Book My Show -

12.02.2019

—

Group members:

G. Badrinath Reddy	-	111601006
Vishnu Teja Vallala	-	111601028
NVS. Vishnu Kanth Reddy	-	111601012

Instructed By:

Sahely Bhadra

Mrinal Kanti Das

Contributions:

Week 5:(Graphical Interface)

Front End : Vishnu Teja Vallala

Backend : NVS Vishnu Kanth Reddy,Badrinath Reddy Gujjula

Week 6:

BCNF Normalization: Equal contribution from all the team members

Login Page: NVS Vishnu Kanth Reddy,Badrinath Reddy Gujjula

Registration Page and Logout: Vishnu Teja Vallala

Customer HomePage, My Account: NVS Vishnu Kanth Reddy, Badrinath Reddy Gujjula

Week 7:

Badrinath Reddy Gujjula:

Created procedure booking_history,host_theatre,screen_tickets.

NVS Vishnu Kanth Reddy:

Created View of trending,coming_soon,filter_movie.

Vishnu Teja Vallala:

Created Host login,logout ,My Account,Registration page.

Project Plan

The Project will try to mimic the famous online booking app bookmyshow.

Our Database mainly contains two types of users :

- Movie Theatre Owner
- Ticket Purchaser

User Level Organization:

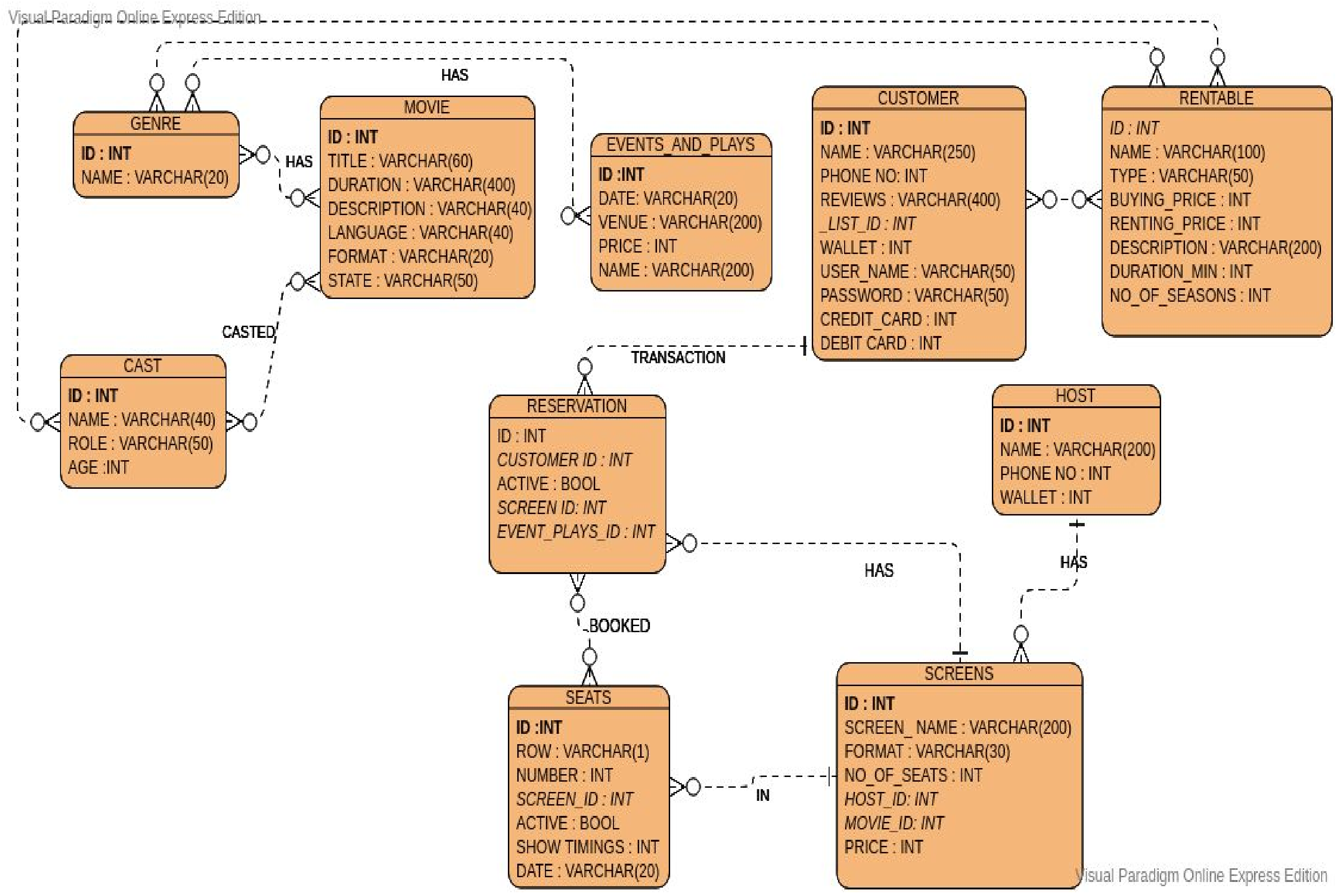
Movie Theatre Owners:

- They could register and enroll their movie and theatre.
- They can also fill in the details of the movie like cast , director, language, ticket cost
- There can be multiple screens in a single theatre, so they can give different movies for different screens for different show timings.

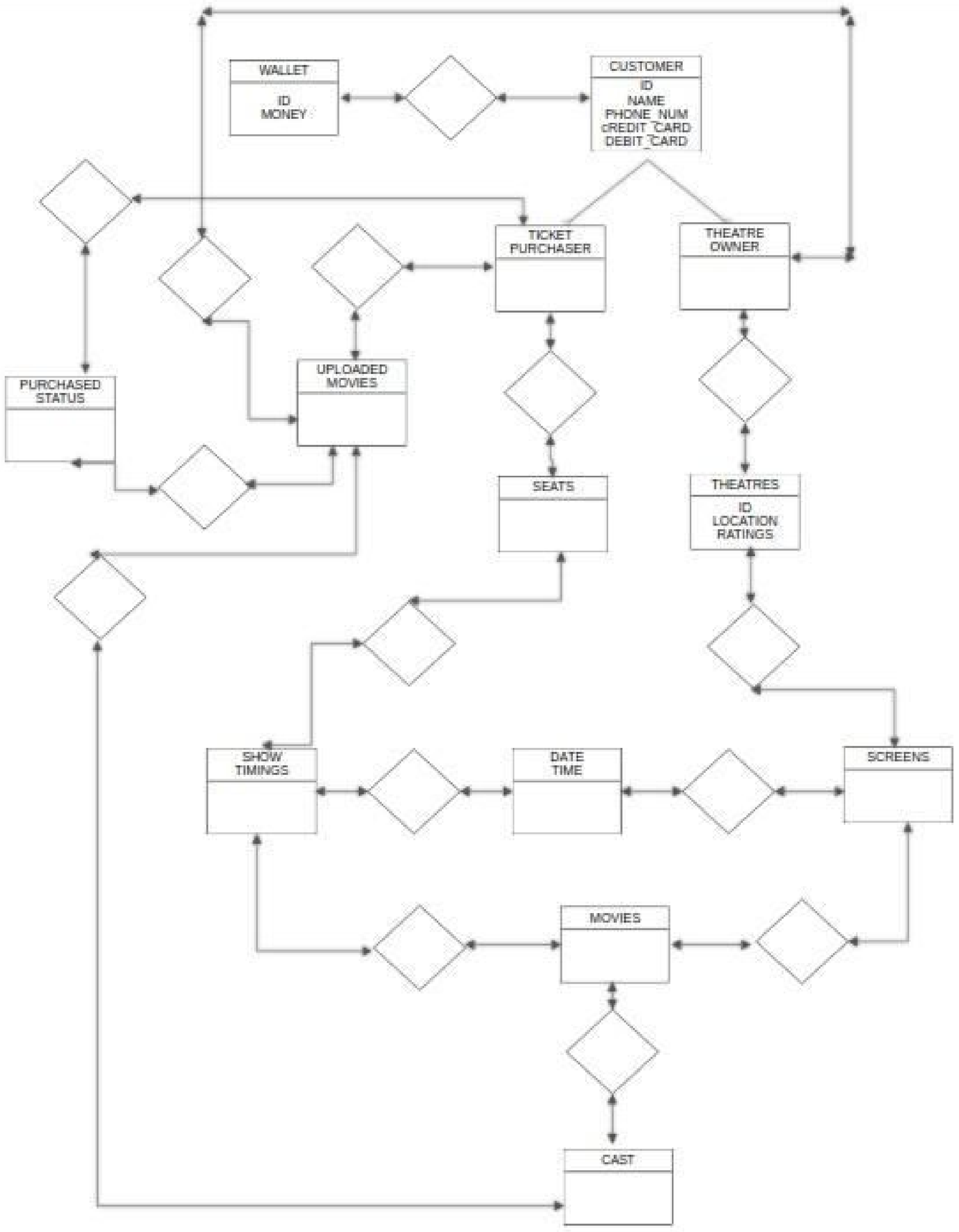
Ticket Purchaser:

- They can either register or enter as a guest.
- If the user registers, then he/she will have a e-wallet and proceed to book tickets. They can update their profile and add money to their wallets and can use it to book tickets.
- The user can choose a movie and a theatre based on various filters like movie name, place, theatre name, cast, director etc..
- They can rent as well as buy movies online with the money in their wallet. The rented movies will last for a particular number of days.

Updated Design of ERD as a Developer:


Visual Paradigm Online Express Edition
HAS
GENRE
ID : INT
NAME : VARCHAR(20)
HAS
MOVIE
ID : INT
TITLE : VARCHAR(60)
DURATION : VARCHAR(400)
DESCRIPTION : VARCHAR(40)
LANGUAGE : VARCHAR(40)
FORMAT : VARCHAR(20)
STATE : VARCHAR(50)
EVENTS_AND_PLAYS
ID :INT
DATE: VARCHAR(20)
VENUE : VARCHAR(200)
PRICE : INT
NAME : VARCHAR(200)
CUSTOMER
ID : INT
NAME : VARCHAR(250)
PHONE NO: INT
REVIEWS : VARCHAR(400)
_LIST_ID : INT
WALLET : INT
USER_NAME : VARCHAR(50)
PASSWORD : VARCHAR(50)
CREDIT_CARD : INT
DEBIT CARD : INT
RENTABLE
ID : INT
NAME : VARCHAR(100)
TYPE : VARCHAR(50)
BUYING_PRICE : INT
RENTING_PRICE : INT
DESCRIPTION : VARCHAR(200)
DURATION_MIN : INT
NO_OF_SEASONS : INT
CASTED
CAST
ID : INT
NAME : VARCHAR(40)
ROLE : VARCHAR(50)
AGE :INT
TRANSACTION
RESERVATION
ID : INT
CUSTOMER ID : INT
ACTIVE : BOOL
SCREEN ID: INT
EVENT_PLAYS_ID : INT
HOST
ID : INT
NAME : VARCHAR(200)
PHONE NO : INT
WALLET : INT
HAS
HAS
BOOKED
SEATS
ID :INT
ROW : VARCHAR(1)
NUMBER : INT
SCREEN_ID : INT
ACTIVE : BOOL
SHOW TIMINGS : INT
DATE : VARCHAR(20)
IN
SCREENS
ID : INT
SCREEN_ NAME : VARCHAR(200)
FORMAT : VARCHAR(30)
NO_OF_SEATS : INT
HOST_ID: INT
MOVIE_ID: INT
PRICE : INT
Visual Paradigm Online Express Edition

PREVIOUS Design of ERD as a Developer:



Roles:

- Customer:
    - multiple users can have the role of Customer
    - They can only book movies,events and rent movies and have access to their own history and wallet
    - These users cannot host movies nor can look into other users info.
- Host :
    - multiple hosts can have the role of Host
    - They can host their movie or events and have access on their screens and theatre
    - Can sell their movies or series through rentables
- Admin : badri,vishnu,v_vishnu,root..

Views:

- Trending:
    - Based on movie release date, we can decide whether a movie is trending, coming soon etc...
- Filter Movies to book tickets By :
    - Cast,Language,Format,Show Timings and Date,Location,Genre,Theatre
- Filter Events and plays:
    - Location,Price,Date,Genre
- Show Customer's Booking history:
- Filter Rentable Movies and Series by:

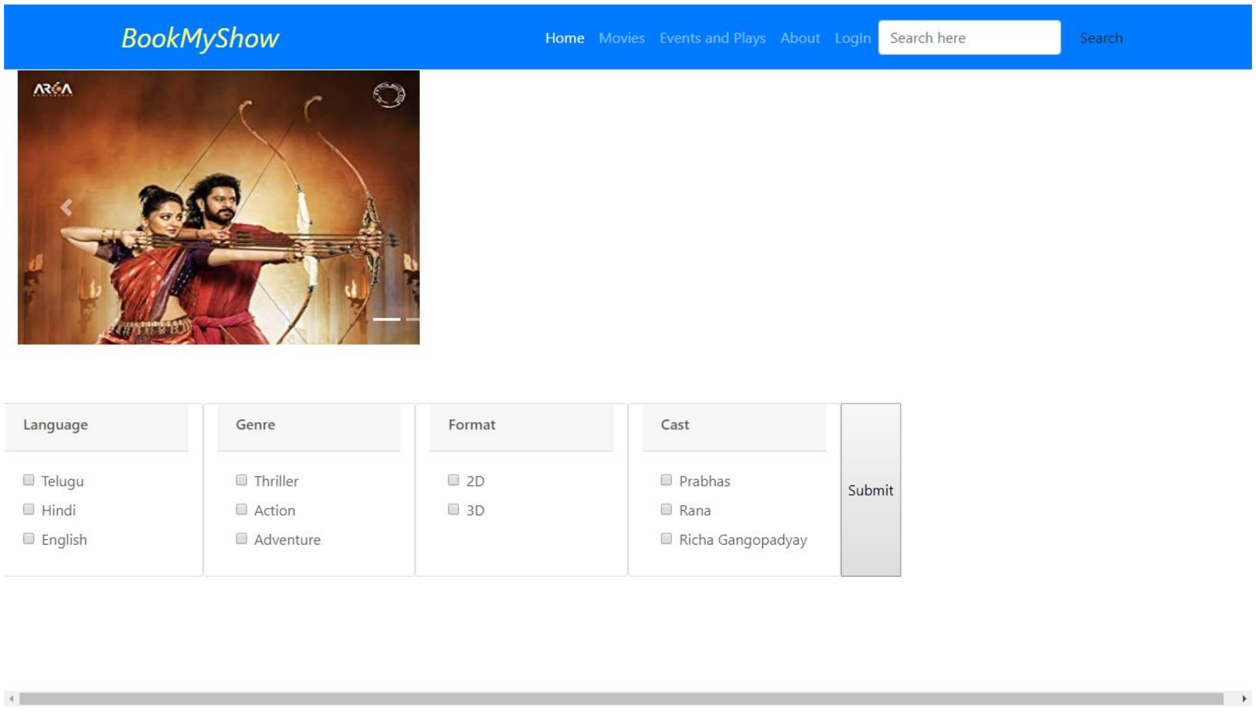
  - Movie name, tv series name, renting price, buying price,format,duration

Triggers:

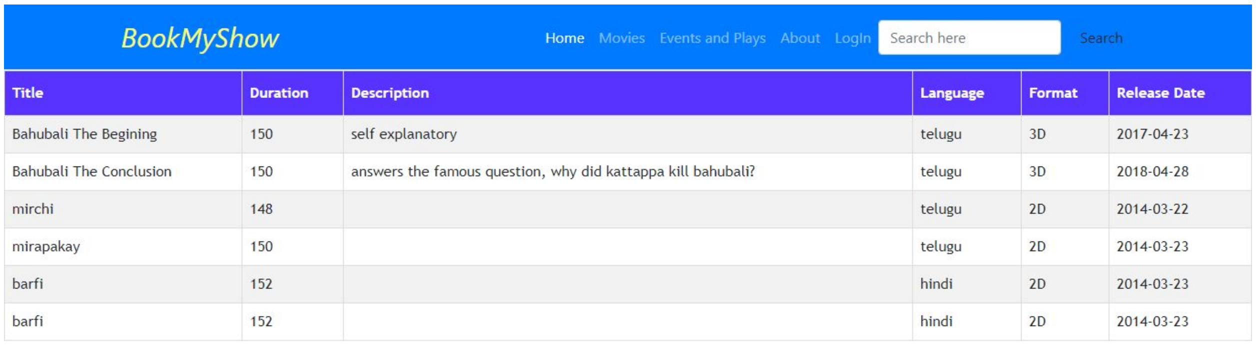
- before_reservation_update:
  - When a customer confirms a booking, corresponding ticket should be blocked and ticket price should be deducted from the customer wallet.
  - If the customer wallet has insufficient funds the same will be conveyed.
  - The deducted amount will be accounted to the host account and some predefined percentage of money will be added automatically to the administrator from host for each booking
- Before_screens_update:
  - When a host updates existing movie in a particular screen and if that movie is not in the movie table it is added to the table and cast is also updated if needed.
- Before_reservation(cancel)_update:
  - When a customer cancels a booking, the reservation is made invalid and seat is unblocked and the customer is refunded half of the ticket price from host account.
  - And the cancellation is possible only at least one hour before the show timing.

Graphical Interface:

Home Page:

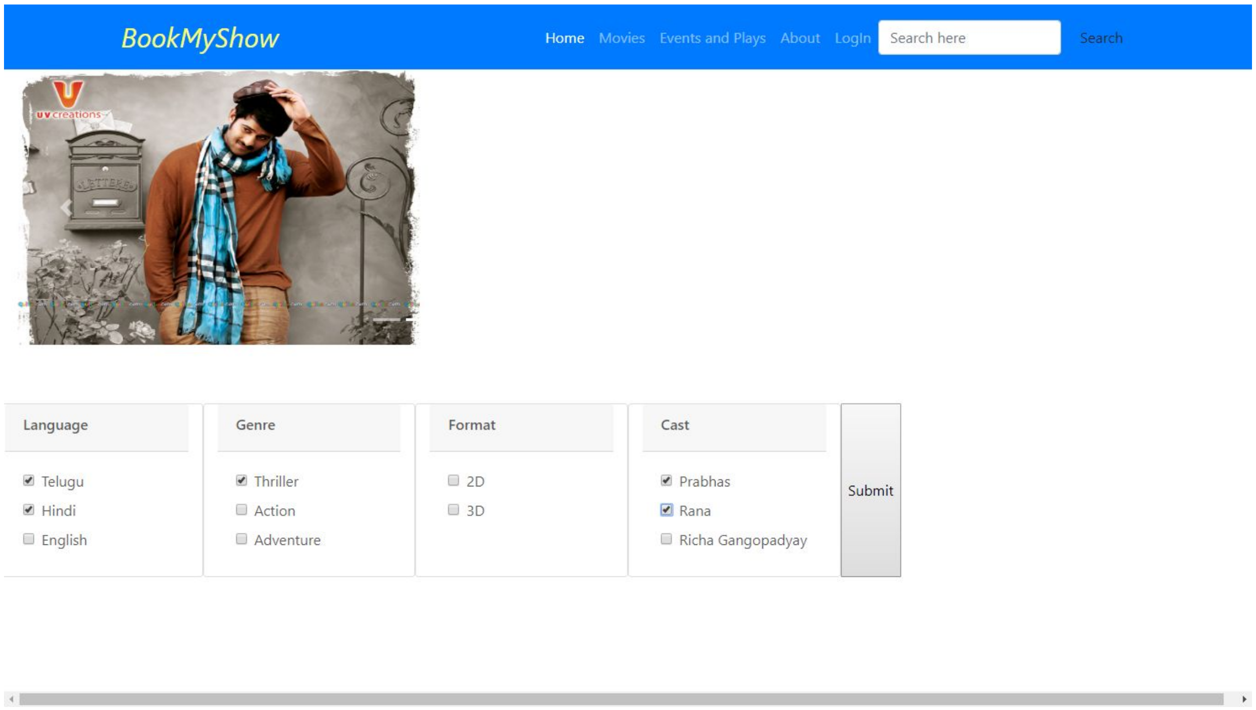


Clicking on the Movies Button in Navigation Bar:(Displays all the movies in the DB)



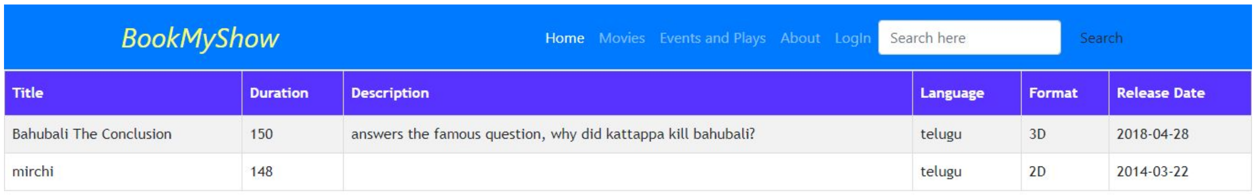
Title	Duration	Description	Language	Format	Release Date
Bahubali The Begining	150	self explanatory	telugu	3D	2017-04-23
Bahubali The Conclusion	150	answers the famous question, why did kattappa kill bahubali?	telugu	3D	2018-04-28
mirchi	148		telugu	2D	2014-03-22
mirapakay	150		telugu	2D	2014-03-23
barfi	152		hindi	2D	2014-03-23
barfi	152		hindi	2D	2014-03-23

Selecting Multiple Filters :



Language	Genre	Format	Cast
✓ Telugu	✓ Thriller	2D	✓ Prabhas
✓ Hindi	Action	3D	✓ Rana
English	Adventure		Richa Gangopadyay

Shows all the movies that satisfies the selected filters



Title	Duration	Description	Language	Format	Release Date
Bahubali The Conclusion	150	answers the famous question, why did kattappa kill bahubali?	telugu	3D	2018-04-28
mirchi	148		telugu	2D	2014-03-22

BCNF Normalization:

- A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F+ of the form

  $$\alpha \rightarrow \beta$$

  where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R

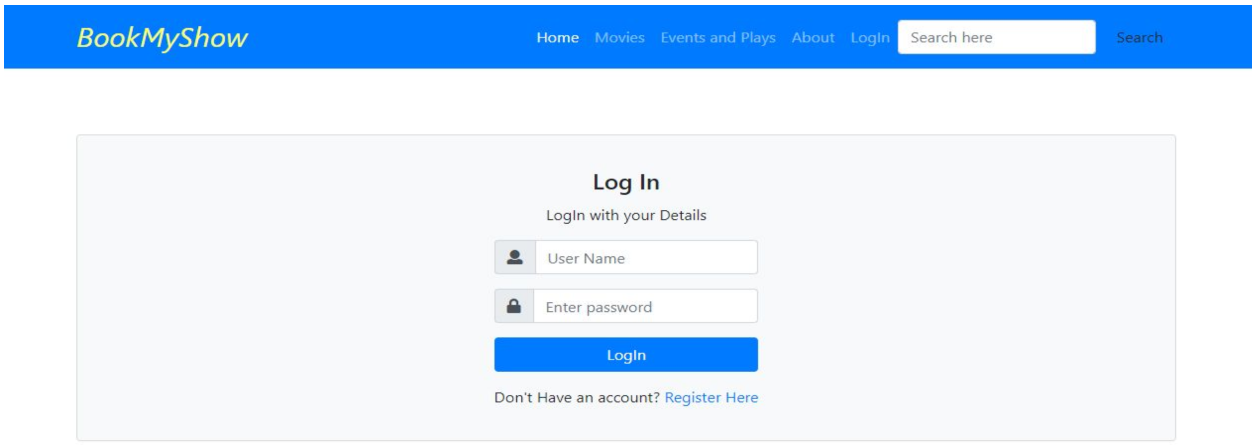
- Our design of the database is such that every column of the table is only dependent on the primary key of that table.
- For example consider the table customer(ID,name,phone_no,wallet,user_name,password,credit_card,debit_card) each and every row is dependent on column ID i.e it is the primary key
- If there is a functional dependency $\alpha \rightarrow \beta$ in a table where α ,β are columns of that table, then our design inherently included them being in two different tables connected via a foreign key and making sure that column being referenced is kept as primary key in the other table.
- For Example consider these two tables from our database screens(screen_id,screen_name,format,no_of_seats,host_id,movie_id,theatre) and movie(movie_id,title,duration,description,language,format,release_date). Now one movie can be screened in multiple screens ,but given a screen ,a unique movie is being played in that screen at any given moment. So this is clearly a case of functional dependency screen_id -> movie_id. So we have put them in two tables with the two columns screen_id and movie_id being the primary keys of their respective tables.
- If there are two columns α and β (primary keys) such that one value in α can imply multiple values in β and one value in β can imply multiple values in α, then they can neither be called as functionally dependent nor independent. In such cases to reduce the data stored and reduce the redundancy, we just stored α and β (only the primary keys) in a separate relational table. So the redundancy corresponding to these two columns still exist, but the redundancy with other columns would not exist anymore.
- For example consider the two tables casts(cast_id,name,role,age,gender) and movie(ID,title,duration,description,language,format,release_date). Now one cast member ,uniquely determined by the cast_id, can be a part of multiple movies, similarly one movie,uniquely determined by the movie_id, can have multiple cast members. So they are not dependent and were these two tables had been together

(without the cast_id), even then the table would be in BCNF form but there would be data redundancy.

- So all our tables are already in BCNF form.

Implement the Roles and Privileges:

Customer Login:



BookMyShow

Home Movies Events and Plays About LogIn Search here Search

Log In

LogIn with your Details

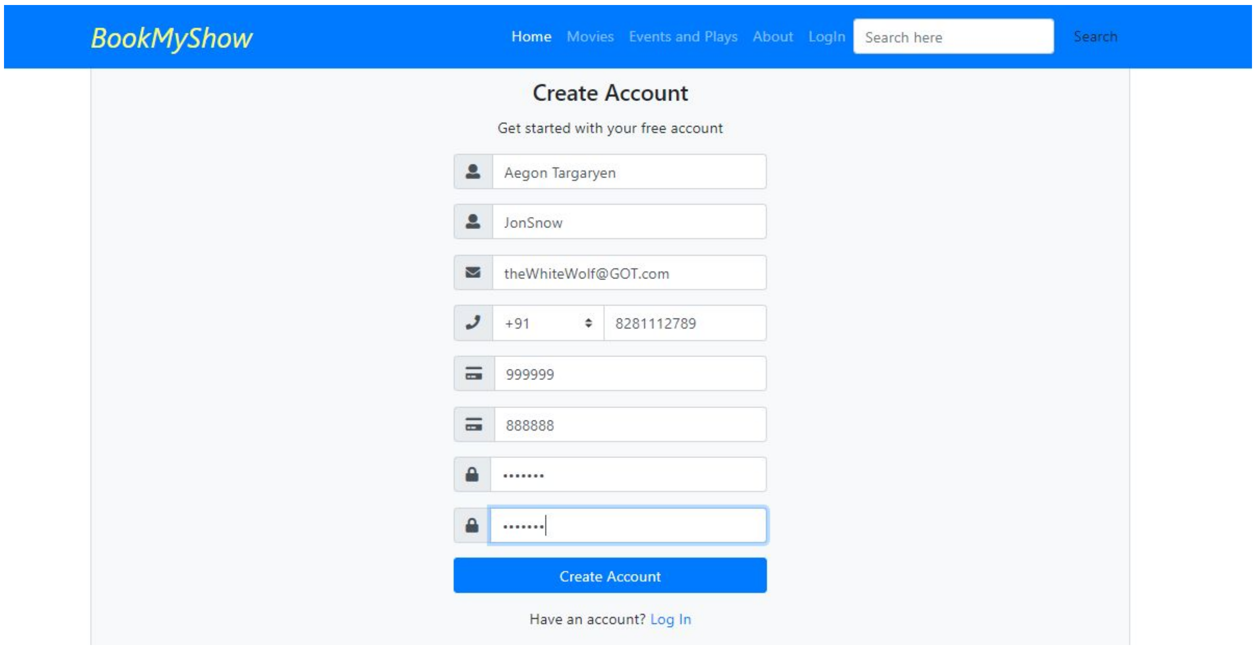
User Name

Enter password

LogIn

Don't Have an account? Register Here

Customer Registration:



BookMyShow

Home Movies Events and Plays About LogIn Search here Search

Create Account

Get started with your free account

Aegon Targaryen

JonSnow

theWhiteWolf@GOT.com

+91 8281112789

999999

888888

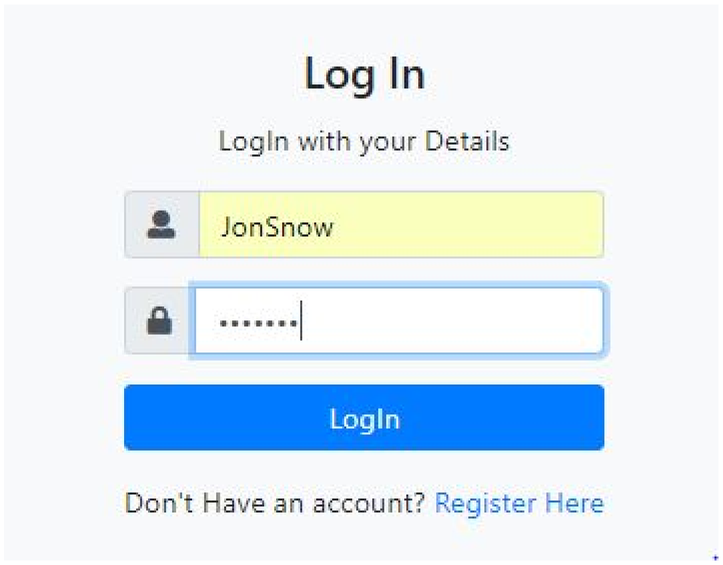
•••••••

•••••••

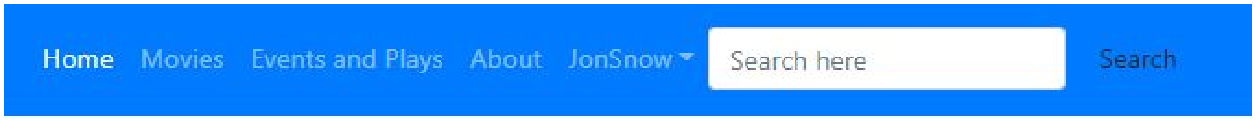
Create Account

Have an account? Log In

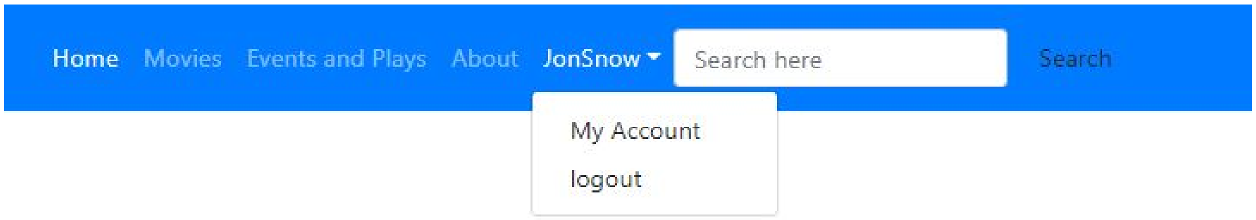
Logging in as a user



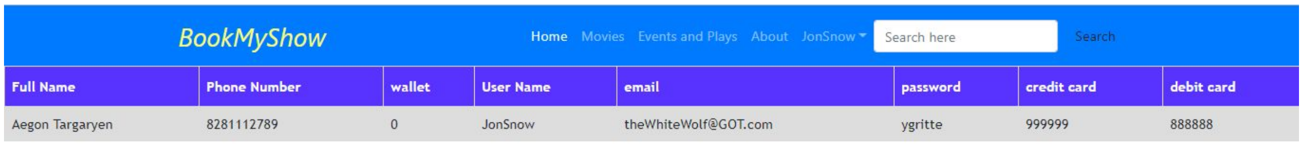
Logged in as JonSnow



Clicking on JonSnow:

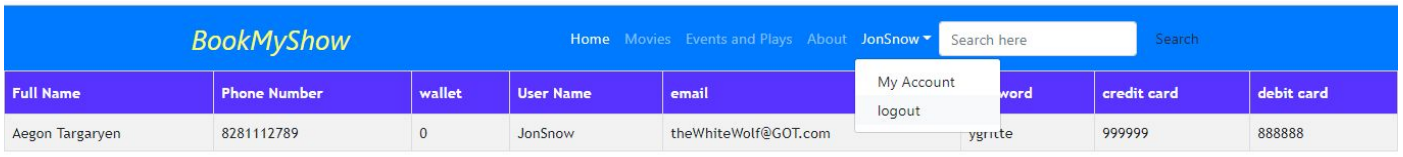


Clicking on My Account:



Full Name	Phone Number	wallet	User Name	email	password	credit card	debit card
Aegon Targaryen	8281112789	0	JonSnow	theWhiteWolf@GOT.com	ygritte	999999	888888

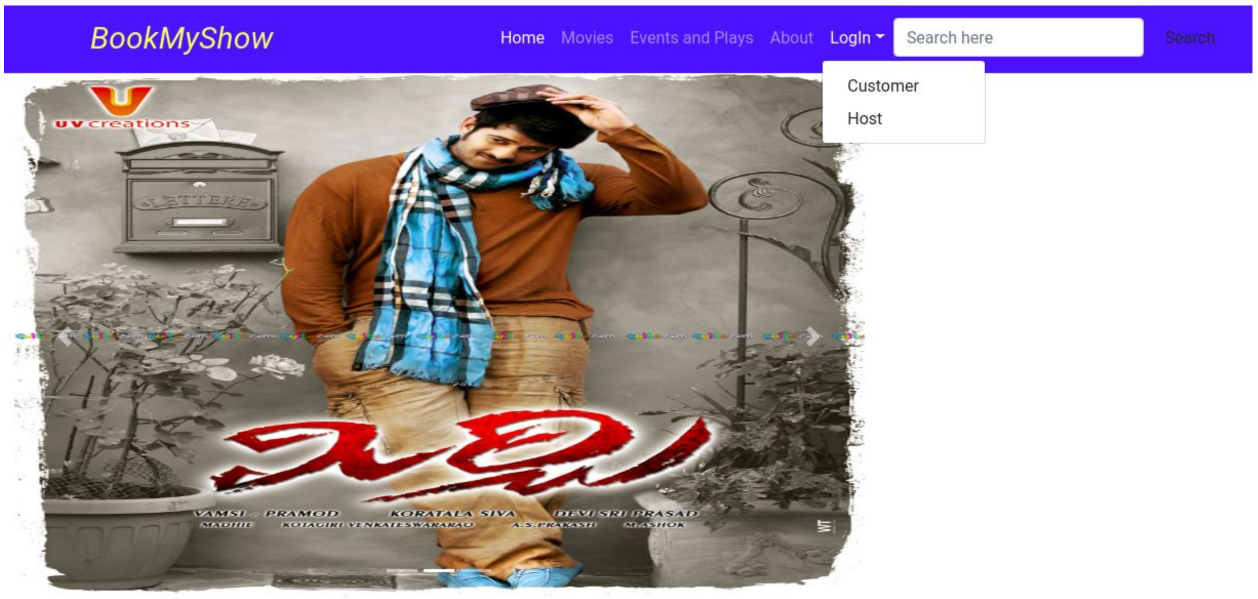
At the same time we can log in as another user from a new tab and their corresponding information is shown when clicked on My Account.

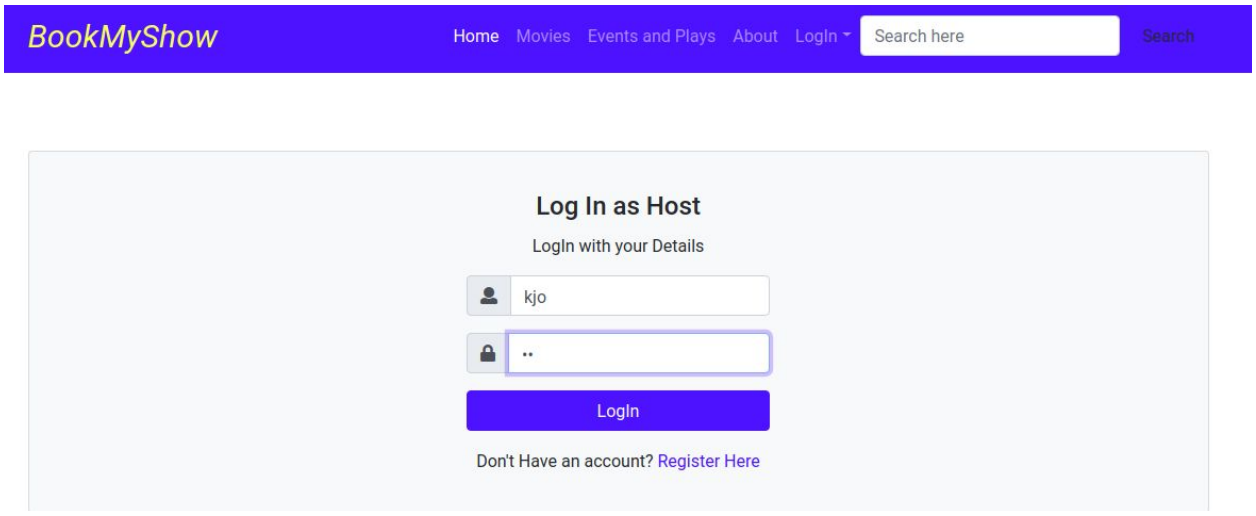


Full Name	Phone Number	wallet	User Name	email	word	credit card	debit card
Aegon Targaryen	8281112789	0	JonSnow	theWhiteWolf@GOT.com	ygritte	999999	888888

It takes them to the home page.

Logging in as a host





Log In as Host

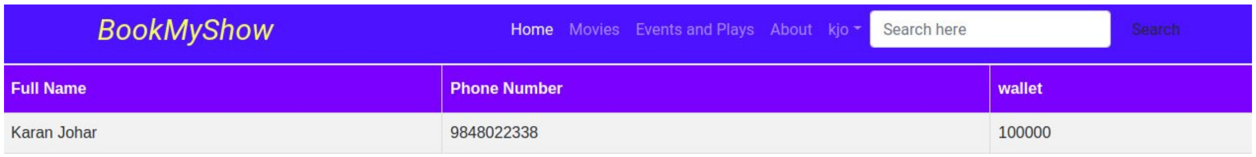
LogIn with your Details

kjo

••

LogIn

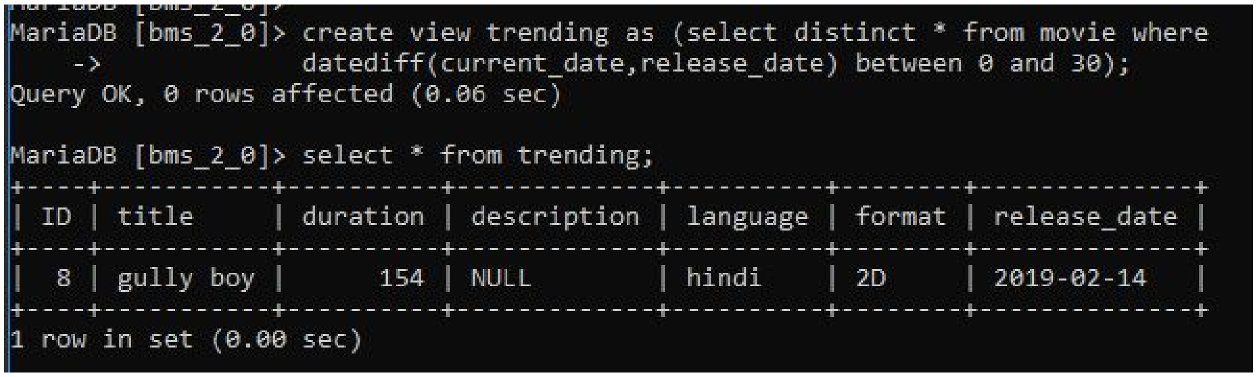
Don't Have an account? Register Here



Full Name	Phone Number	wallet
Karan Johar	9848022338	100000

Creating view trending:

Shows movies which are released in a span of 30 days from current date.

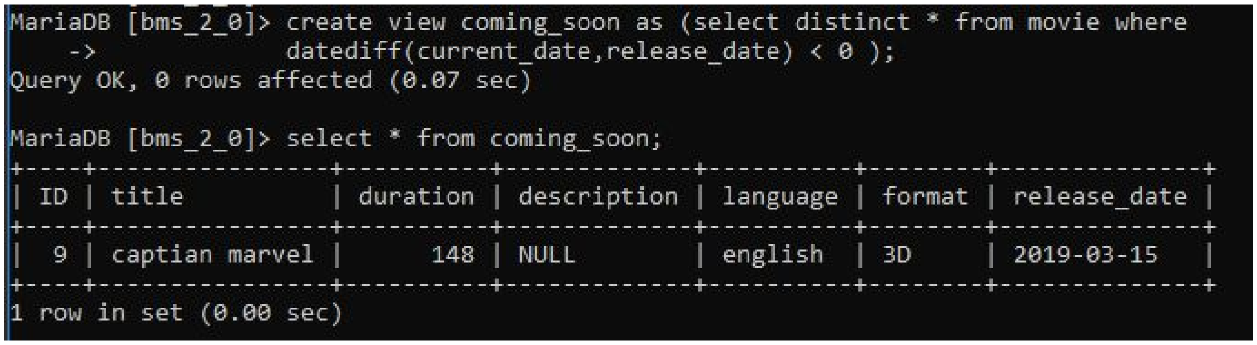


```
MariaDB [bms_2_0]> create view trending as (select distinct * from movie where
    ->             datediff(current_date,release_date) between 0 and 30);
Query OK, 0 rows affected (0.06 sec)

MariaDB [bms_2_0]> select * from trending;
+----+-----------+----------+-------------+----------+--------+--------------+
| ID | title     | duration | description | language | format | release_date |
+----+-----------+----------+-------------+----------+--------+--------------+
|  8 | gully boy |      154 | NULL        | hindi    | 2D     | 2019-02-14   |
+----+-----------+----------+-------------+----------+--------+--------------+
1 row in set (0.00 sec)
```

Creating view ComingSoon:

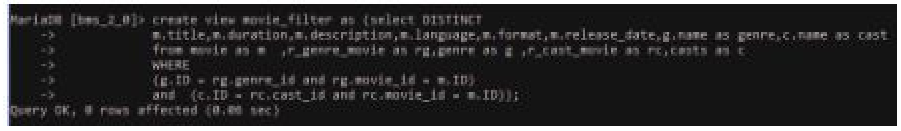
Shows movies which are yet to be released.



```
MariaDB [bms_2_0]> create view coming_soon as (select distinct * from movie where
    ->              datediff(current_date,release_date) < 0 );
Query OK, 0 rows affected (0.07 sec)

MariaDB [bms_2_0]> select * from coming_soon;
+----+----------------+----------+-------------+----------+--------+--------------+
| ID | title          | duration | description | language | format | release_date |
+----+----------------+----------+-------------+----------+--------+--------------+
|  9 | captian marvel |      148 | NULL        | english  | 3D     | 2019-03-15   |
+----+----------------+----------+-------------+----------+--------+--------------+
1 row in set (0.00 sec)
```

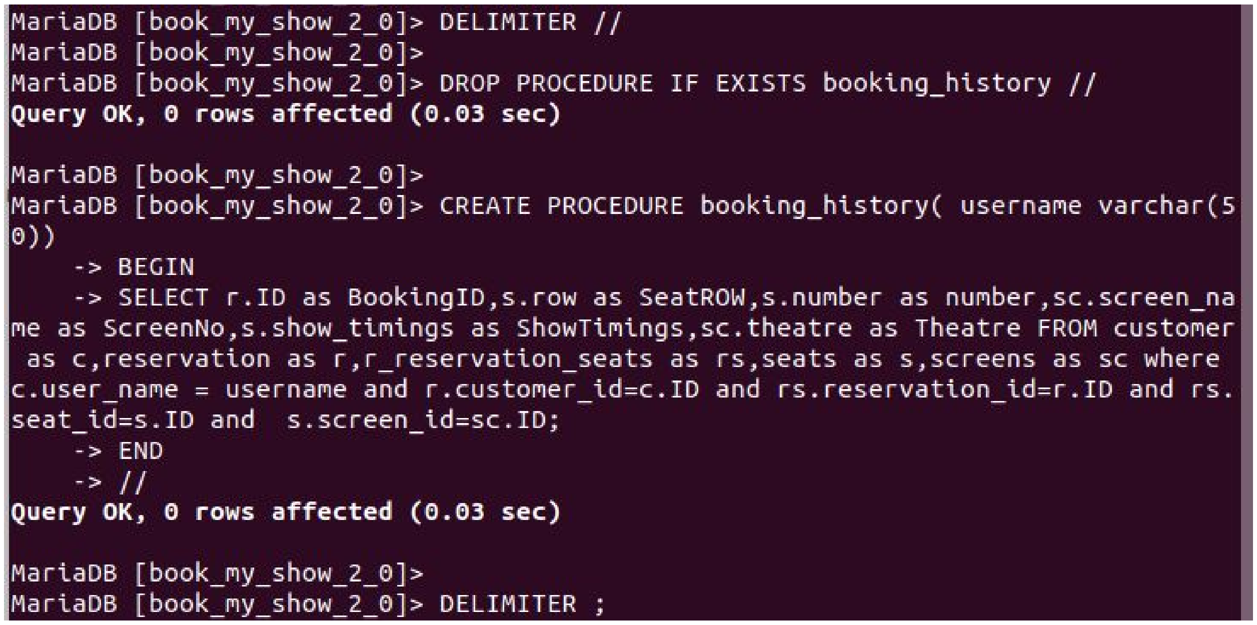
Creating view movie_filter:



```
MariaDB [bms_2_0]> create view movie_filter as (select DISTINCT
    ->              m.title,m.duration,m.description,m.language,m.format,m.release_date,g.name as genre,c.name as cast
    ->              from movie as m  ,r_genre_movie as rg,genre as g ,r_cast_movie as rc,casts as c
    ->              WHERE
    ->              (g.ID = rg.genre_id and rg.movie_id = m.ID)
    ->              and  (c.ID = rc.cast_id and rc.movie_id = m.ID));
Query OK, 0 rows affected (0.00 sec)
```

Creating Procedure booking_history:

The procedure takes customer user name as input and gives details of all the booking history

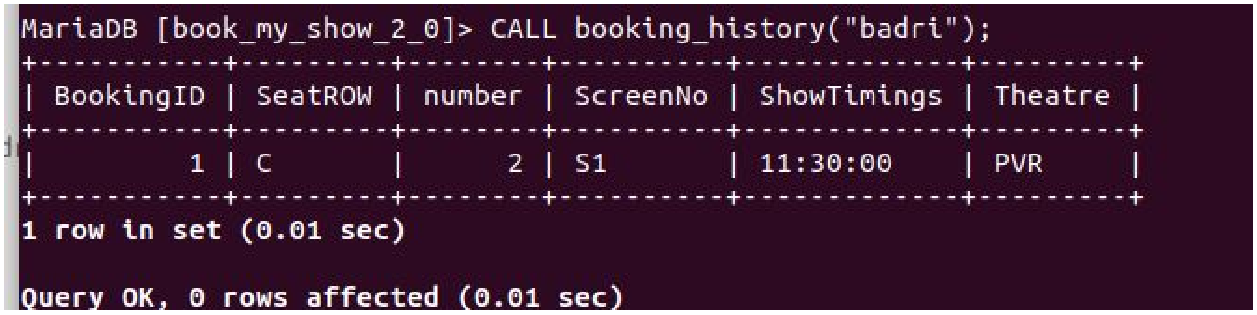


```
MariaDB [book_my_show_2_0]> DELIMITER //
MariaDB [book_my_show_2_0]>
MariaDB [book_my_show_2_0]> DROP PROCEDURE IF EXISTS booking_history //
Query OK, 0 rows affected (0.03 sec)

MariaDB [book_my_show_2_0]>
MariaDB [book_my_show_2_0]> CREATE PROCEDURE booking_history( username varchar(5
0))
    -> BEGIN
    -> SELECT r.ID as BookingID,s.row as SeatROW,s.number as number,sc.screen_na
me as ScreenNo,s.show_timings as ShowTimings,sc.theatre as Theatre FROM customer
 as c,reservation as r,r_reservation_seats as rs,seats as s,screens as sc where
c.user_name = username and r.customer_id=c.ID and rs.reservation_id=r.ID and rs.
seat_id=s.ID and  s.screen_id=sc.ID;
    -> END
    -> //
Query OK, 0 rows affected (0.03 sec)

MariaDB [book_my_show_2_0]>
MariaDB [book_my_show_2_0]> DELIMITER ;
```

Calling the procedure with username “badri”



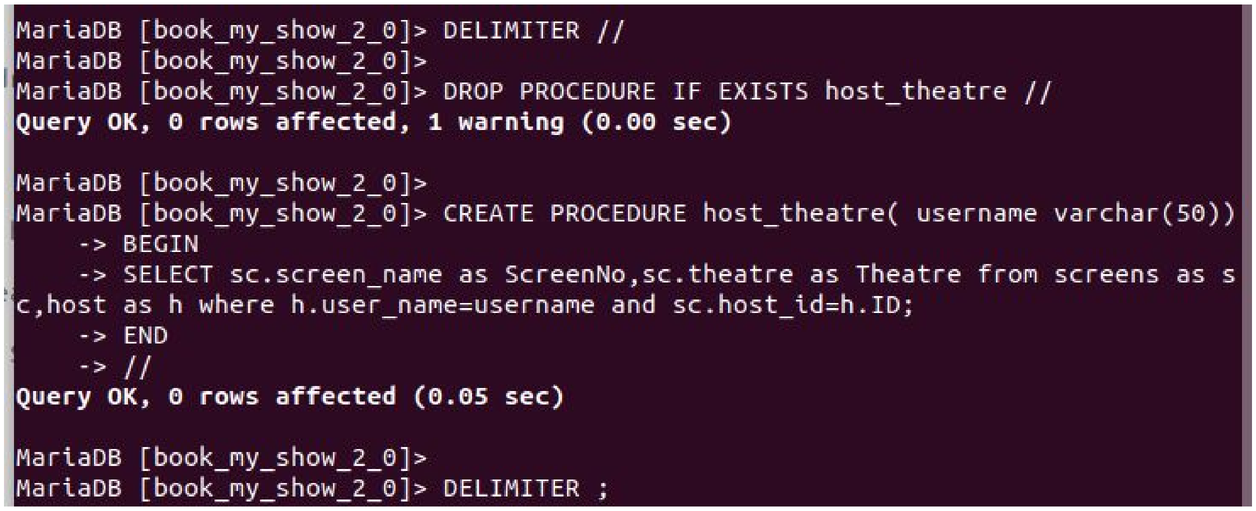
```
MariaDB [book_my_show_2_0]> CALL booking_history("badri");
+-----------+---------+--------+----------+-------------+---------+
| BookingID | SeatROW | number | ScreenNo | ShowTimings | Theatre |
+-----------+---------+--------+----------+-------------+---------+
|         1 | C       |      2 | S1       | 11:30:00    | PVR     |
+-----------+---------+--------+----------+-------------+---------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

Creating Procedure host_theatre:

It takes host username as input and gives all the theatres along with its screens owned by that particular host

Later it will be used to modify screens owned by that owner such as changing movie,increasing ticket price,....

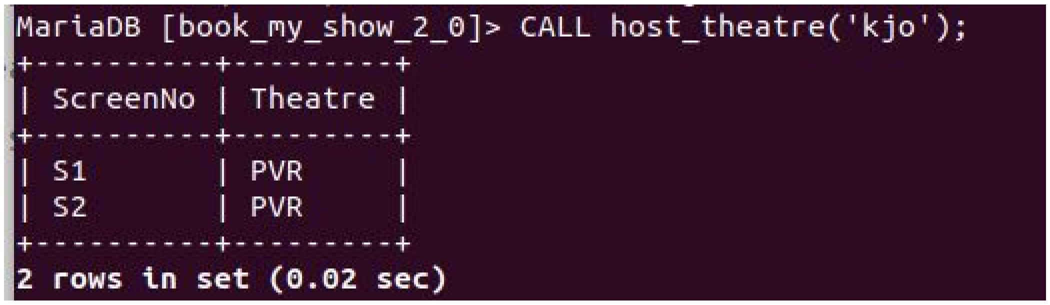


```
MariaDB [book_my_show_2_0]> DELIMITER //
MariaDB [book_my_show_2_0]>
MariaDB [book_my_show_2_0]> DROP PROCEDURE IF EXISTS host_theatre //
Query OK, 0 rows affected, 1 warning (0.00 sec)

MariaDB [book_my_show_2_0]>
MariaDB [book_my_show_2_0]> CREATE PROCEDURE host_theatre( username varchar(50))
    -> BEGIN
    -> SELECT sc.screen_name as ScreenNo,sc.theatre as Theatre from screens as s
c,host as h where h.user_name=username and sc.host_id=h.ID;
    -> END
    -> //
Query OK, 0 rows affected (0.05 sec)

MariaDB [book_my_show_2_0]>
MariaDB [book_my_show_2_0]> DELIMITER ;
```

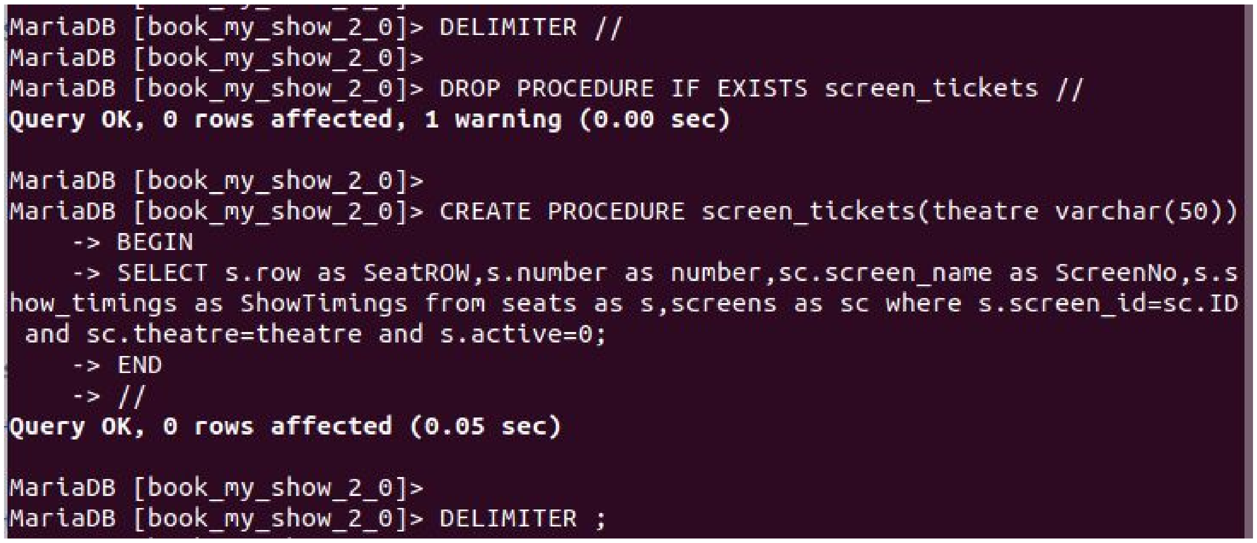
Calling the above procedure with username kjo.



```
MariaDB [book_my_show_2_0]> CALL host_theatre('kjo');
+----------+---------+
| ScreenNo | Theatre |
+----------+---------+
| S1       | PVR     |
| S2       | PVR     |
+----------+---------+
2 rows in set (0.02 sec)
```

Creating Procedure screen_tickets:

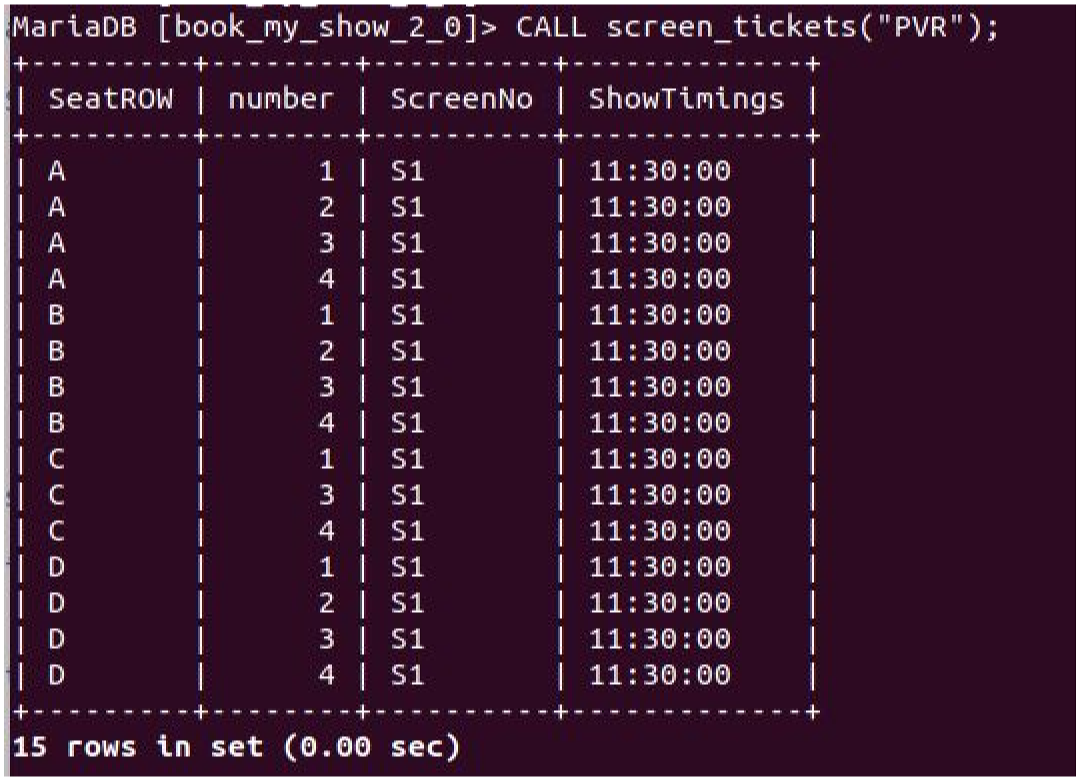
When a user wants to book tickets we need to show all the available tickets in a screen. So the procedure screen_tickets takes a theatre name as input and gives all the seats available for each screen.



```
MariaDB [book_my_show_2_0]> DELIMITER //
MariaDB [book_my_show_2_0]>
MariaDB [book_my_show_2_0]> DROP PROCEDURE IF EXISTS screen_tickets //
Query OK, 0 rows affected, 1 warning (0.00 sec)

MariaDB [book_my_show_2_0]>
MariaDB [book_my_show_2_0]> CREATE PROCEDURE screen_tickets(theatre varchar(50))
    -> BEGIN
    -> SELECT s.row as SeatROW,s.number as number,sc.screen_name as ScreenNo,s.s
how_timings as ShowTimings from seats as s,screens as sc where s.screen_id=sc.ID
 and sc.theatre=theatre and s.active=0;
    -> END
    -> //
Query OK, 0 rows affected (0.05 sec)

MariaDB [book_my_show_2_0]>
MariaDB [book_my_show_2_0]> DELIMITER ;
```



```
MariaDB [book_my_show_2_0]> CALL screen_tickets("PVR");
+---------+--------+----------+-------------+
| SeatROW | number | ScreenNo | ShowTimings |
+---------+--------+----------+-------------+
| A       |      1 | S1       | 11:30:00    |
| A       |      2 | S1       | 11:30:00    |
| A       |      3 | S1       | 11:30:00    |
| A       |      4 | S1       | 11:30:00    |
| B       |      1 | S1       | 11:30:00    |
| B       |      2 | S1       | 11:30:00    |
| B       |      3 | S1       | 11:30:00    |
| B       |      4 | S1       | 11:30:00    |
| C       |      1 | S1       | 11:30:00    |
| C       |      3 | S1       | 11:30:00    |
| C       |      4 | S1       | 11:30:00    |
| D       |      1 | S1       | 11:30:00    |
| D       |      2 | S1       | 11:30:00    |
| D       |      3 | S1       | 11:30:00    |
| D       |      4 | S1       | 11:30:00    |
+---------+--------+----------+-------------+
15 rows in set (0.00 sec)
```