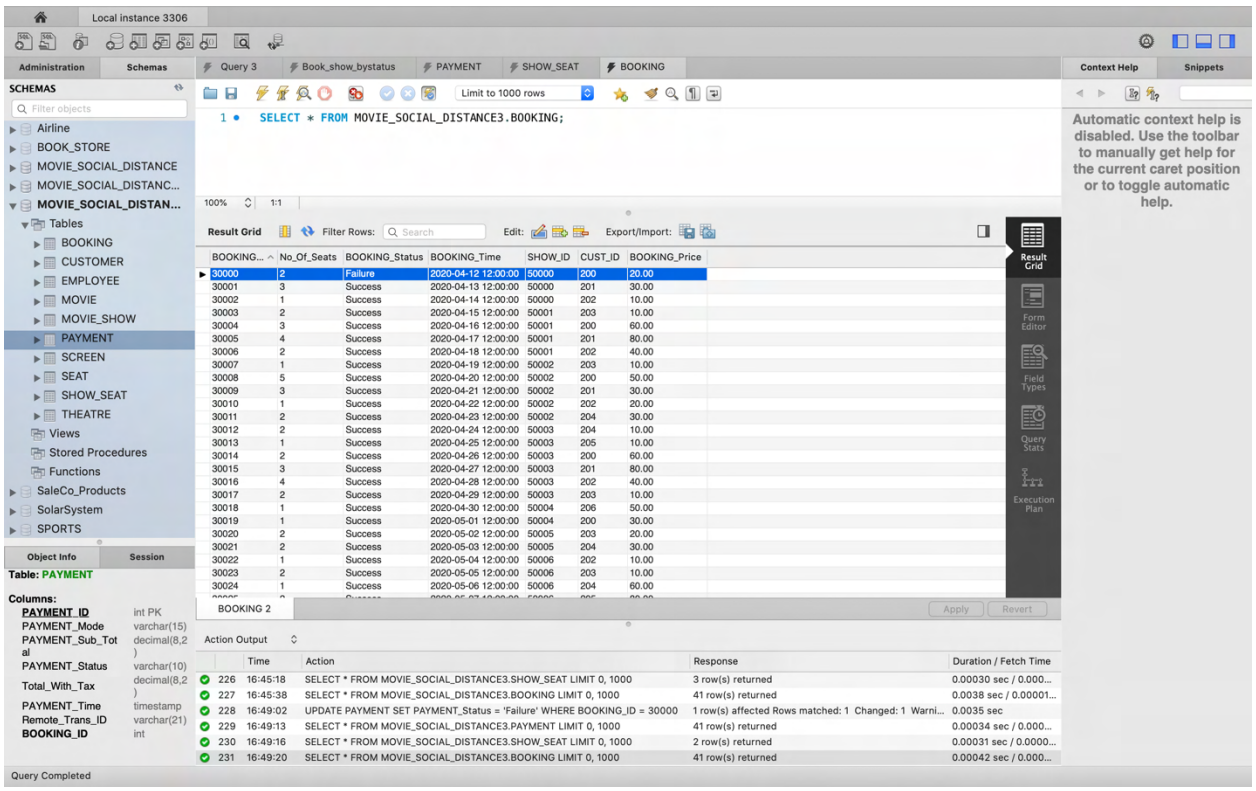


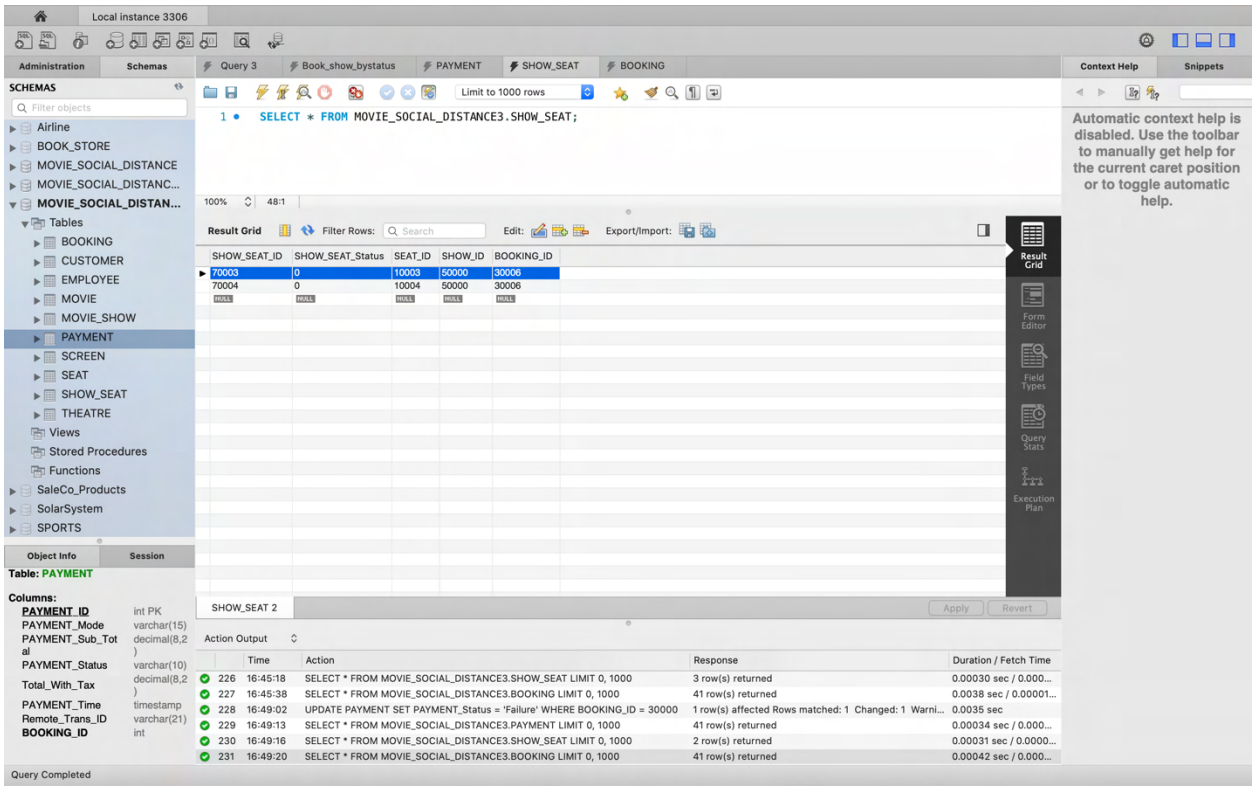




Entries in tables payment, booking and show_seat after a payment failure.

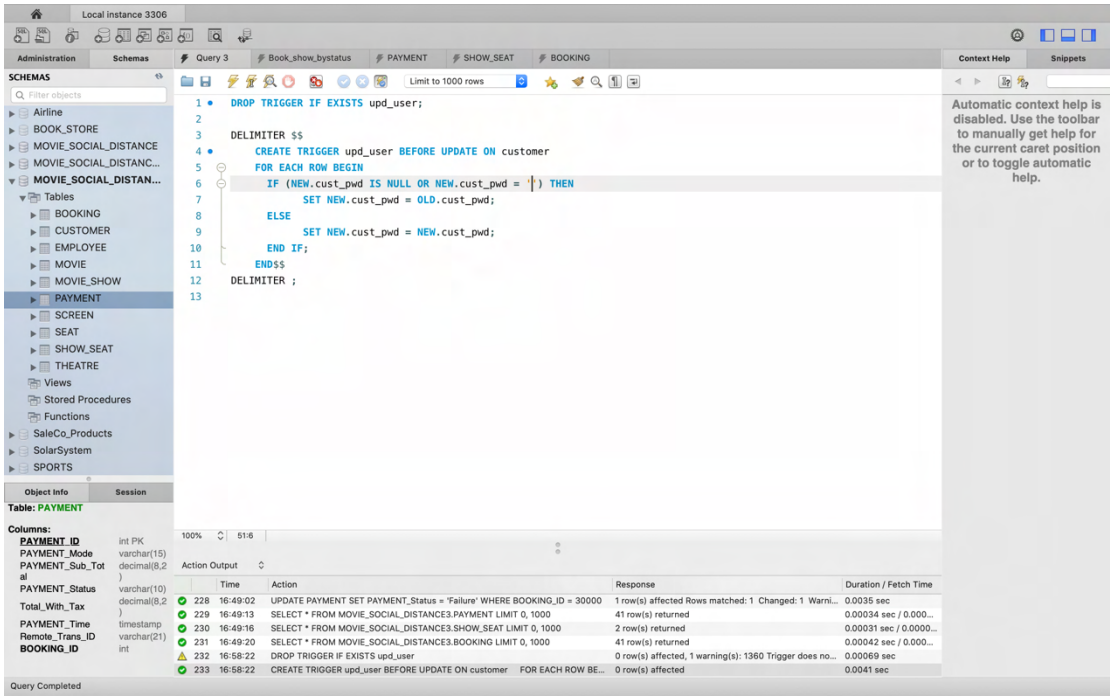

Local instance 3306
SELECT * FROM MOVIE_SOCIAL_DISTANCE3.PAYMENT;
PAYMENT_ID PAYMENT_Mode PAYMENT_Sub_Total PAYMENT_Status Total_With_Tax PAYMENT_Time Remote_Trans_ID BOOKING_ID
50 Debit 20.00 Failure 0.00 2020-04-12 12:04:00 P4A00000 30000
51 Credit 30.00 Success 0.00 2020-04-13 12:04:00 P4A00001 30001
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 Wallet 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
Action Output
Query Completed


Local instance 3306
SELECT * FROM MOVIE_SOCIAL_DISTANCE3.BOOKING;
BOOKING_ID No_Of_Seats BOOKING_Status BOOKING_Time SHOW_ID CUST_ID BOOKING_Price
30000 2 Failure 2020-04-12 12:00:00 50000 200 20.00
30001 3 Success 2020-04-13 12:00:00 50000 201 30.00
30002 1 Success 2020-04-14 12:00:00 50000 202 10.00
30003 2 Success 2020-04-15 12:00:00 50001 203 10.00
30004 3 Success 2020-04-16 12:00:00 50001 200 60.00
30005 4 Success 2020-04-17 12:00:00 50001 201 80.00
30006 2 Success 2020-04-18 12:00:00 50001 202 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 2020-04-22 12:00:00 50002 202 20.00
30011 2 Success 2020-04-23 12:00:00 50002 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 206 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 2020-05-03 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
Action Output
Query Completed

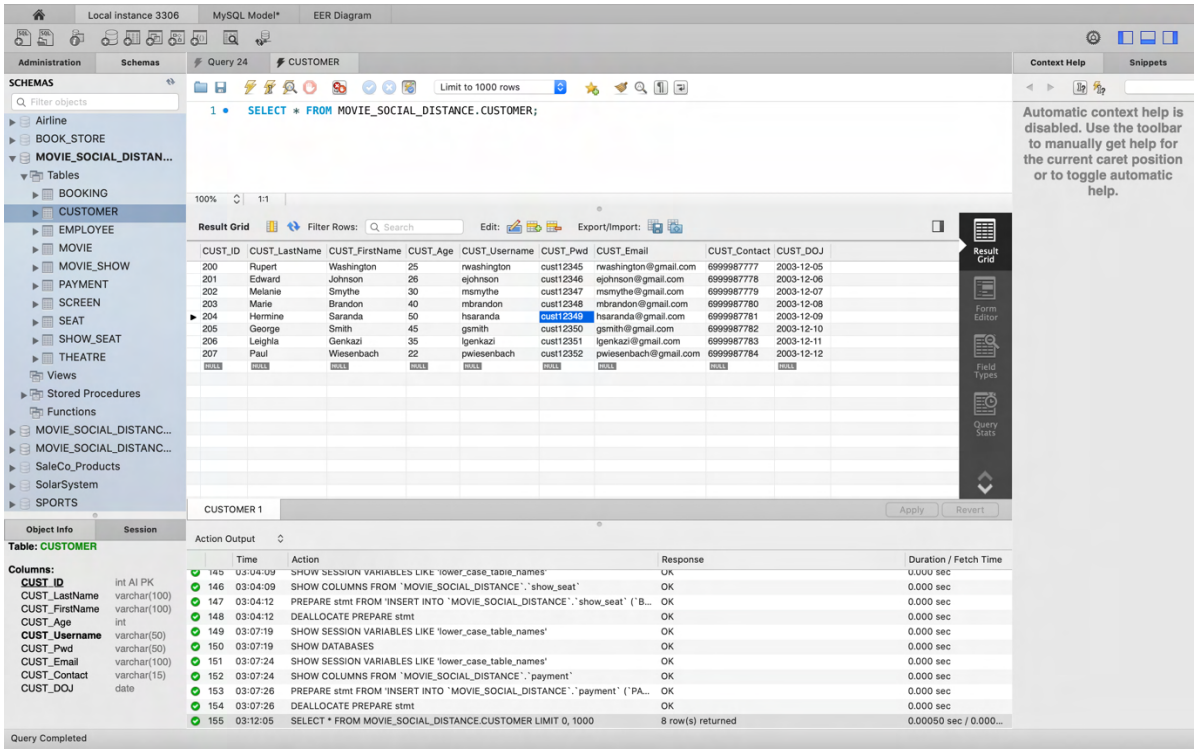


upd_user

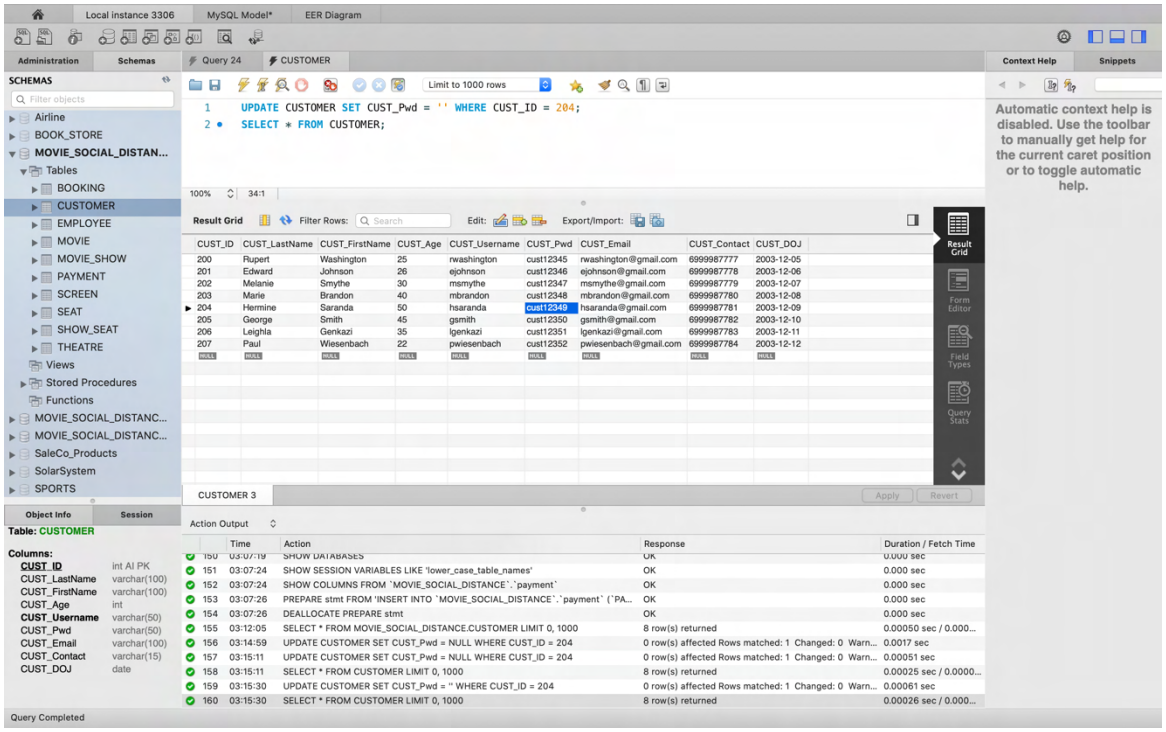
Execution of the trigger:



Before update in the CUSTOMER table

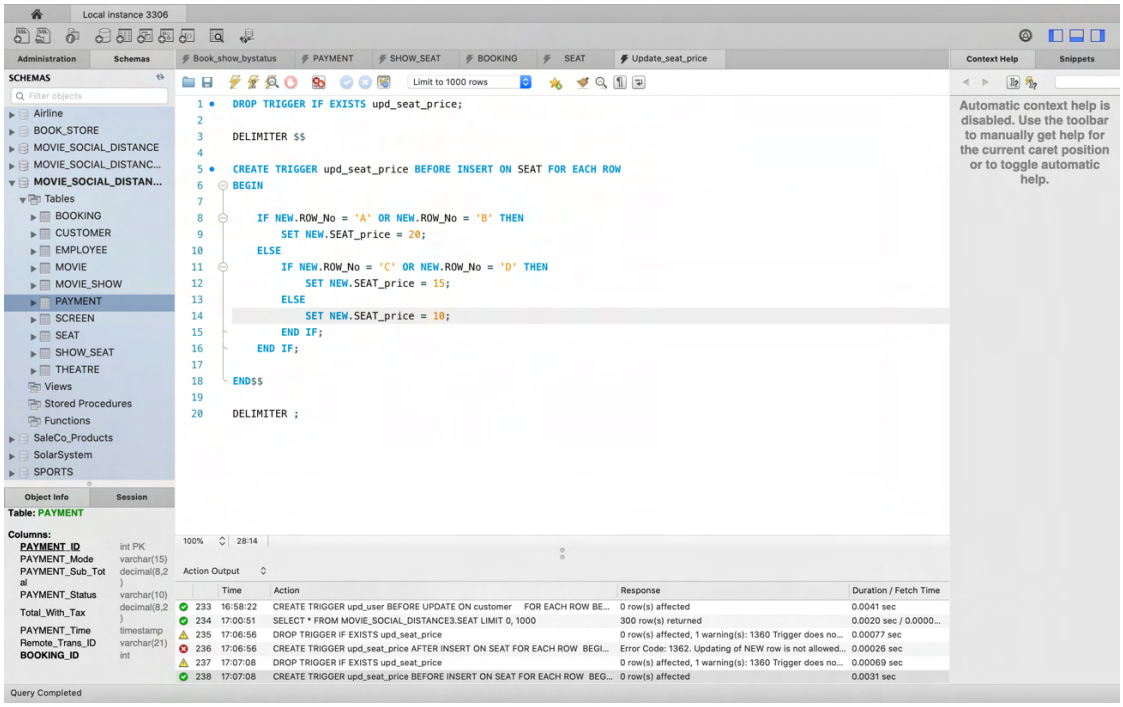


After update in the CUSTOMER table

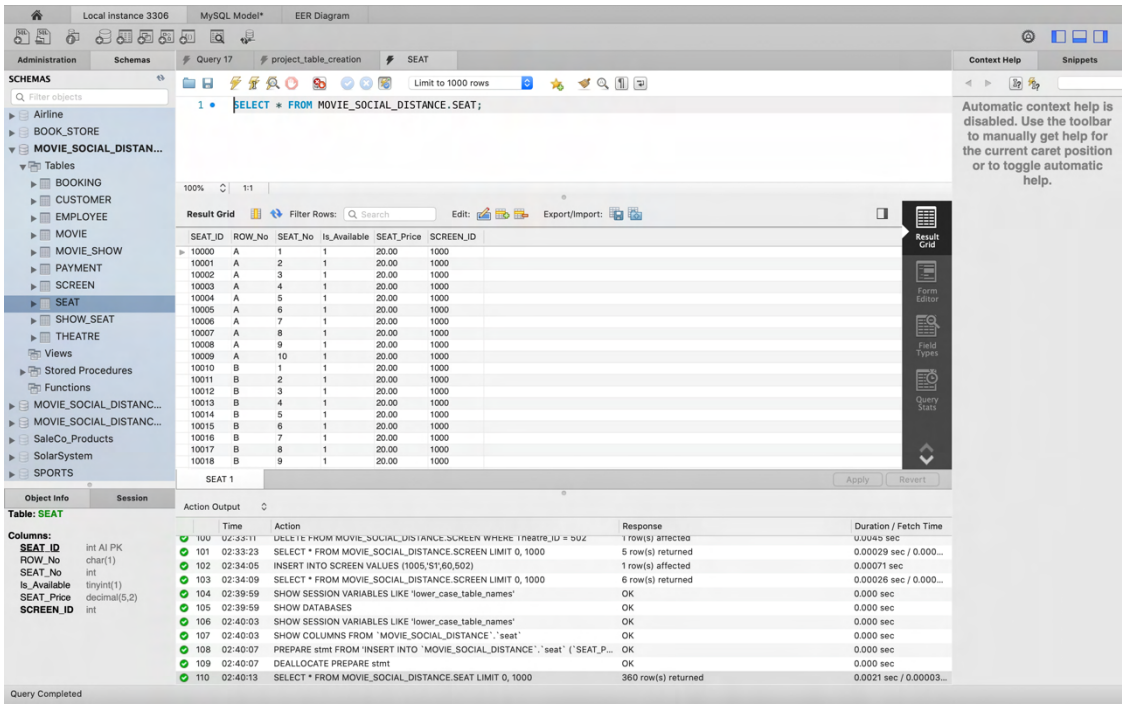


upd_seat_price

Execution of the trigger:

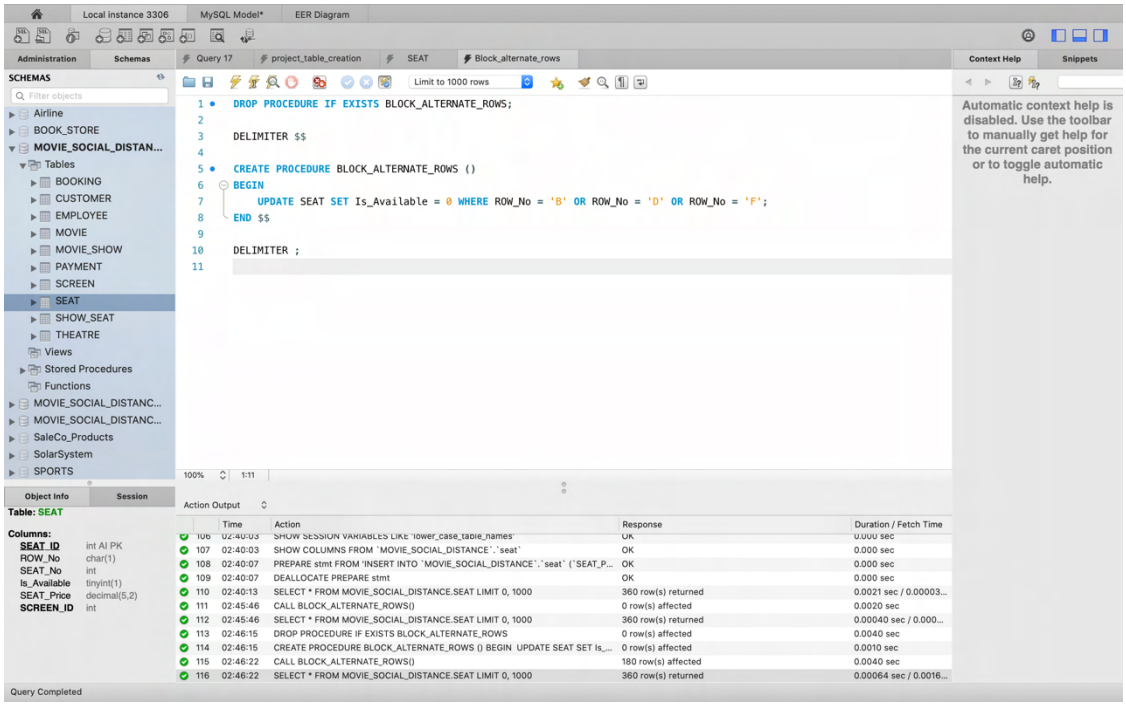


After insert in the SEAT table

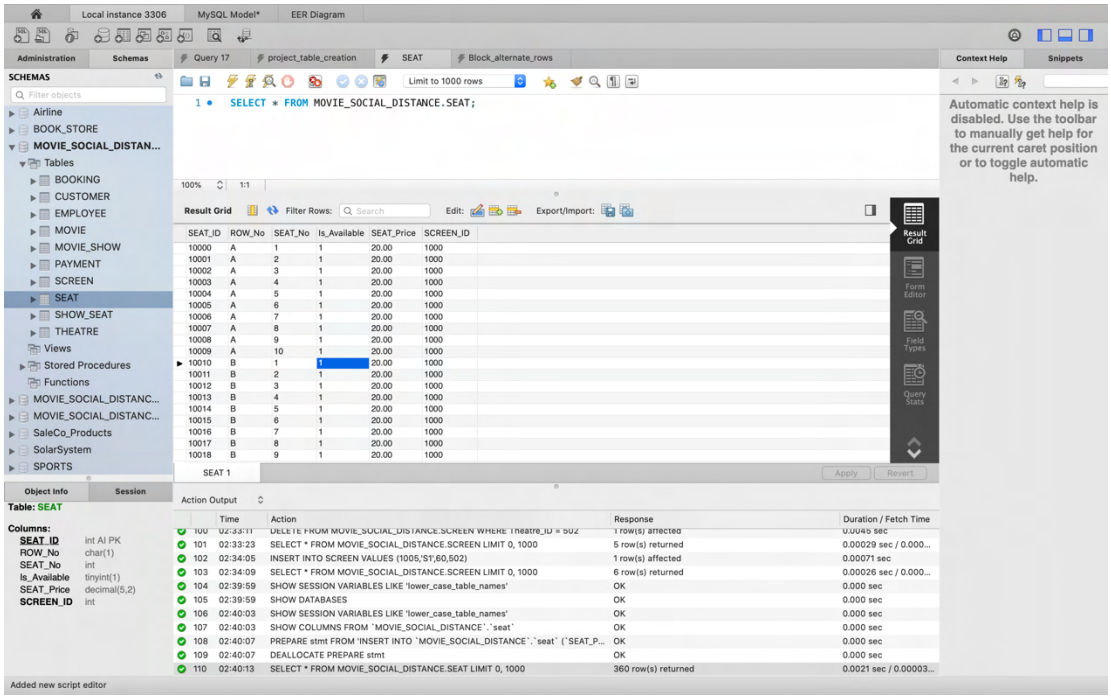


BLOCK_ALTERNATE_ROWS

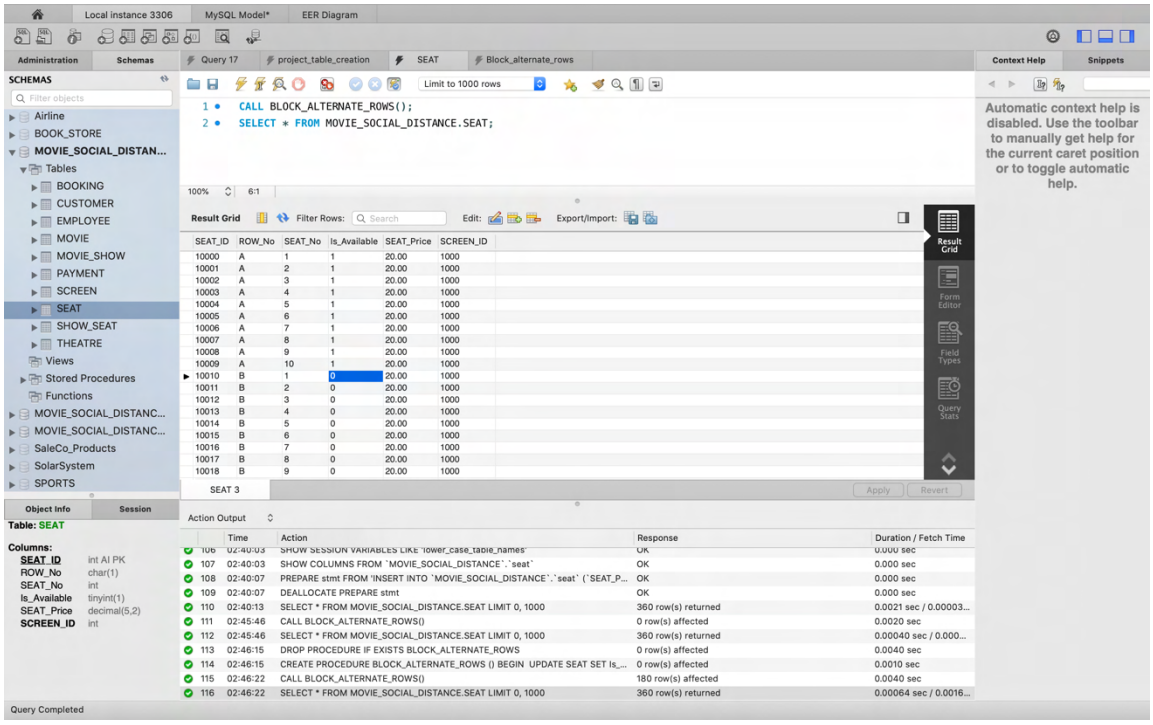
Execution of the stored procedure:



Before calling the procedure

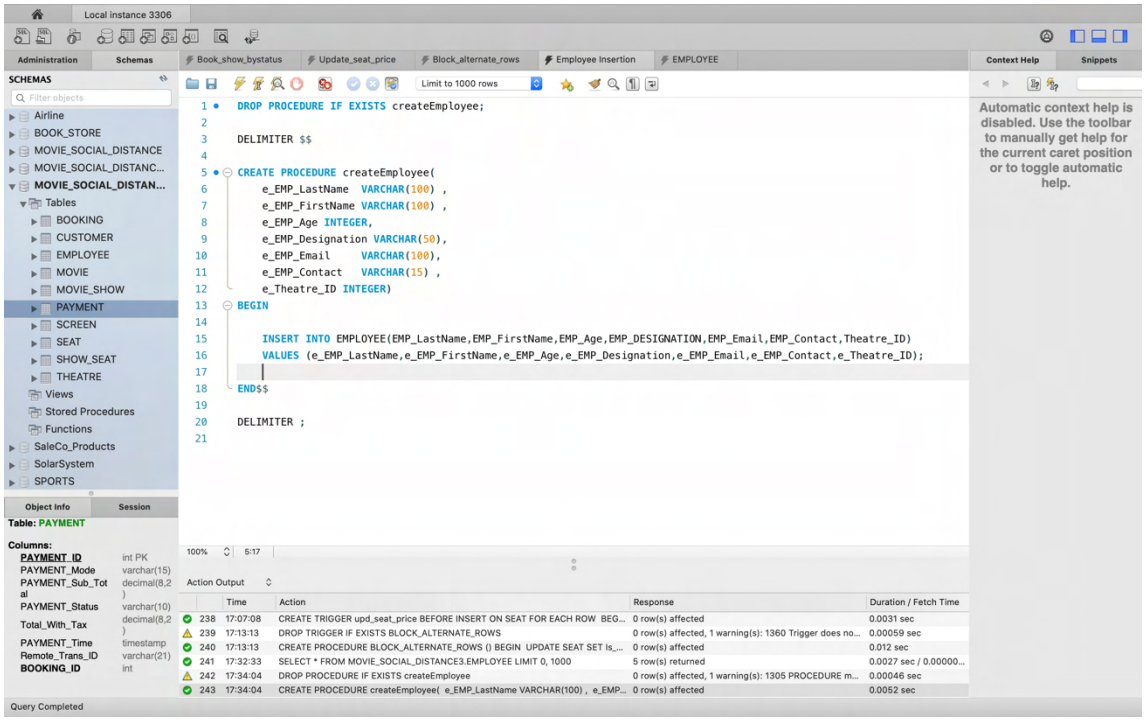


After calling the procedure



createEmployee

Execution of the Stored Procedure:



Before calling the procedure

```
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	lrhonda@tht.com	8221182212	501
3	Rhett	Vandam	Manager	vrhett@tht.com	8221182213	500
4	Anne	Jones	Manager	janne@tht.com	8221182214	501
5	John	Lange	Associate Manager	ljohn@tht.com	8221182215	501
NULL	NULL	NULL	NULL	NULL	NULL	NULL

After calling the procedure

Query 1 | SQL File 6* | employee | createEmployee - Routine

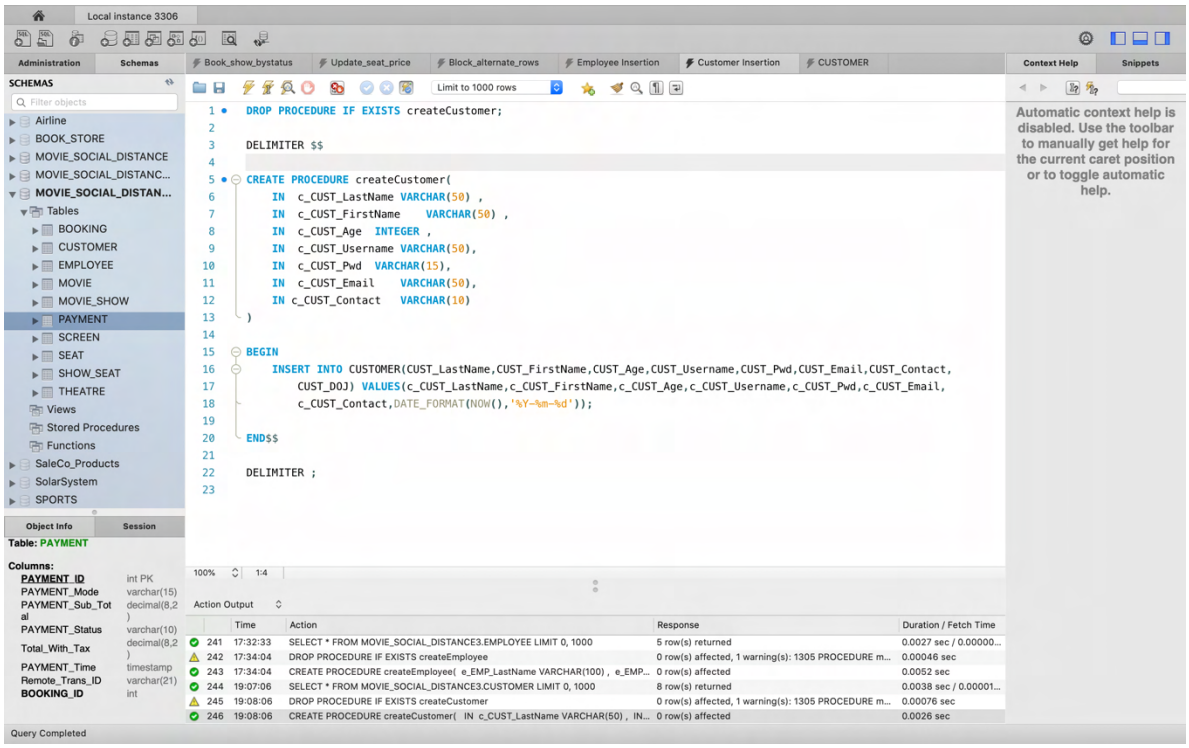
Limit to 1000 rows

```
1 • call createEmployee(
2   'John','Snow','Designer','johnsnow@tht.com','12345654189',500
3   );
4
5 • SELECT * FROM movie_social_distance_1.employee;
6
```

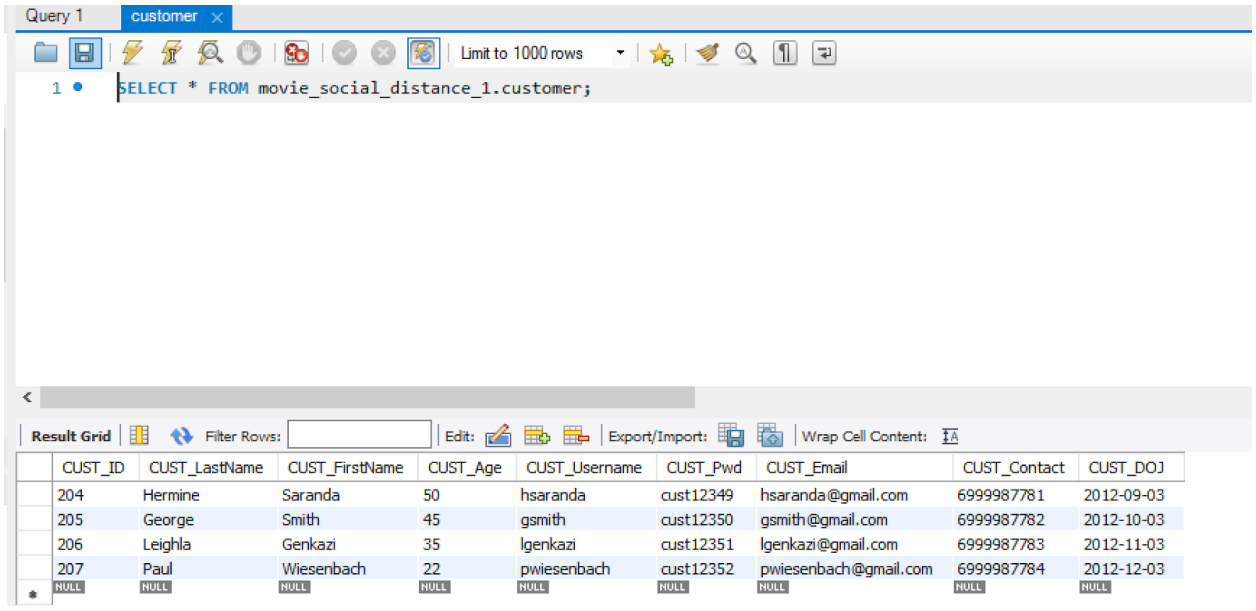
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

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

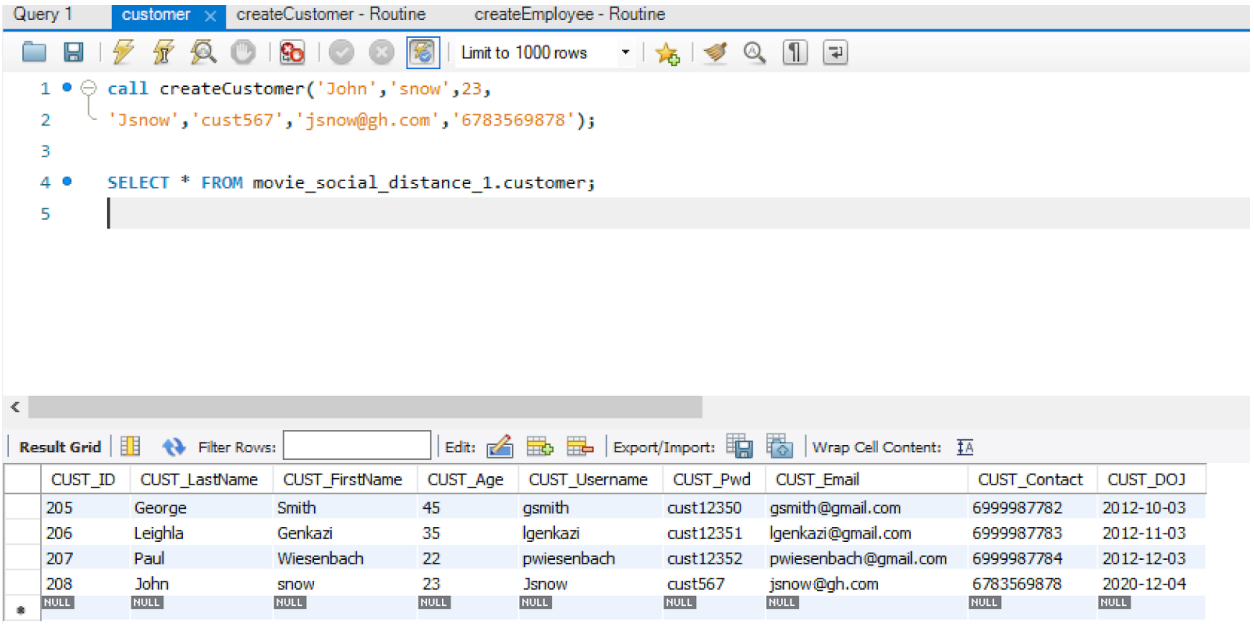
createCustomer



Before calling the procedure

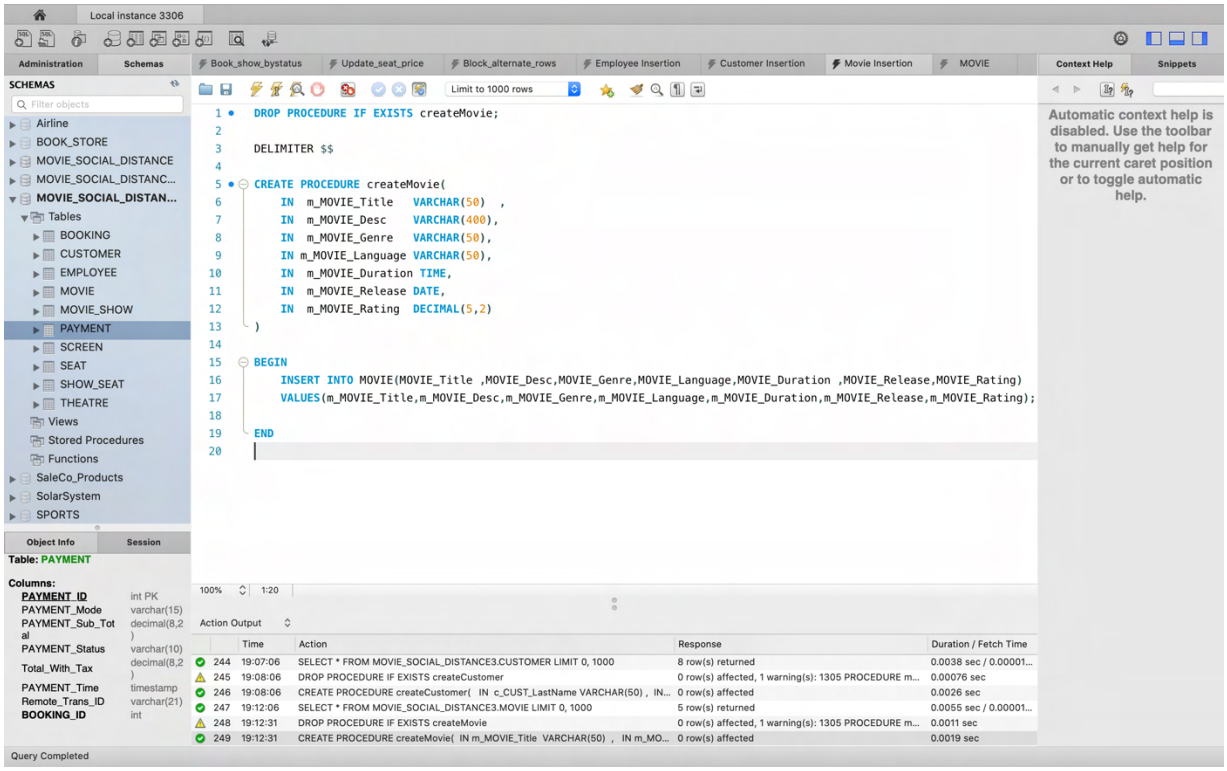


After calling the procedure

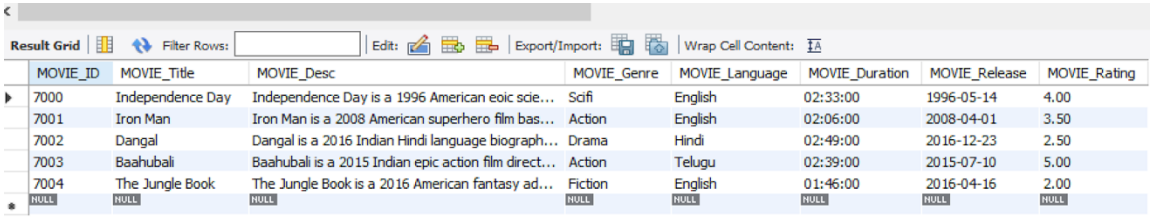


createMovie

Execution of the Stored Procedure:



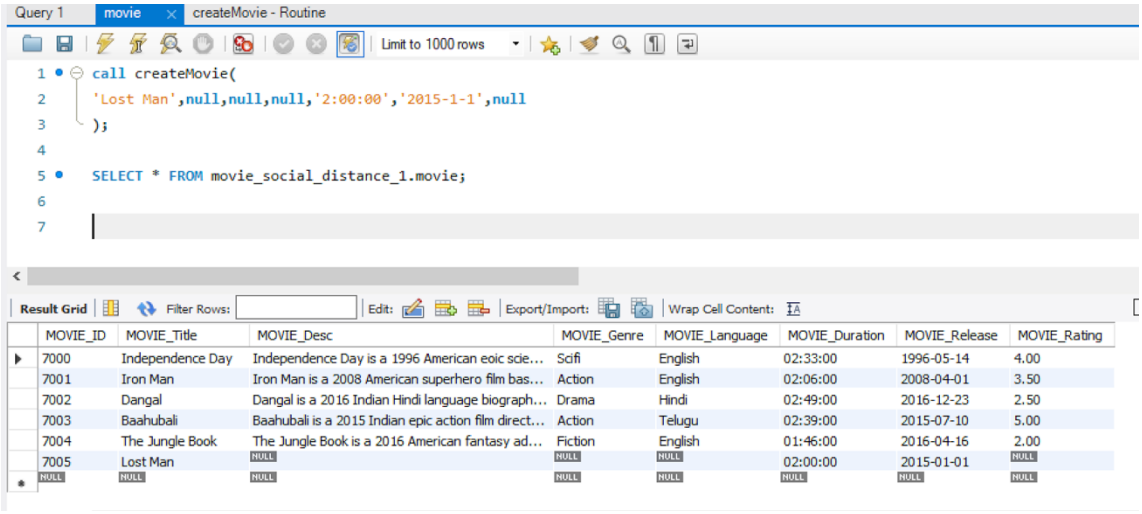
Before calling the procedure



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 eoic scie…	Scifi	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	NULL

After calling the procedure



Query 1 | movie | createMovie - Routine

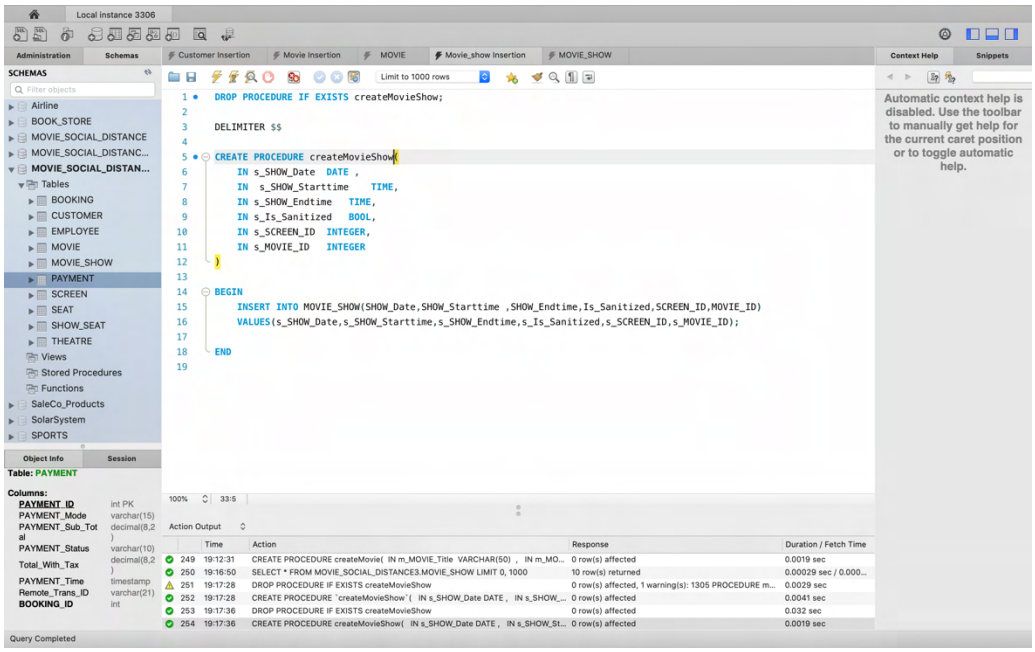
Limit to 1000 rows

```
1  call createMovie(
2  'Lost Man',null,null,null,'2:00:00','2015-1-1',null
3  );
4
5  SELECT * FROM movie_social_distance_1.movie;
6
7
```

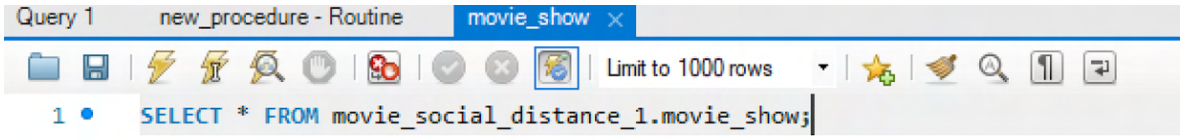
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

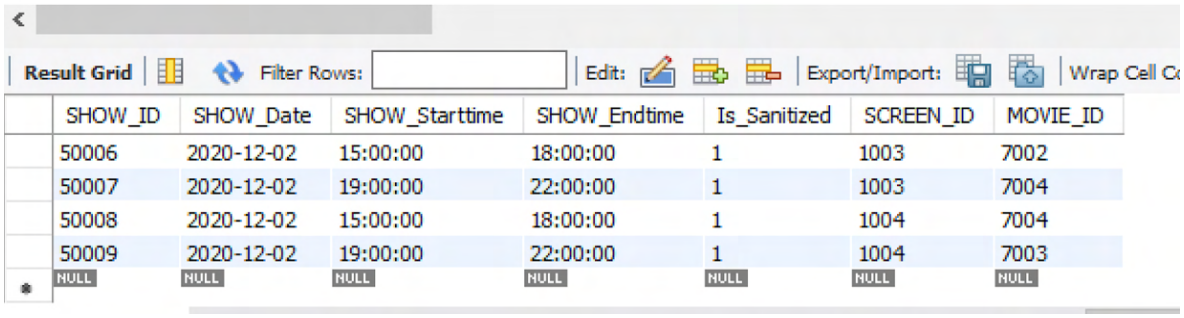
MOVIE_ID	MOVIE_Title	MOVIE_Desc	MOVIE_Genre	MOVIE_Language	MOVIE_Duration	MOVIE_Release	MOVIE_Rating
7000	Independence Day	Independence Day is a 1996 American eoic scie…	Scifi	English	02:33:00	1996-05-14	4.00
7001	Iron Man	Iron Man is a 2008 American superhero film bas…	Action	English	02:06:00	2008-04-01	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	NULL

createMovieShow



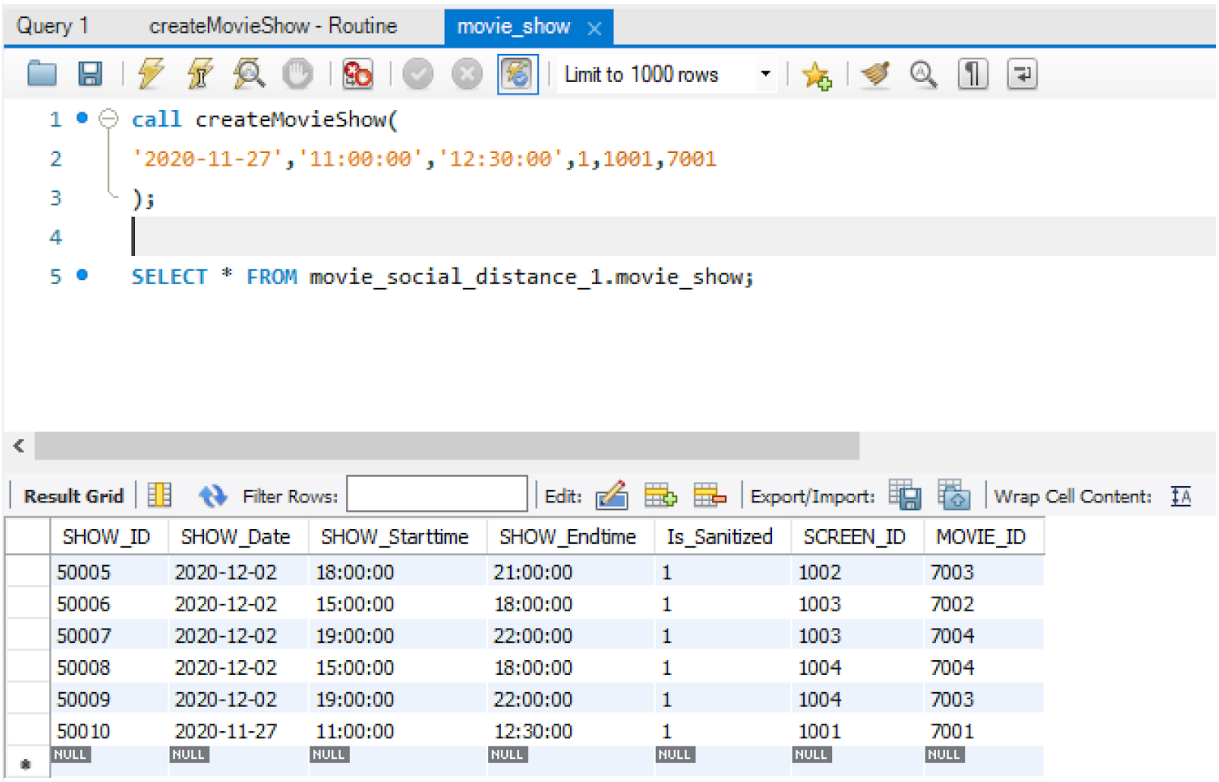
Before calling the procedure





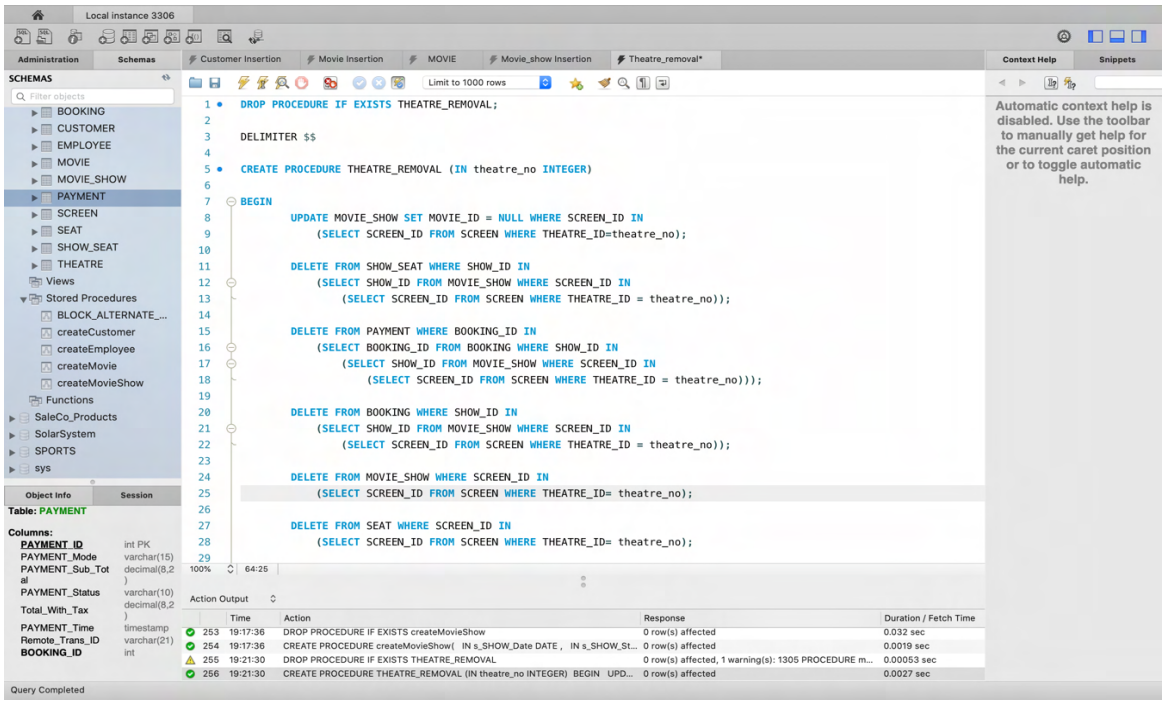
SHOW_ID	SHOW_Date	SHOW_Starttime	SHOW_Endtime	Is_Sanitized	SCREEN_ID	MOVIE_ID
50006	2020-12-02	15:00:00	18:00:00	1	1003	7002
50007	2020-12-02	19:00:00	22:00:00	1	1003	7004
50008	2020-12-02	15:00:00	18:00:00	1	1004	7004
50009	2020-12-02	19:00:00	22:00:00	1	1004	7003
NULL	NULL	NULL	NULL	NULL	NULL	NULL

After calling the procedure

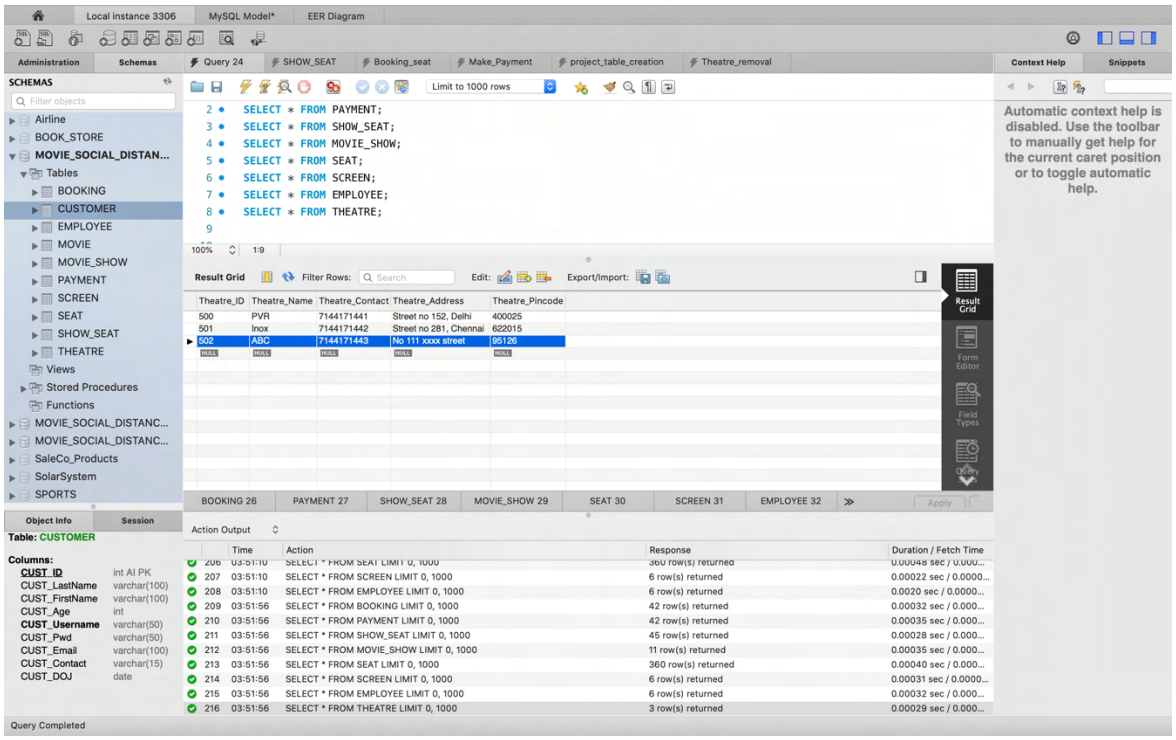


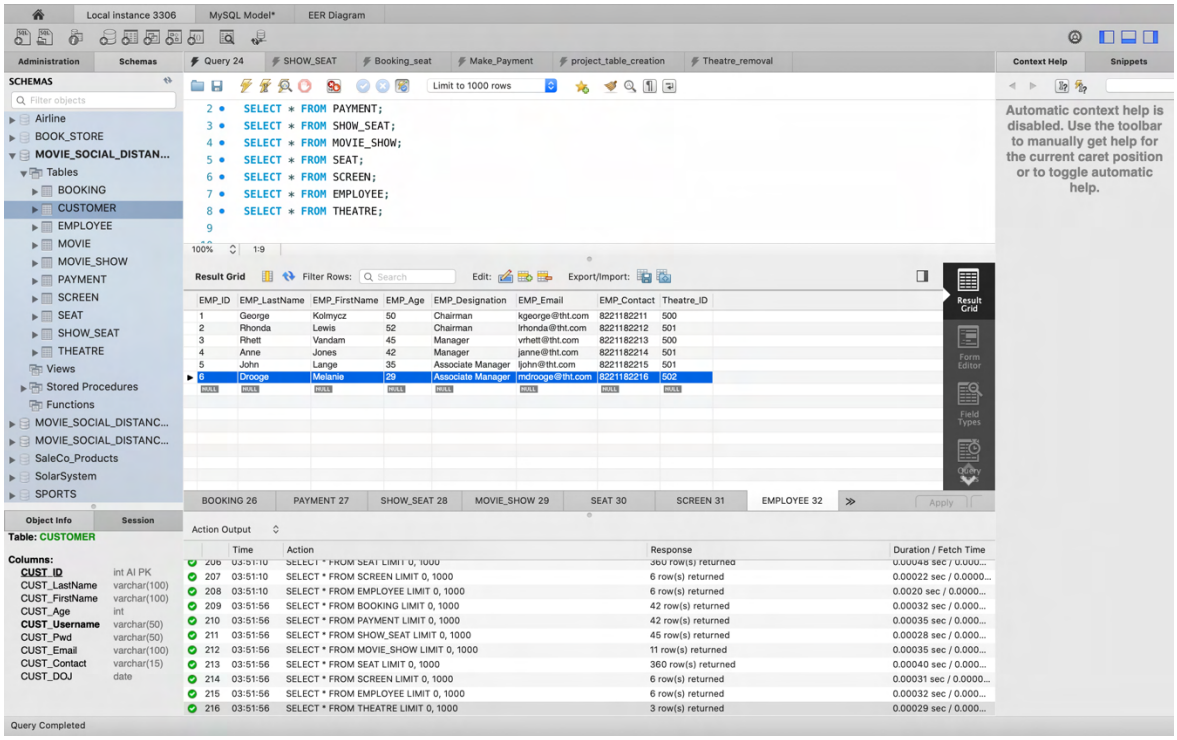
SHOW_ID	SHOW_Date	SHOW_Starttime	SHOW_Endtime	Is_Sanitized	SCREEN_ID	MOVIE_ID
50005	2020-12-02	18:00:00	21:00:00	1	1002	7003
50006	2020-12-02	15:00:00	18:00:00	1	1003	7002
50007	2020-12-02	19:00:00	22:00:00	1	1003	7004
50008	2020-12-02	15:00:00	18:00:00	1	1004	7004
50009	2020-12-02	19:00:00	22:00:00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

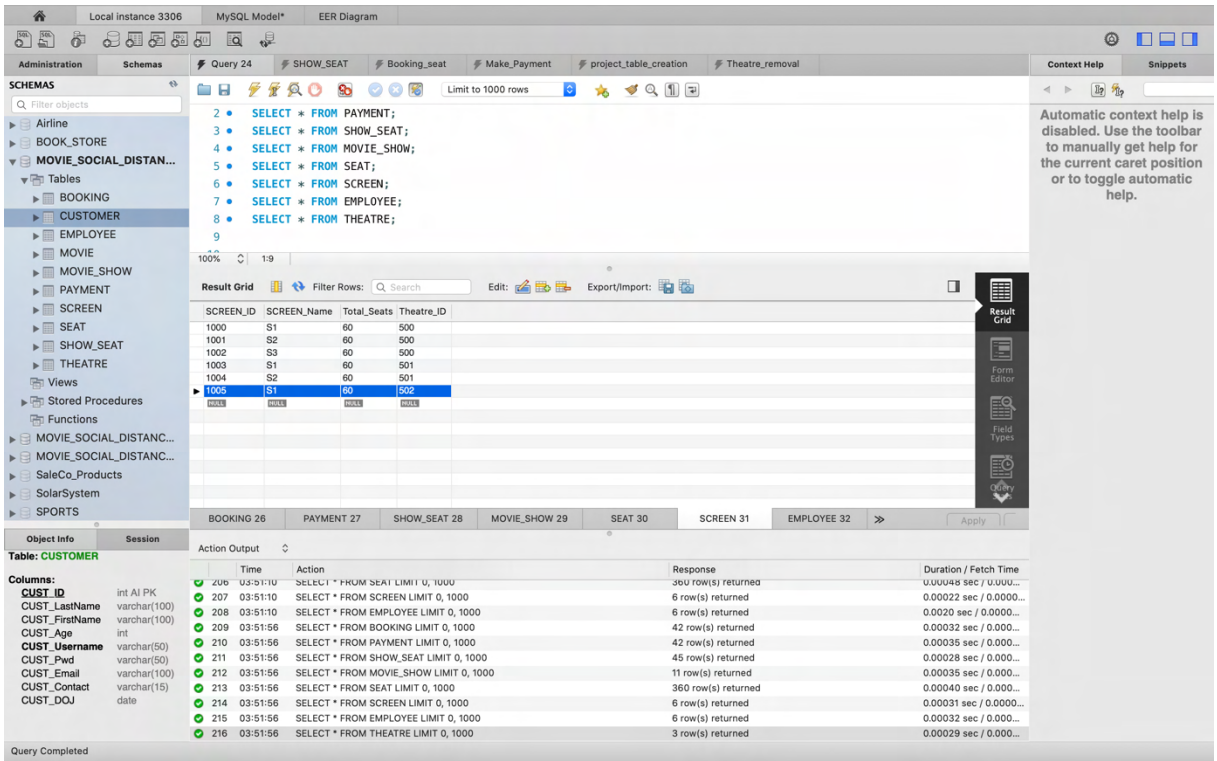
THEATRE_REMOVAL

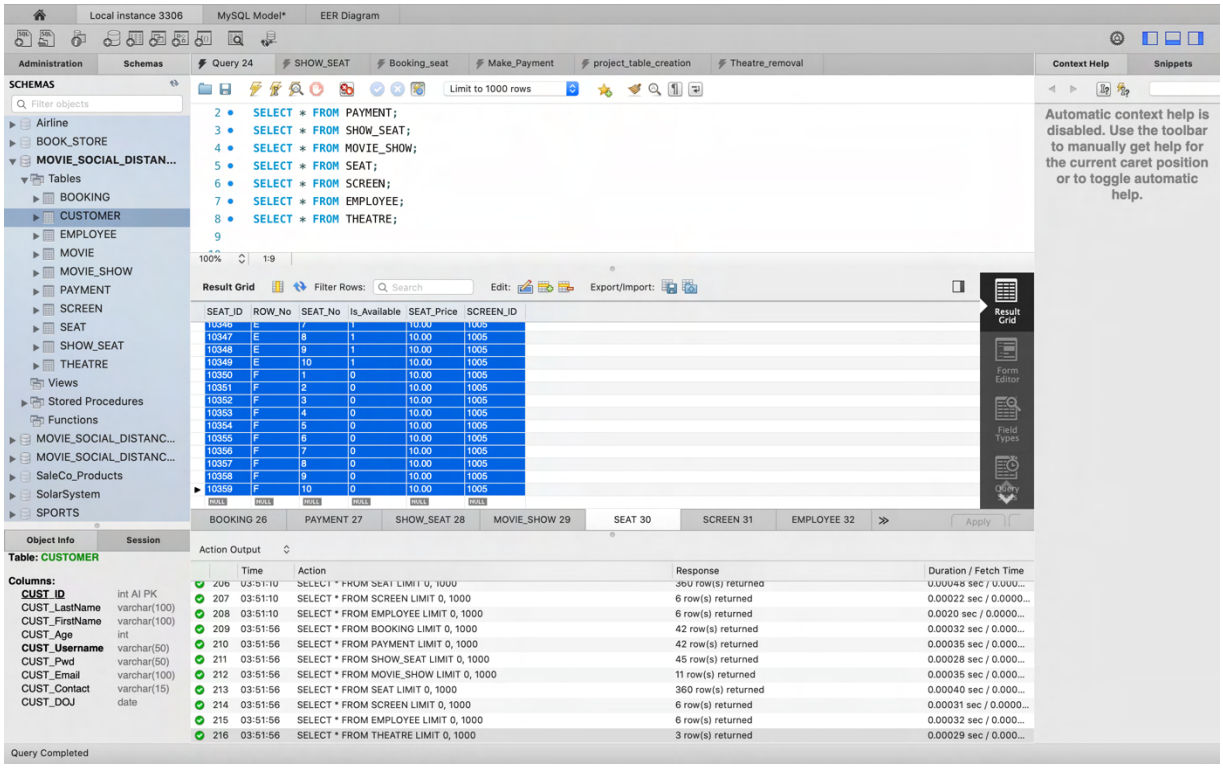


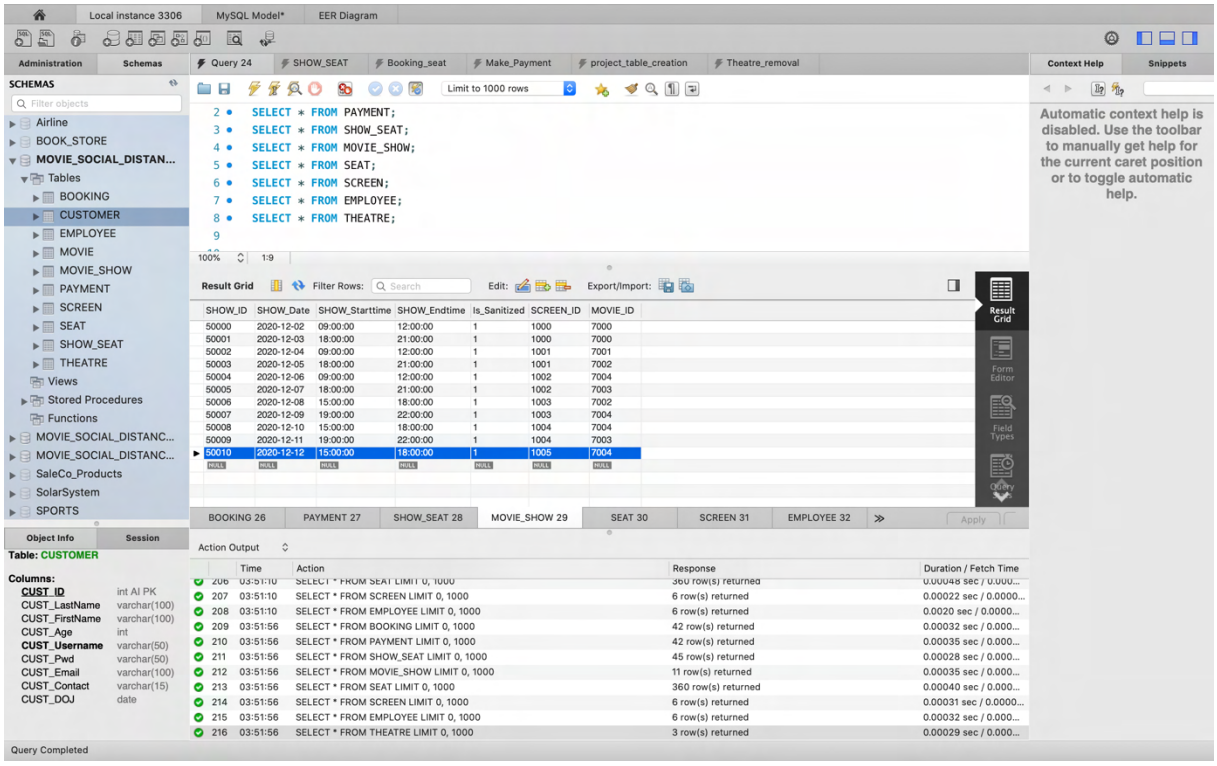
Before calling the procedure

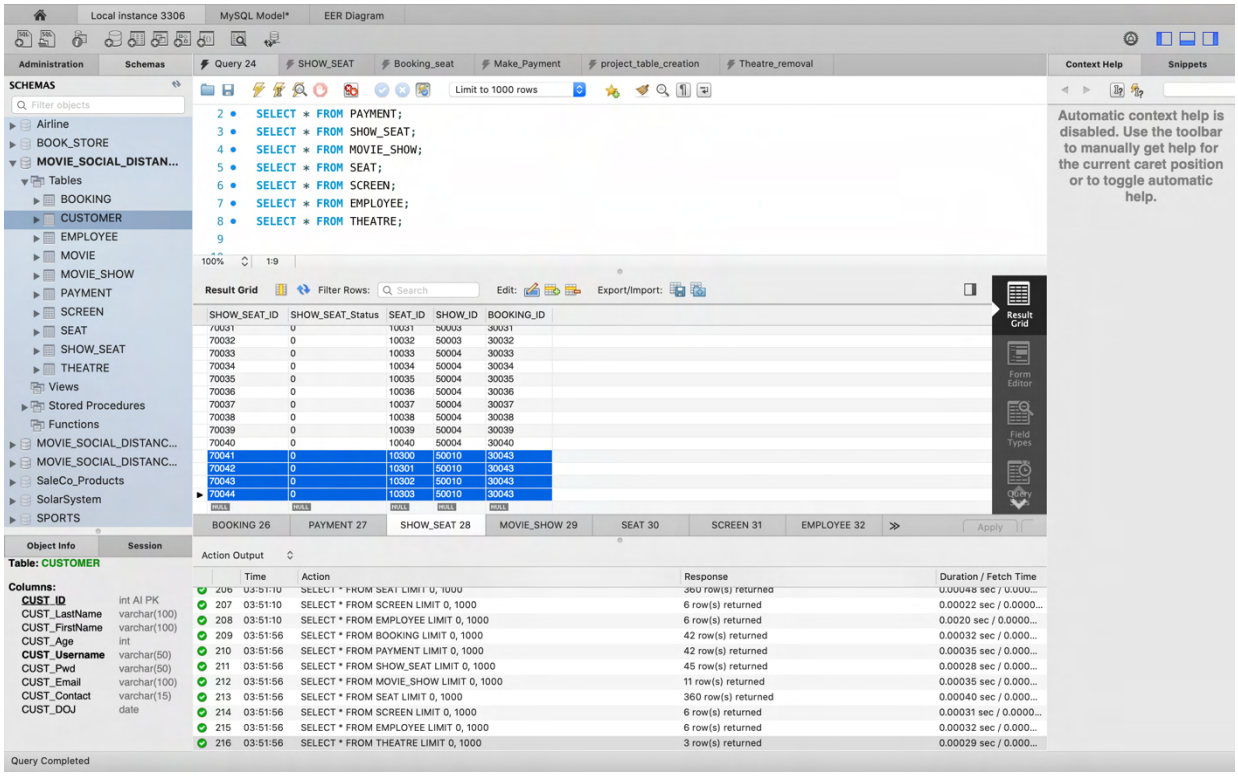


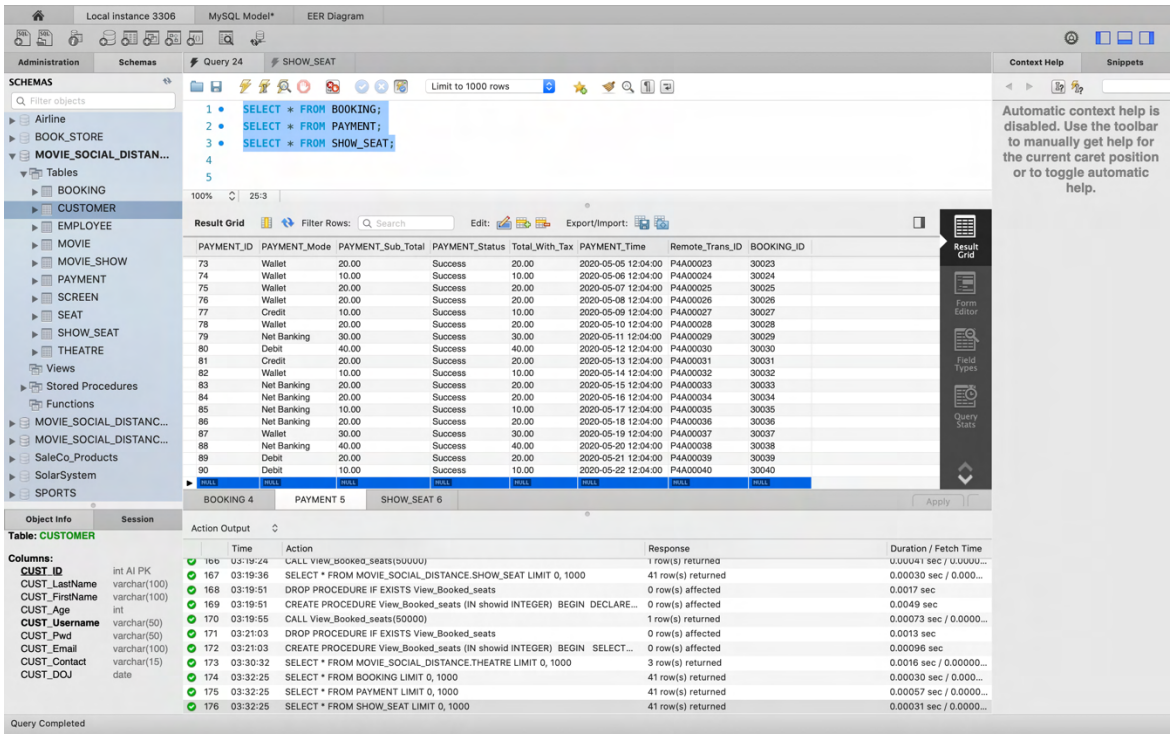

Local instance 3306
MySQL Model*
EER Diagram
Query 24
SHOW_SEAT
Booking_seat
Make_Payment
project_table_creation
Theatre_removal
SCHEMAS
Airline
BOOK_STORE
MOVIE_SOCIAL_DISTAN...
Tables
BOOKING
CUSTOMER
EMPLOYEE
MOVIE
MOVIE_SHOW
PAYMENT
SCREEN
SEAT
SHOW_SEAT
THEATRE
Views
Stored Procedures
Functions
SaleCo_Products
SolarSystem
SPORTS
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
SELECT * FROM PAYMENT;
SELECT * FROM SHOW_SEAT;
SELECT * FROM MOVIE_SHOW;
SELECT * FROM SEAT;
SELECT * FROM SCREEN;
SELECT * FROM EMPLOYEE;
SELECT * FROM THEATRE;
Result Grid
EMP_ID EMP_LastName EMP_FirstName EMP_Age EMP_Designation EMP_Email EMP_Contact Theatre_ID
1 George Kolmycz 50 Chairman kgeorge@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 jjohn@tht.com 8221182215 501
6 Drooge Melanie 29 Associate Manager mdrooge@tht.com 8221182216 502
Action Output
Query Completed

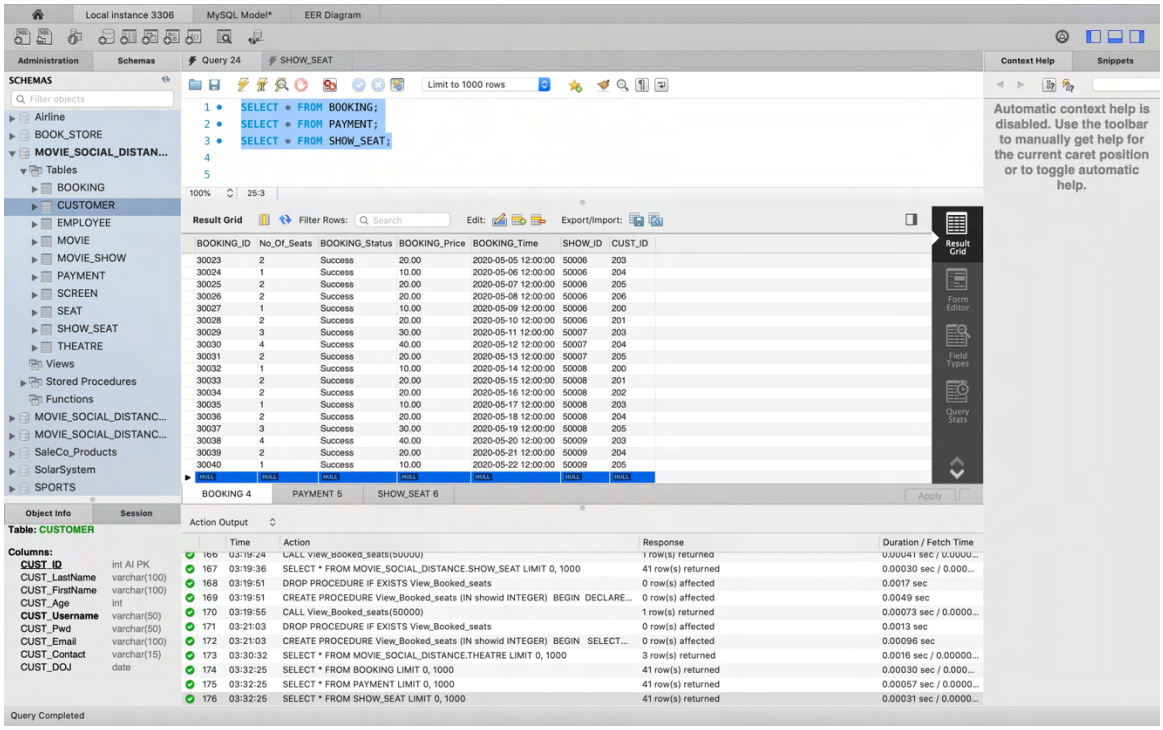

Local instance 3306
MySQL Model*
EER Diagram
SCHEMAS
CUSTOMER
SELECT * FROM PAYMENT;
SELECT * FROM SHOW_SEAT;
SELECT * FROM MOVIE_SHOW;
SELECT * FROM SEAT;
SELECT * FROM SCREEN;
SELECT * FROM EMPLOYEE;
SELECT * FROM THEATRE;
Result Grid
SCREEN_ID SCREEN_Name Total_Seats Theatre_ID
1000 S1 60 500
1001 S2 60 500
1002 S3 60 500
1003 S1 60 501
1004 S2 60 501
1005 S1 60 502
Action Output
Query Completed


Local instance 3306
MySQL Model*
EER Diagram
Query 24
SHOW_SEAT
Booking_seat
Make_Payment
project_table_creation
Theatre_removal
SCHEMAS
MOVIE_SOCIAL_DISTAN...
Tables
BOOKING
CUSTOMER
EMPLOYEE
MOVIE
MOVIE_SHOW
PAYMENT
SCREEN
SEAT
SHOW_SEAT
THEATRE
Views
Stored Procedures
Functions
SELECT * FROM PAYMENT;
SELECT * FROM SHOW_SEAT;
SELECT * FROM MOVIE_SHOW;
SELECT * FROM SEAT;
SELECT * FROM SCREEN;
SELECT * FROM EMPLOYEE;
SELECT * FROM THEATRE;
Result Grid
Action Output
Query Completed

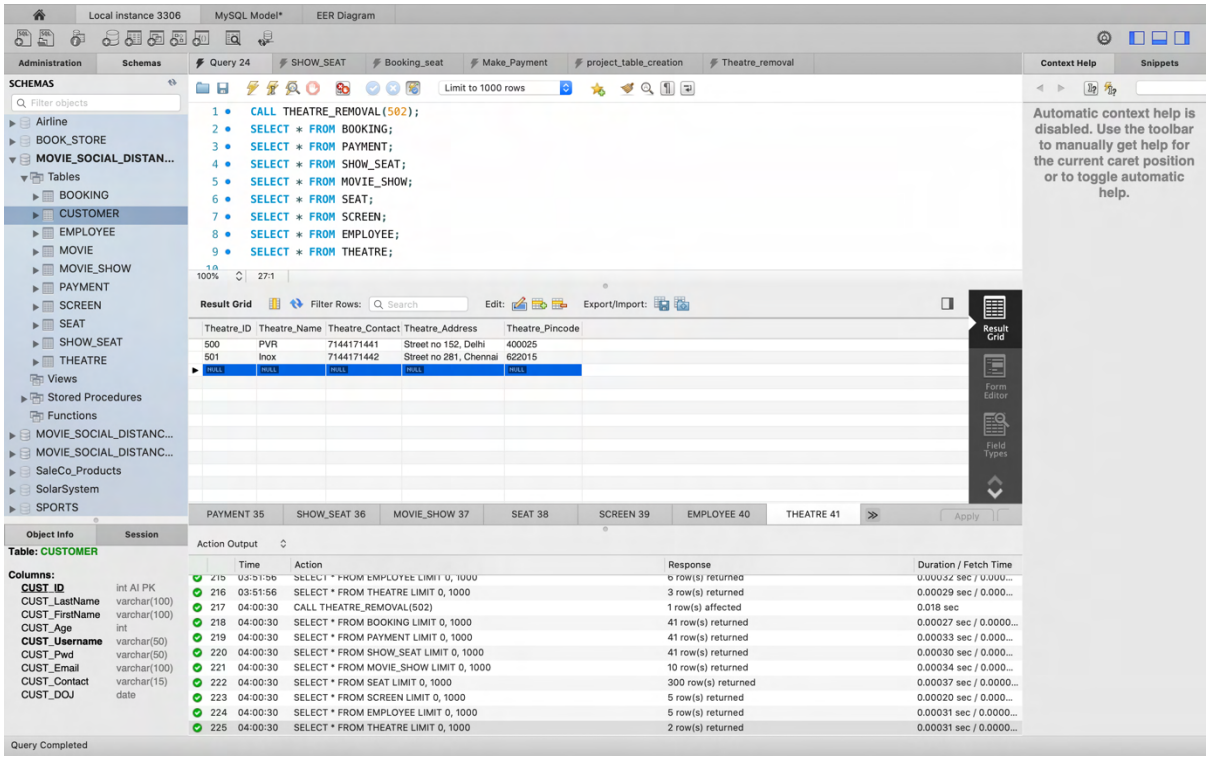

Local instance 3306
MySQL Model*
EER Diagram
Query 24
SHOW_SEAT
Booking_seat
Make_Payment
project_table_creation
Theatre_removal
SCHEMAS
MOVIE_SOCIAL_DISTAN...
Tables
BOOKING
CUSTOMER
EMPLOYEE
MOVIE
MOVIE_SHOW
PAYMENT
SCREEN
SEAT
SHOW_SEAT
THEATRE
Views
Stored Procedures
Functions
SELECT * FROM PAYMENT;
SELECT * FROM SHOW_SEAT;
SELECT * FROM MOVIE_SHOW;
SELECT * FROM SEAT;
SELECT * FROM SCREEN;
SELECT * FROM EMPLOYEE;
SELECT * FROM THEATRE;
Result Grid
Action Output
Query Completed

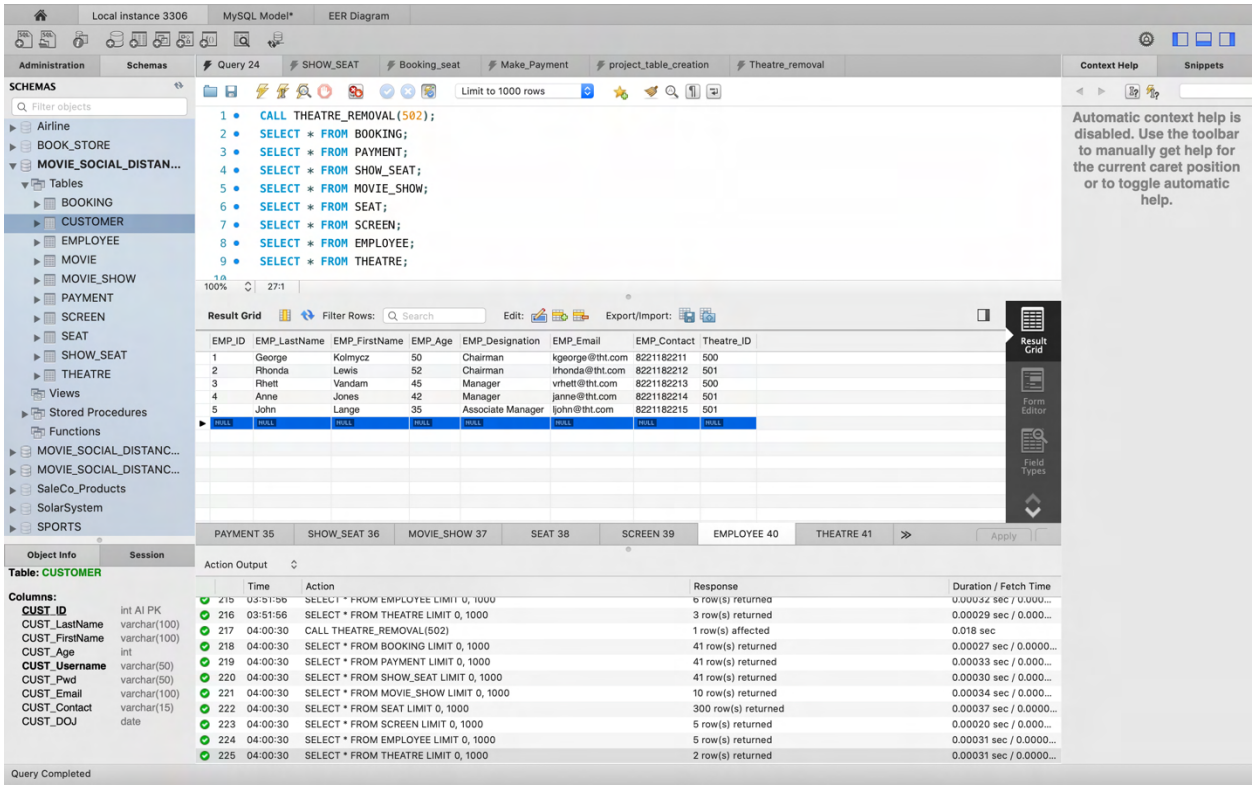

Local instance 3306
MySQL Model*
EER Diagram
Query 24
SHOW_SEAT
Booking_seat
Make_Payment
project_table_creation
Theatre_removal
SELECT * FROM PAYMENT;
SELECT * FROM SHOW_SEAT;
SELECT * FROM MOVIE_SHOW;
SELECT * FROM SEAT;
SELECT * FROM SCREEN;
SELECT * FROM EMPLOYEE;
SELECT * FROM THEATRE;
Result Grid
BOOKING 26
PAYMENT 27
SHOW_SEAT 28
MOVIE_SHOW 29
SEAT 30
SCREEN 31
EMPLOYEE 32
Action Output
Query Completed

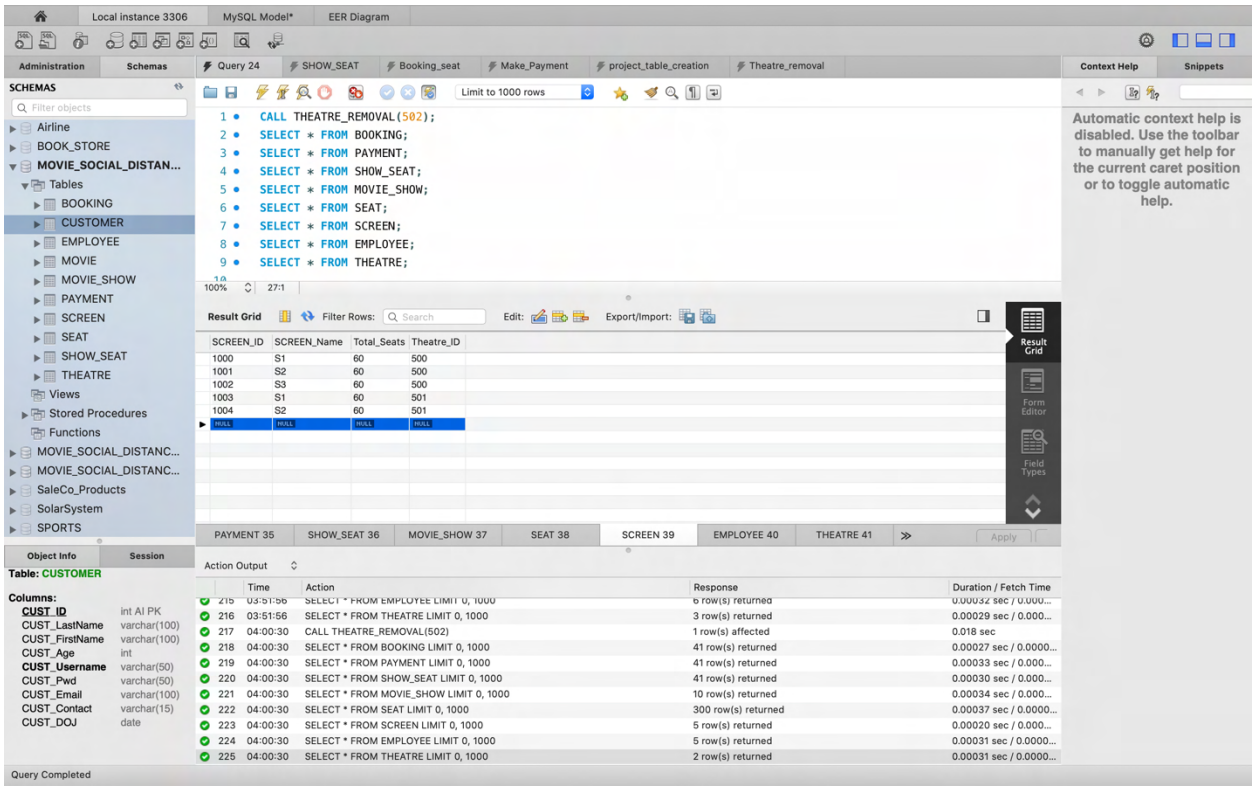

Local instance 3306
MySQL Model*
EER Diagram
Query 24
SHOW_SEAT
SELECT * FROM BOOKING;
SELECT * FROM PAYMENT;
SELECT * FROM SHOW_SEAT;
Result Grid
BOOKING 4
PAYMENT 5
SHOW_SEAT 6
Action Output
Query Completed

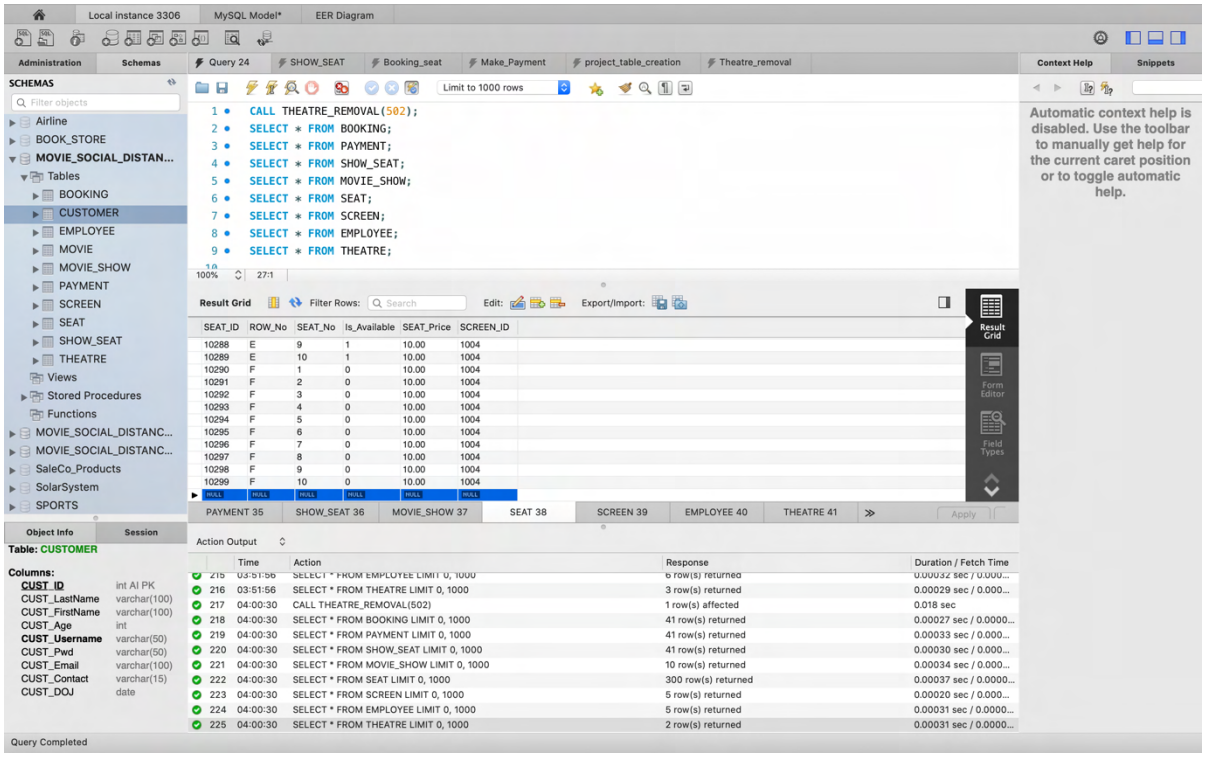


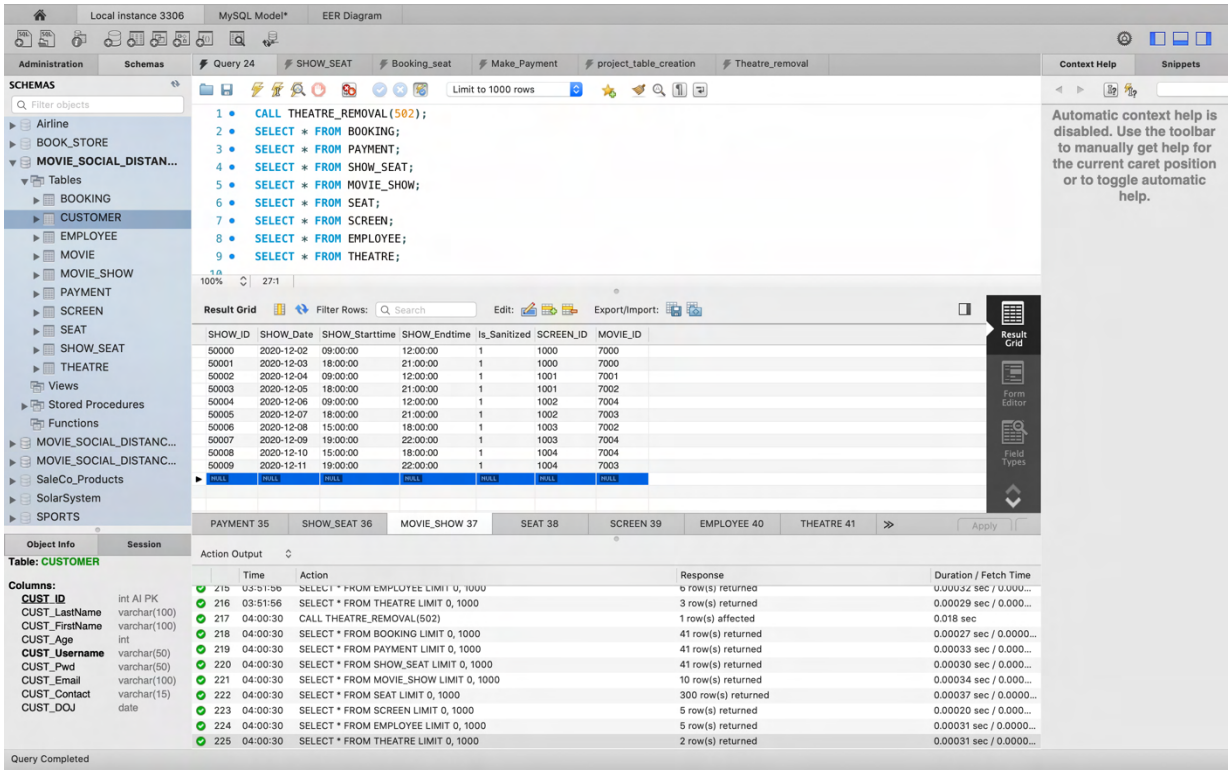
After calling the procedure

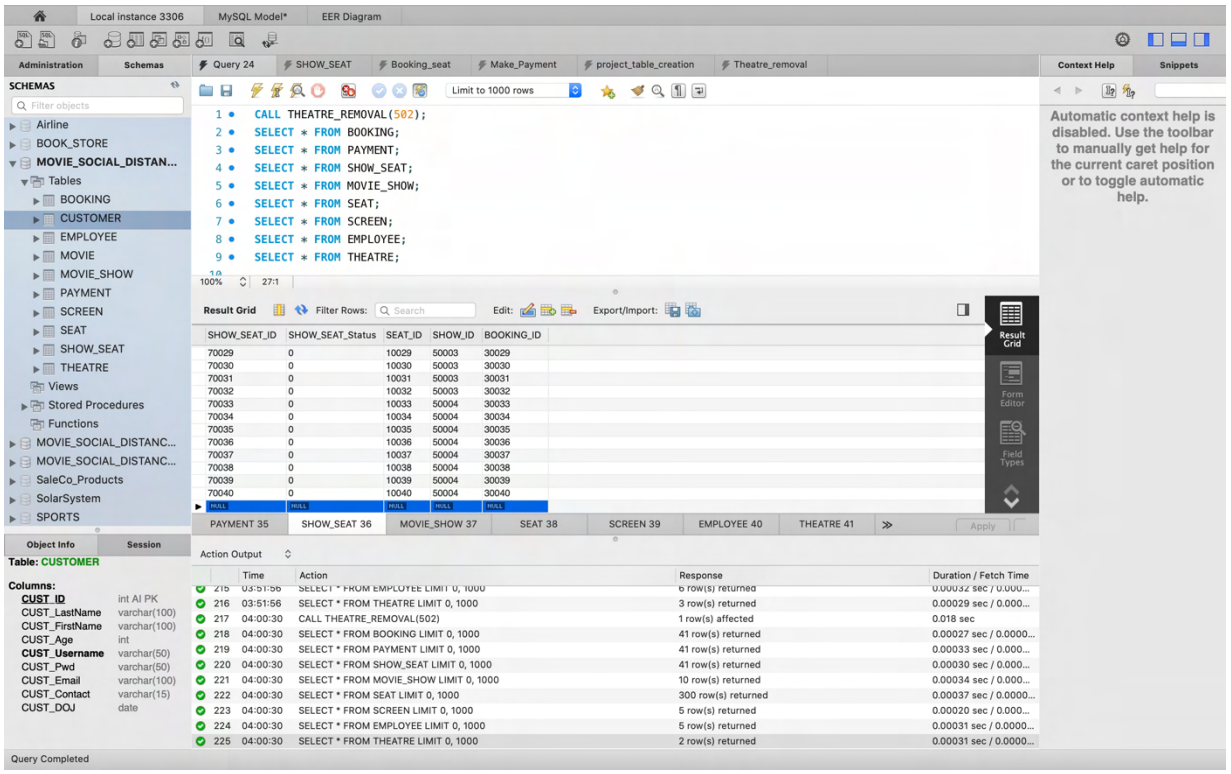

Local instance 3306
MySQL Model*
EER Diagram
Query 24
SHOW_SEAT
Booking_seat
Make_Payment
project_table_creation
Theatre_removal
SCHEMAS
Airline
BOOK_STORE
MOVIE_SOCIAL_DISTAN...
Tables
BOOKING
CUSTOMER
EMPLOYEE
MOVIE
MOVIE_SHOW
PAYMENT
SCREEN
SEAT
SHOW_SEAT
THEATRE
Views
Stored Procedures
Functions
MOVIE_SOCIAL_DISTANC...
MOVIE_SOCIAL_DISTANC...
SaleCo_Products
SolarSystem
SPORTS
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
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;
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 622015
Action Output
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 / 0.0000...
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 / 0.0000...
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 / 0.0000...
225 04:00:30 SELECT * FROM THEATRE LIMIT 0, 1000 2 row(s) returned 0.00031 sec / 0.0000...
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Query Completed

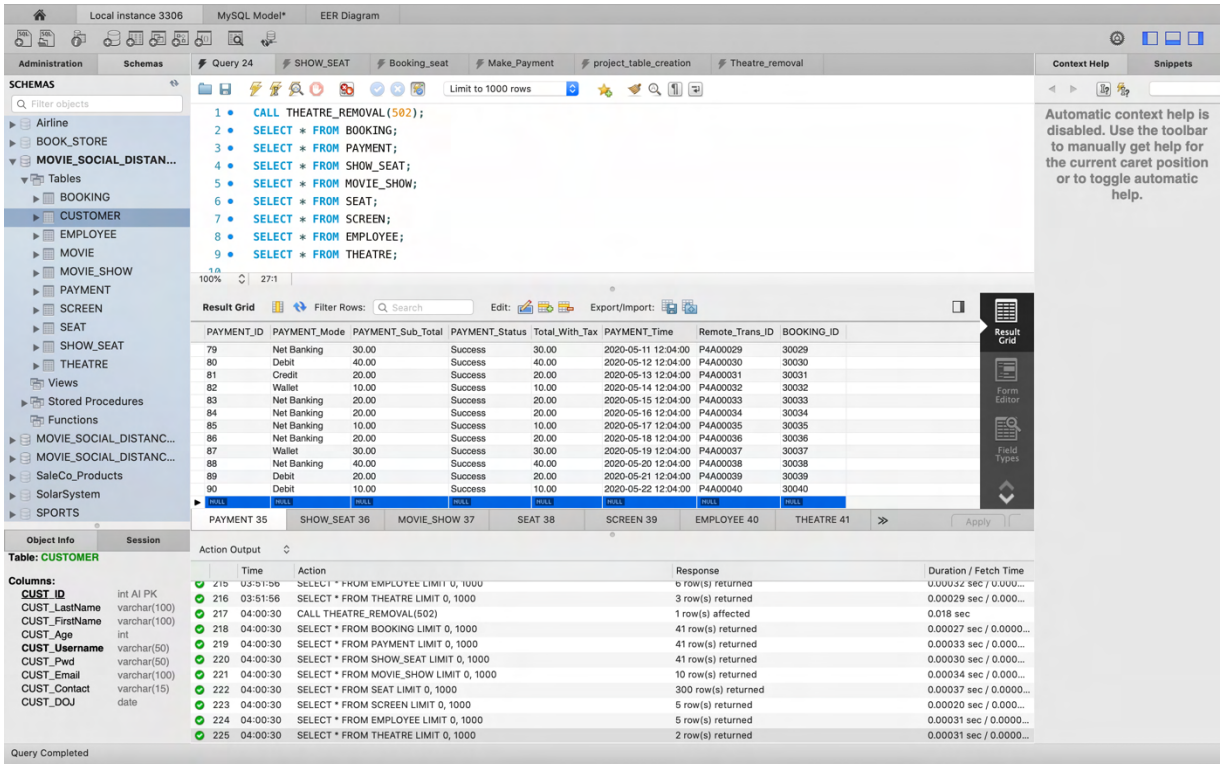

Local instance 3306
MySQL Model*
EER Diagram
Query 24
SHOW_SEAT
Booking_seat
Make_Payment
project_table_creation
Theatre_removal
SCHEMAS
Airline
BOOK_STORE
MOVIE_SOCIAL_DISTAN...
Tables
BOOKING
CUSTOMER
EMPLOYEE
MOVIE
MOVIE_SHOW
PAYMENT
SCREEN
SEAT
SHOW_SEAT
THEATRE
Views
Stored Procedures
Functions
MOVIE_SOCIAL_DISTANC...
MOVIE_SOCIAL_DISTANC...
SaleCo_Products
SolarSystem
SPORTS
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
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;
EMP_ID EMP_LastName EMP_FirstName EMP_Age EMP_Designation EMP_Email EMP_Contact Theatre_ID
1 George Kolmycz 50 Chairman kgeorge@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
Action Output
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 / 0.0000...
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 / 0.0000...
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 / 0.0000...
225 04:00:30 SELECT * FROM THEATRE LIMIT 0, 1000 2 row(s) returned 0.00031 sec / 0.0000...
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Query Completed

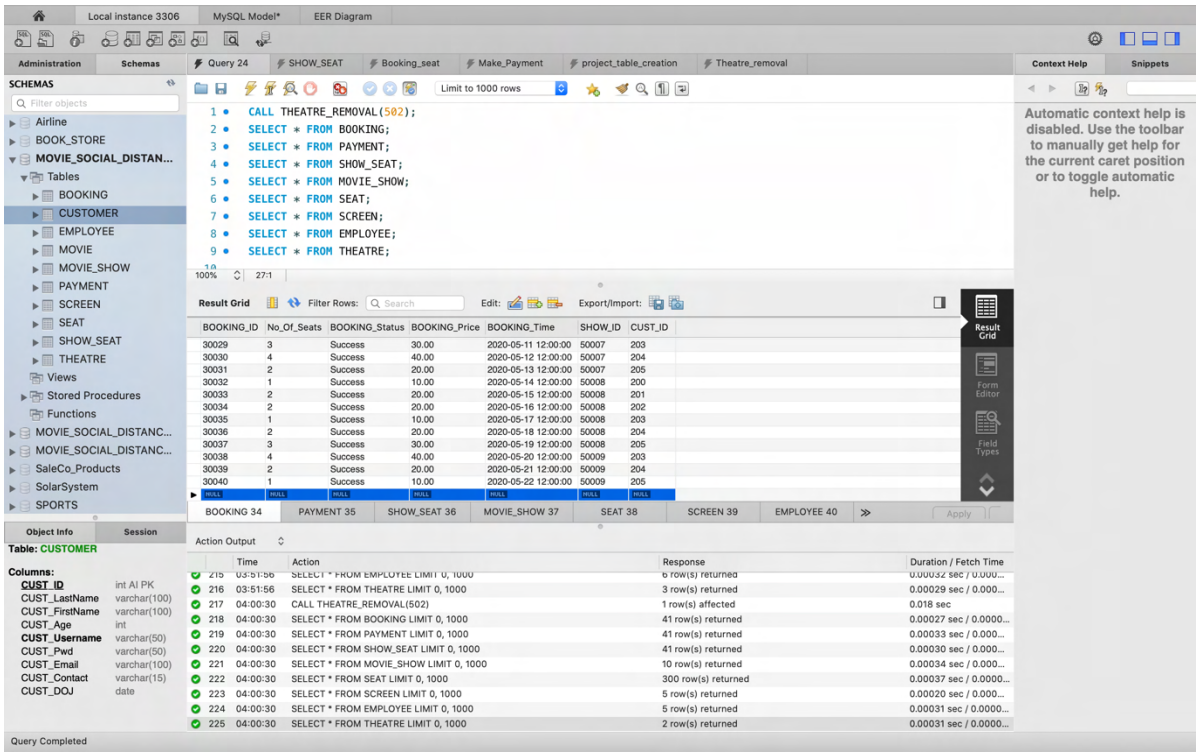

Local instance 3306
MySQL Model*
EER Diagram
Query 24
SHOW_SEAT
Booking_seat
Make_Payment
project_table_creation
Theatre_removal
SCHEMAS
Airline
BOOK_STORE
MOVIE_SOCIAL_DISTAN...
Tables
BOOKING
CUSTOMER
EMPLOYEE
MOVIE
MOVIE_SHOW
PAYMENT
SCREEN
SEAT
SHOW_SEAT
THEATRE
Views
Stored Procedures
Functions
CALL THEATRE_REMOVAL(502);
SELECT * FROM BOOKING;
SELECT * FROM PAYMENT;
SELECT * FROM SHOW_SEAT;
SELECT * FROM MOVIE_SHOW;
SELECT * FROM SEAT;
SELECT * FROM SCREEN;
SELECT * FROM EMPLOYEE;
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
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 / 0.0000...
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 / 0.0000...
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 / 0.0000...
225 04:00:30 SELECT * FROM THEATRE LIMIT 0, 1000 2 row(s) returned 0.00031 sec / 0.0000...
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
Query Completed


Local instance 3306
MySQL Model*
EER Diagram
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
Query Completed


Local instance 3306
MySQL Model*
EER Diagram
Query 24
SHOW_SEAT
Booking_seat
Make_Payment
project_table_creation
Theatre_removal
SCHEMAS
Airline
BOOK_STORE
MOVIE_SOCIAL_DISTAN...
Tables
BOOKING
CUSTOMER
EMPLOYEE
MOVIE
MOVIE_SHOW
PAYMENT
SCREEN
SEAT
SHOW_SEAT
THEATRE
Views
Stored Procedures
Functions
SaleCo_Products
SolarSystem
SPORTS
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
SHOW_ID SHOW_Date SHOW_Starttime SHOW_Endtime Is_Sanitized SCREEN_ID MOVIE_ID
50000 2020-12-02 09:00:00 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
50003 2020-12-05 18:00:00 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
PAYMENT 35
SHOW_SEAT 36
MOVIE_SHOW 37
SEAT 38
SCREEN 39
EMPLOYEE 40
THEATRE 41
Context Help
Snippets
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: 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
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 / 0.0000...
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 / 0.0000...
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 / 0.0000...
225 04:00:30 SELECT * FROM THEATRE LIMIT 0, 1000 2 row(s) returned 0.00031 sec / 0.0000...
Query Completed

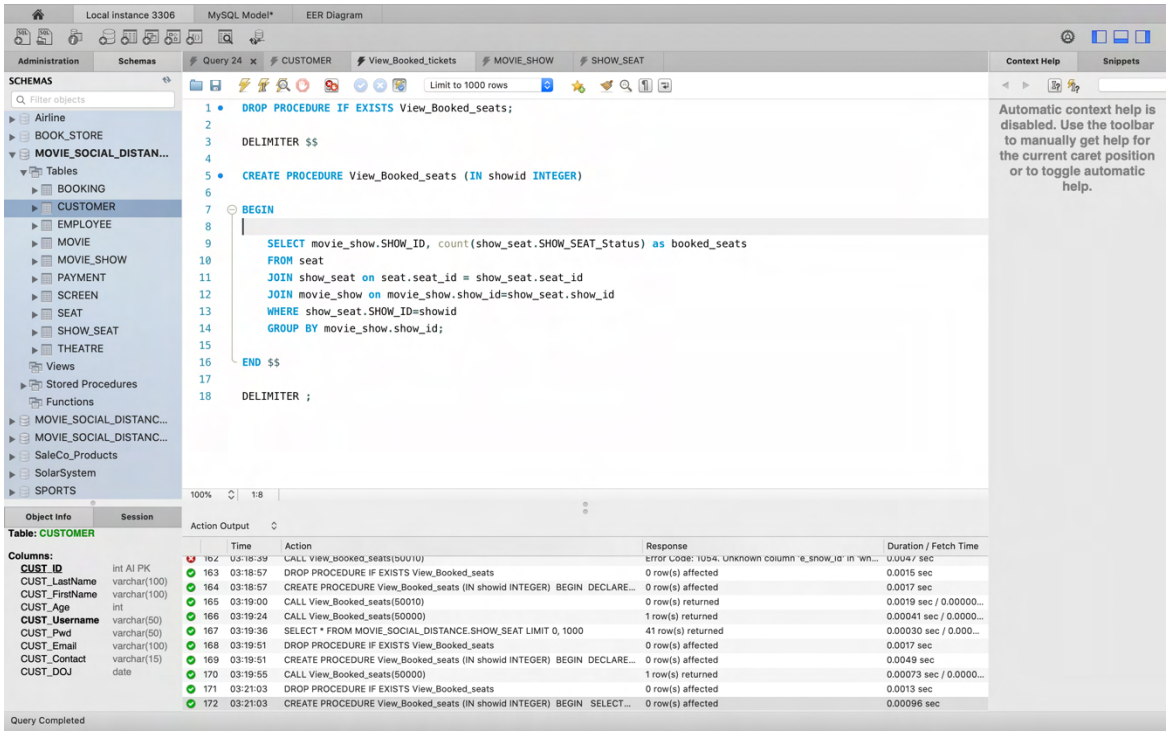

Local instance 3306
MySQL Model*
EER Diagram
Query 24
SHOW_SEAT
Booking_seat
Make_Payment
project_table_creation
Theatre_removal
SCHEMAS
Airline
BOOK_STORE
MOVIE_SOCIAL_DISTAN...
Tables
BOOKING
CUSTOMER
EMPLOYEE
MOVIE
MOVIE_SHOW
PAYMENT
SCREEN
SEAT
SHOW_SEAT
THEATRE
Views
Stored Procedures
Functions
SaleCo_Products
SolarSystem
SPORTS
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
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
PAYMENT 35
SHOW_SEAT 36
MOVIE_SHOW 37
SEAT 38
SCREEN 39
EMPLOYEE 40
THEATRE 41
Context Help
Snippets
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: 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
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 / 0.0000...
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 / 0.0000...
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 / 0.0000...
225 04:00:30 SELECT * FROM THEATRE LIMIT 0, 1000 2 row(s) returned 0.00031 sec / 0.0000...
Query Completed


Local instance 3306
MySQL Model*
EER Diagram
Query 24
SHOW_SEAT
Booking_seat
Make_Payment
project_table_creation
Theatre_removal
SCHEMAS
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
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
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 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
PAYMENT 35
SHOW_SEAT 36
MOVIE_SHOW 37
SEAT 38
SCREEN 39
EMPLOYEE 40
THEATRE 41
Context Help
Snippets
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: 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
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 / 0.0000...
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 / 0.0000...
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 / 0.0000...
225 04:00:30 SELECT * FROM THEATRE LIMIT 0, 1000 2 row(s) returned 0.00031 sec / 0.0000...
Query Completed

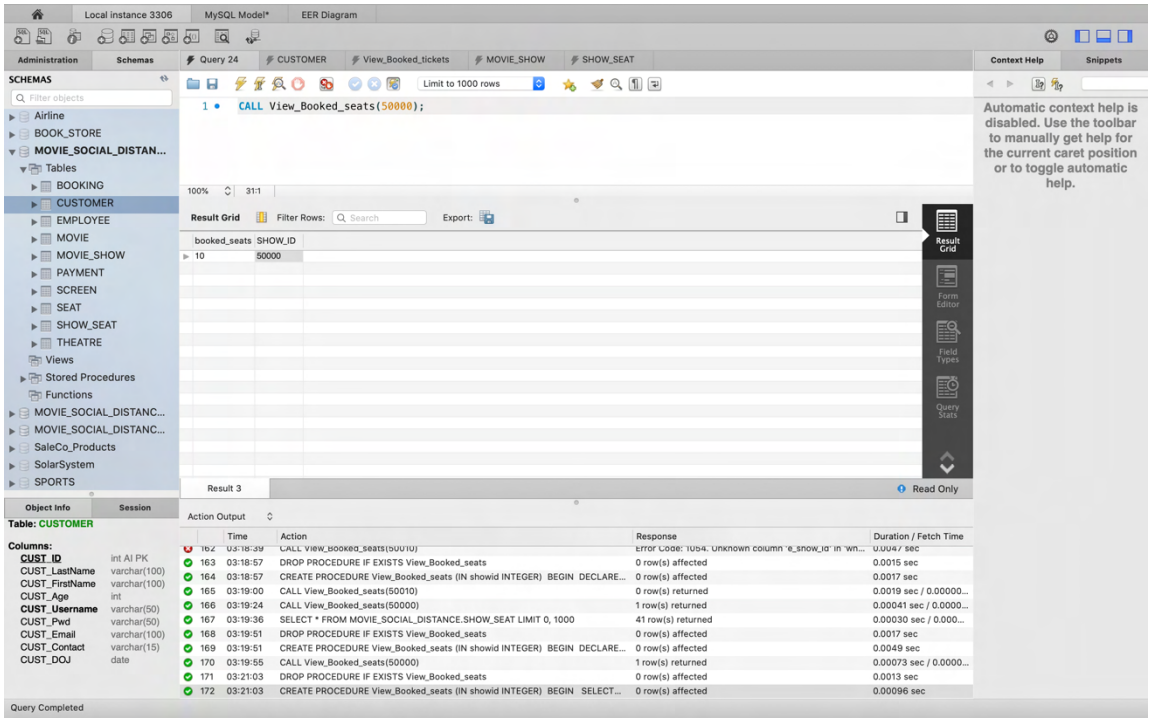

Local instance 3306
MySQL Model*
EER Diagram
Query 24
SHOW_SEAT
Booking_seat
Make_Payment
project_table_creation
Theatre_removal
SCHEMAS
Airline
BOOK_STORE
MOVIE_SOCIAL_DISTAN...
Tables
BOOKING
CUSTOMER
EMPLOYEE
MOVIE
MOVIE_SHOW
PAYMENT
SCREEN
SEAT
SHOW_SEAT
THEATRE
Views
Stored Procedures
Functions
MOVIE_SOCIAL_DISTANC...
MOVIE_SOCIAL_DISTANC...
SaleCo_Products
SolarSystem
SPORTS
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
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
BOOKING 34
PAYMENT 35
SHOW_SEAT 36
MOVIE_SHOW 37
SEAT 38
SCREEN 39
EMPLOYEE 40
Context Help
Snippets
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: 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
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 / 0.0000...
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 / 0.0000...
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 / 0.0000...
225 04:00:30 SELECT * FROM THEATRE LIMIT 0, 1000 2 row(s) returned 0.00031 sec / 0.0000...
Query Completed

View_Booked_seats

Execution of the Stored Procedure:

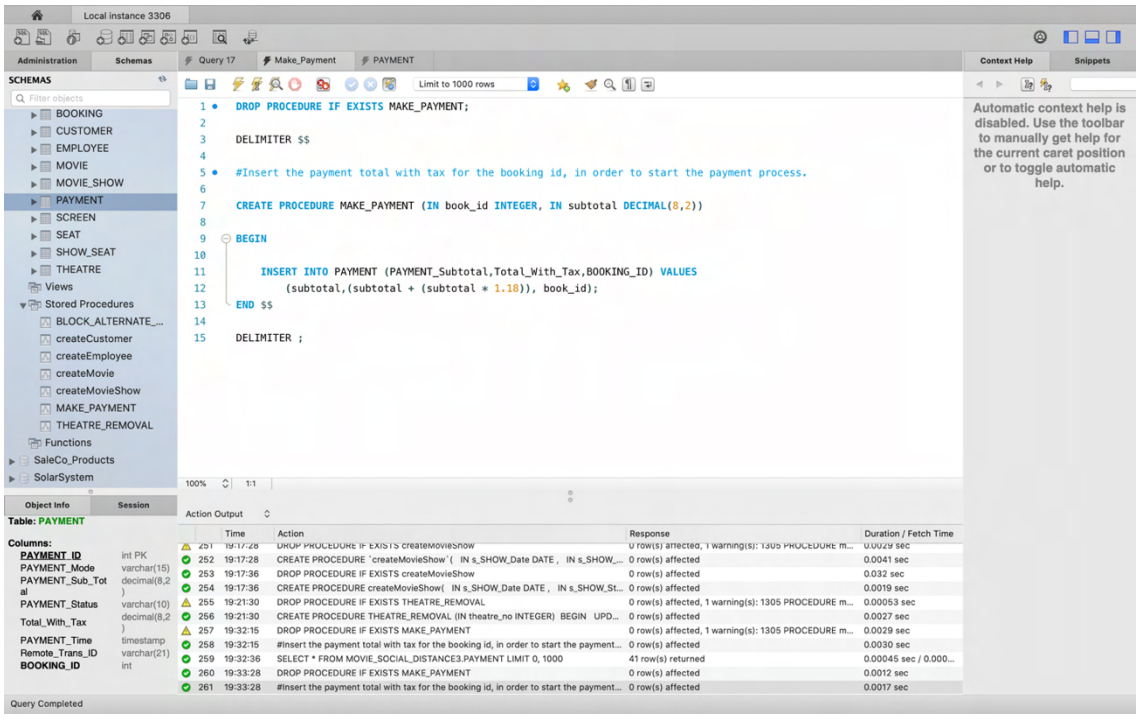


After calling the procedure:

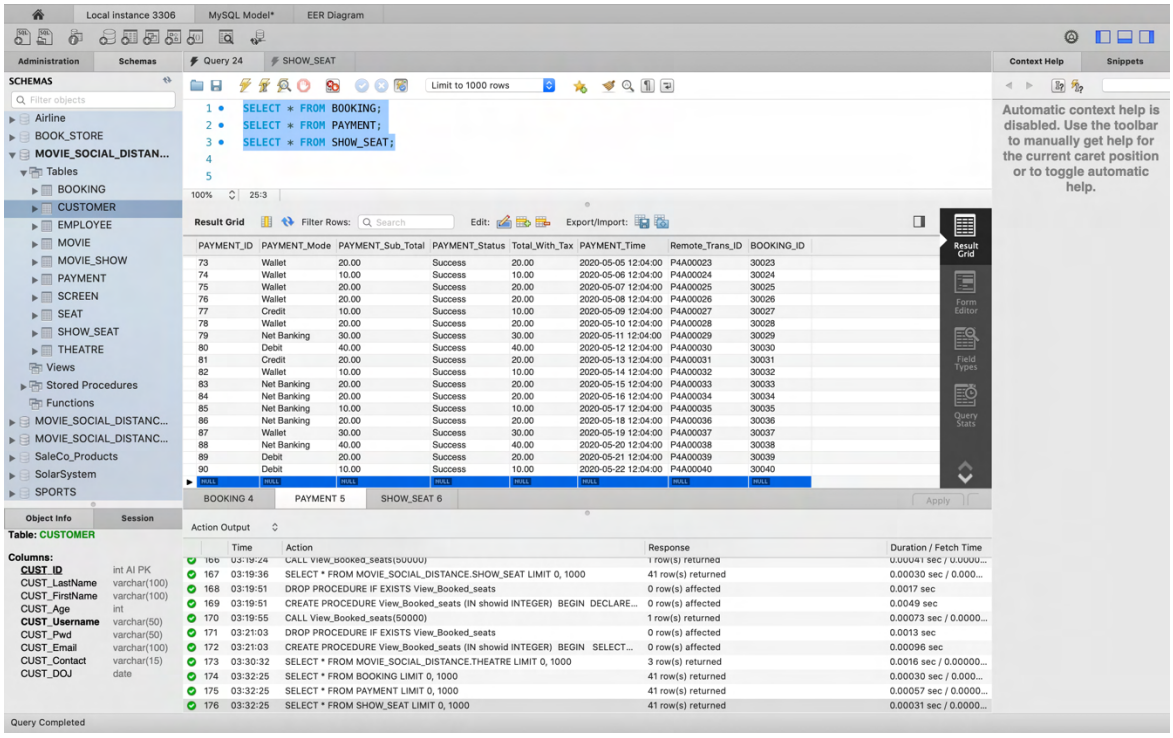


MAKE_PAYMENT

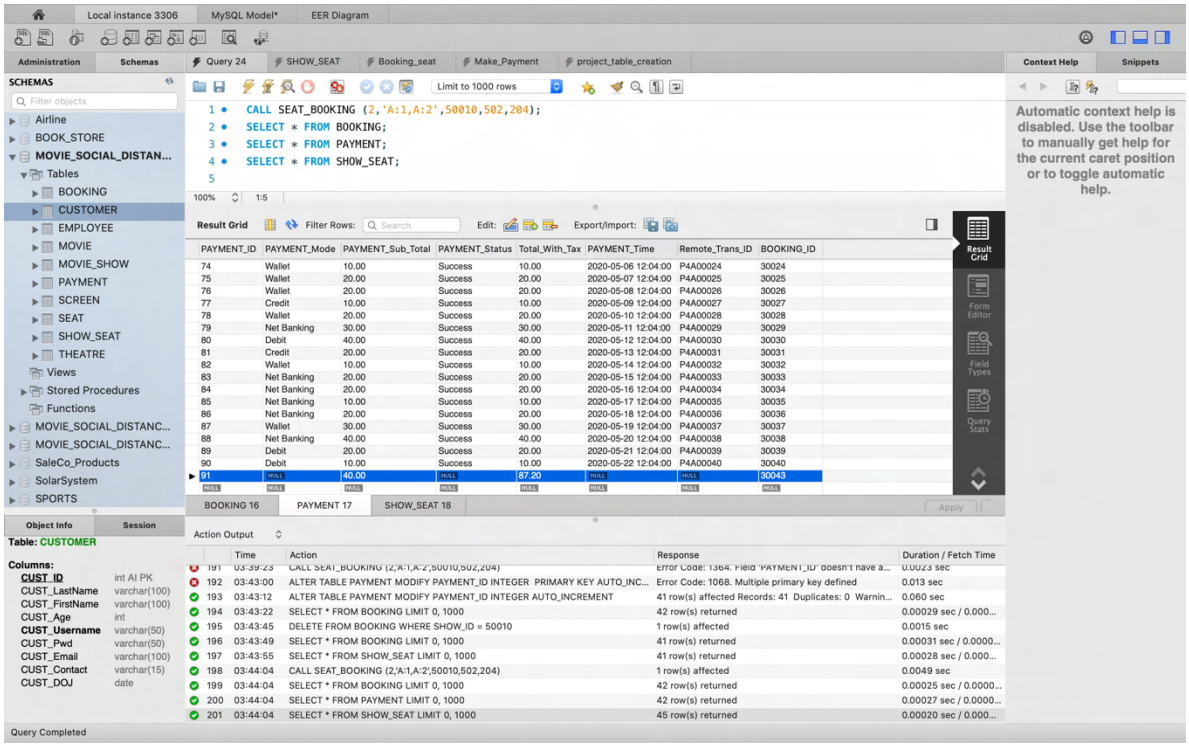
Execution of the Stored Procedure:



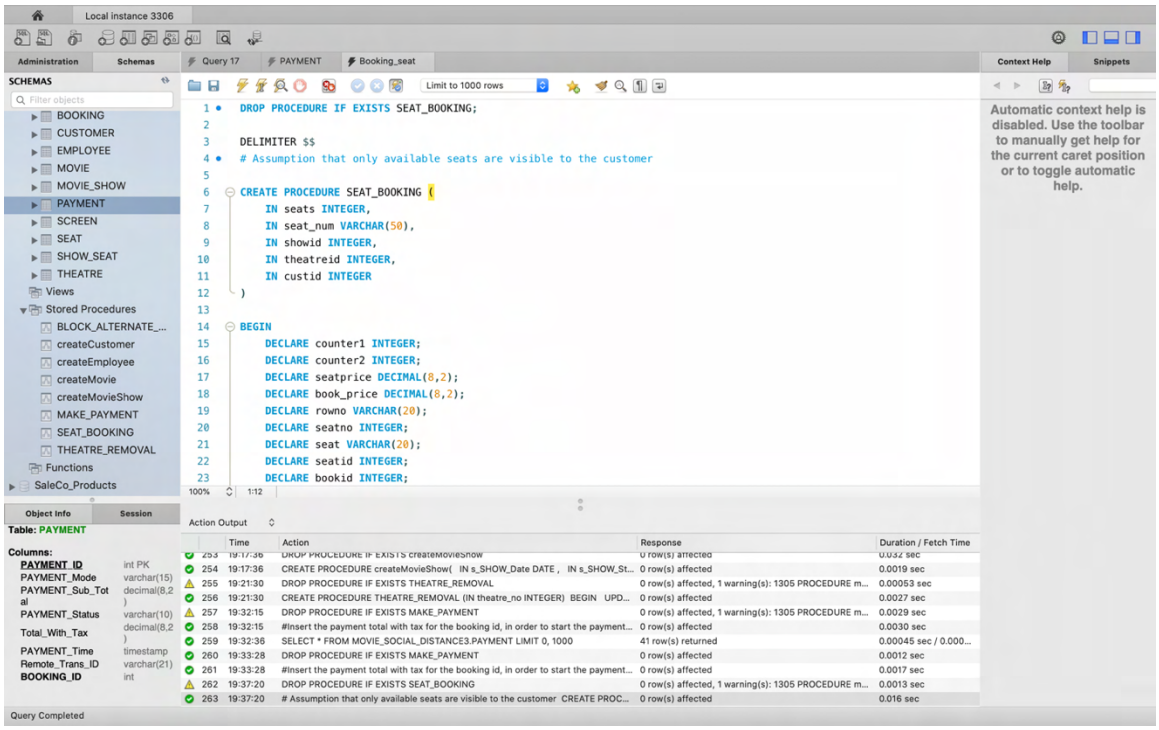
Before calling the procedure



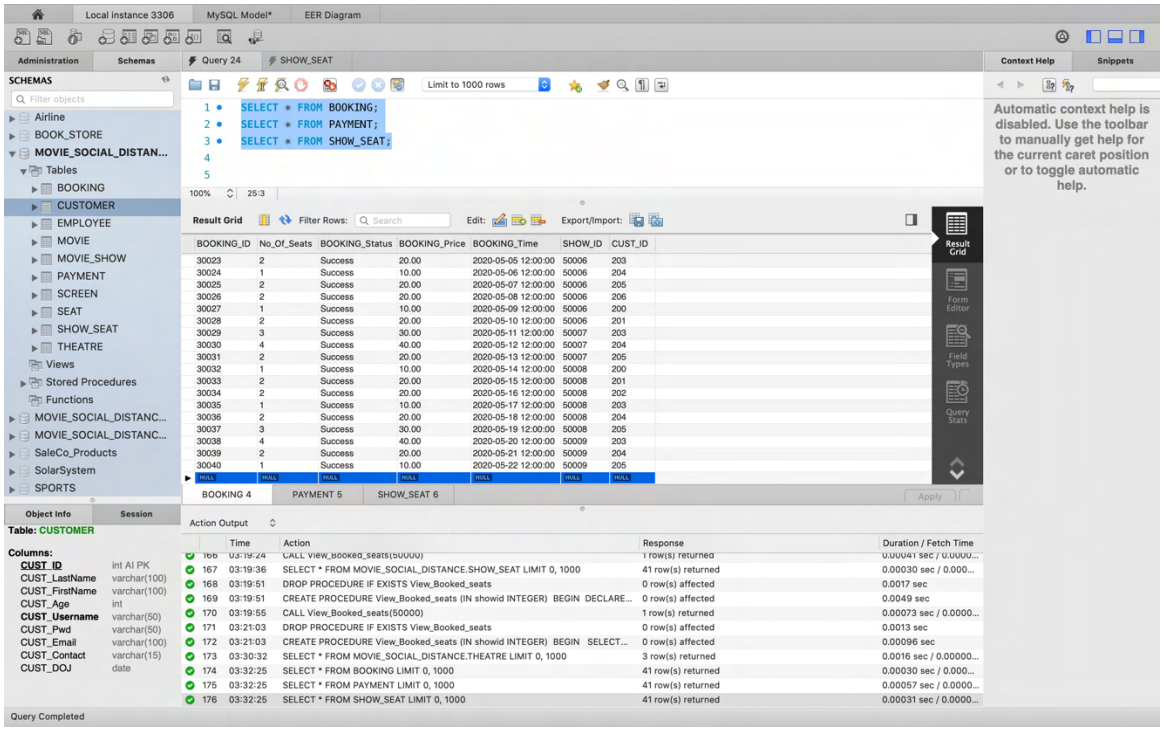
After calling the procedure

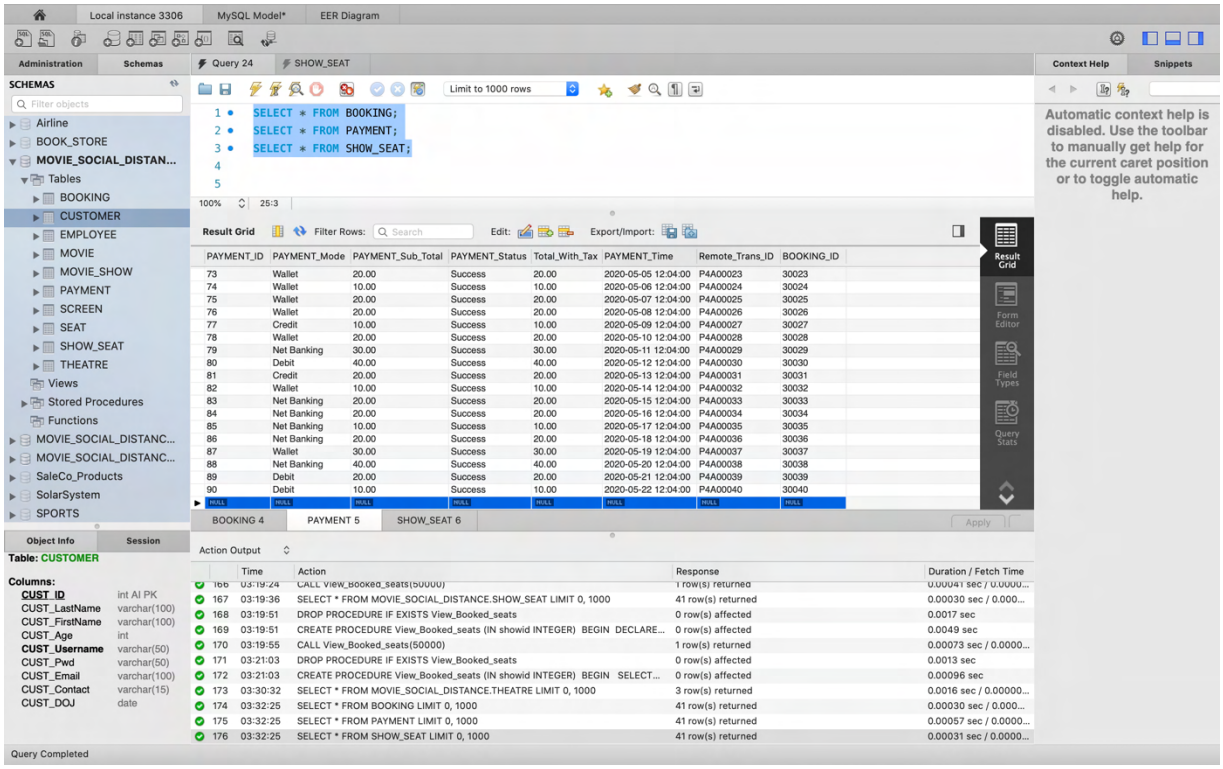


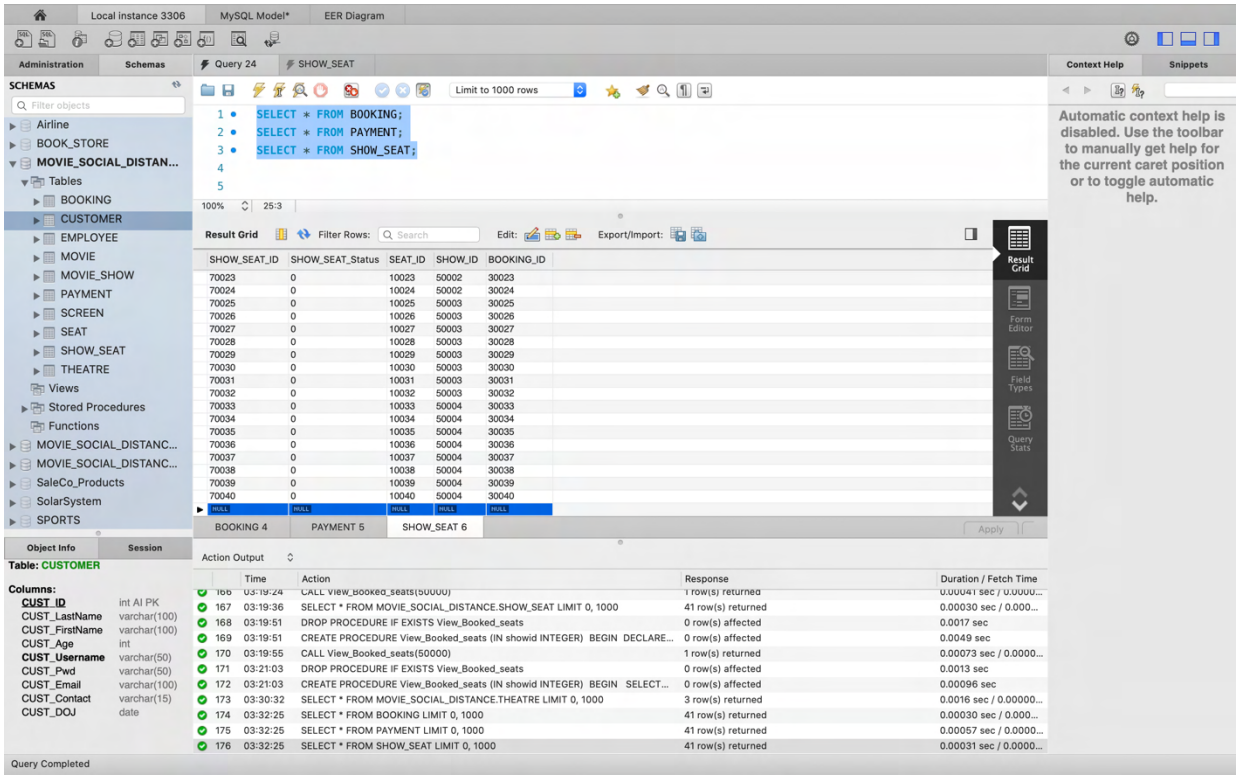
SEAT_BOOKING



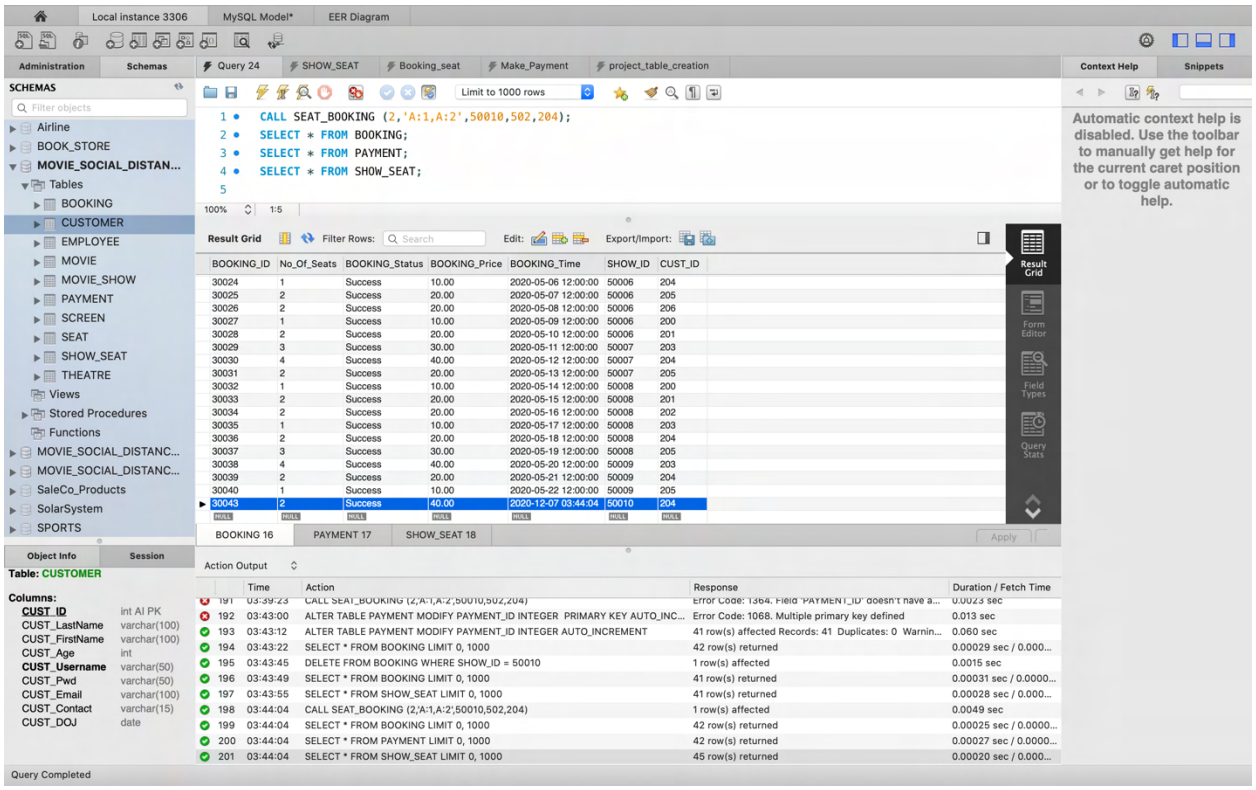
Before calling the procedure

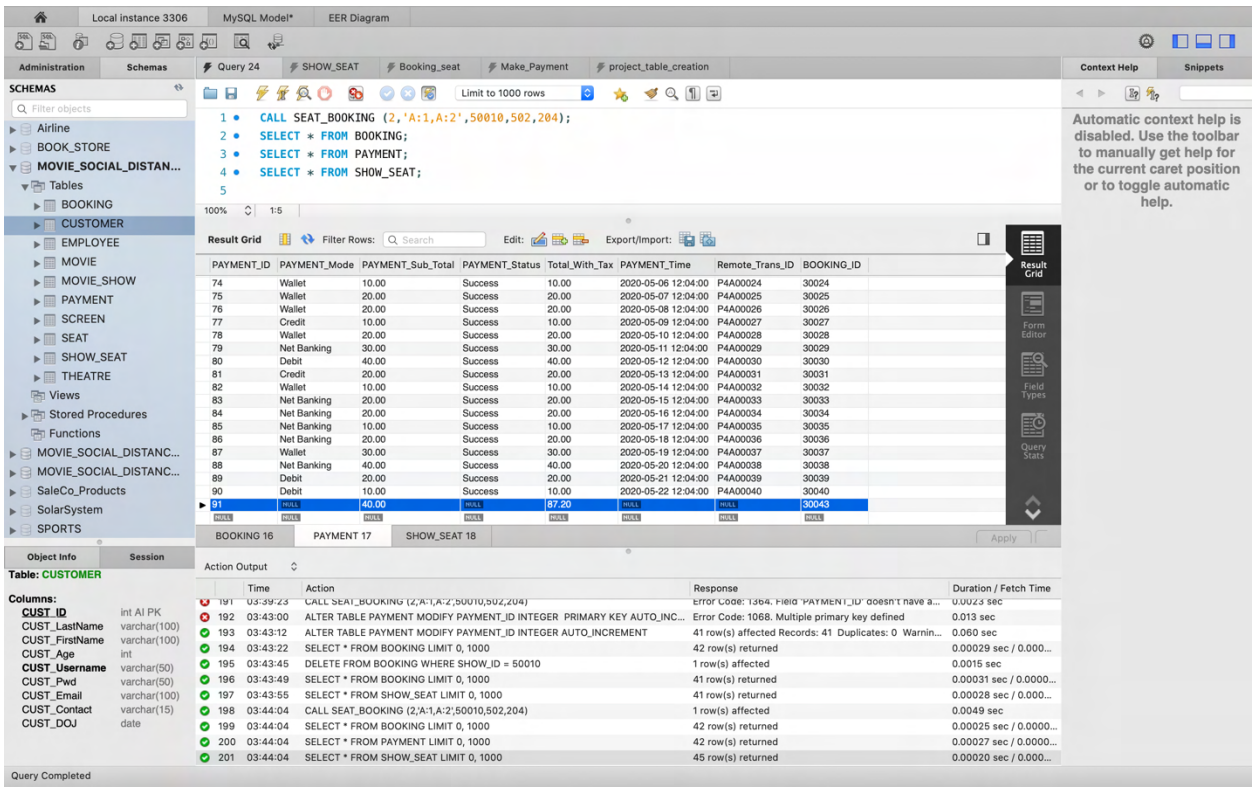


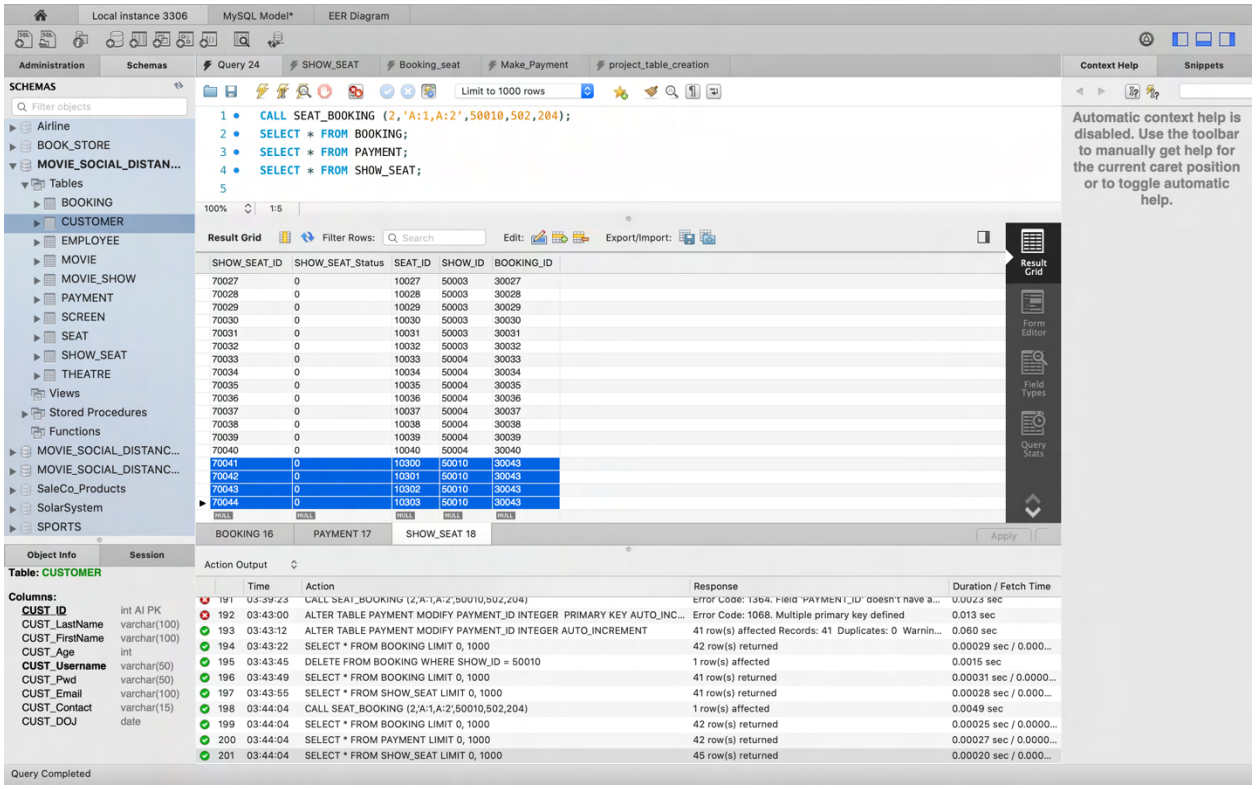

Local instance 3306
MySQL Model*
EER Diagram
Administration
Schemas
Query 24
SHOW_SEAT
Context Help
Snippets
SCHEMAS
Filter objects
Airline
BOOK_STORE
MOVIE_SOCIAL_DISTAN...
Tables
BOOKING
CUSTOMER
EMPLOYEE
MOVIE
MOVIE_SHOW
PAYMENT
SCREEN
SEAT
SHOW_SEAT
THEATRE
Views
Stored Procedures
Functions
MOVIE_SOCIAL_DISTANC...
MOVIE_SOCIAL_DISTANC...
SaleCo_Products
SolarSystem
SPORTS
Limit to 1000 rows
1 SELECT * FROM BOOKING;
2 SELECT * FROM PAYMENT;
3 SELECT * FROM SHOW_SEAT;
4
5
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
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 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
BOOKING 4
PAYMENT 5
SHOW_SEAT 6
Apply
Result Grid
Form Editor
Field Types
Query Stats
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
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
173 03:30:32 SELECT * FROM MOVIE_SOCIAL_DISTANCE.THEATRE LIMIT 0, 1000 3 row(s) returned 0.0016 sec / 0.00000...
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
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.


Local instance 3306
MySQL Model*
EER Diagram
Administration
Schemas
Query 24
SHOW_SEAT
Context Help
Snippets
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
Limit to 1000 rows
1 SELECT * FROM BOOKING;
2 SELECT * FROM PAYMENT;
3 SELECT * FROM SHOW_SEAT;
4
5
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
SHOW_SEAT_ID SHOW_SEAT_Status SEAT_ID SHOW_ID BOOKING_ID
70023 0 10023 50002 30023
70024 0 10024 50002 30024
70025 0 10025 50003 30025
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 50004 30039
70040 0 10040 50004 30040
BOOKING 4
PAYMENT 5
SHOW_SEAT 6
Apply
Result Grid
Form Editor
Field Types
Query Stats
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
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
173 03:30:32 SELECT * FROM MOVIE_SOCIAL_DISTANCE.THEATRE LIMIT 0, 1000 3 row(s) returned 0.0016 sec / 0.00000...
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
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

After calling the procedure







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);
```

```
        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 $$
```

```
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;
```

```
        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 is 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;
```

```
    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
```

```
            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 =
```

```
SUBSTRING_INDEX(SUBSTRING_INDEX(seat_num,',',cou
nter2), ',', -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
```

```
                    (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;
```

```
# 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);
```

```
                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.