AY 13/14 | SEM1

CS2102 Project Report 2

Movie Ticket-Booking System & Database

GROUP 42: Lim Binjie Benjamin, Yang Xiaoyu, Yap Zi Xuan

Project Overview

Our team is developing a movie ticket-booking system, Popcorn, for a local cineplex company. The company owns multiple cinemas across the country, and each cinema has multiple halls where movies are screened.

This booking system will comprise of 2 interfaces - one for customers, and the other for the cineplex's site admin:

1. Customer side: the former interface allows customers to create user accounts to make single bookings, make online payments and cancel bookings prior to payment

2. Admin side: employees of the cineplex are able to login to access an admin panel that would allow them to insert, modify and remove movies, show timings. Other operations that the admin could perform include the ability to view sales data and retrieve booking information of customers.

Design Specifications

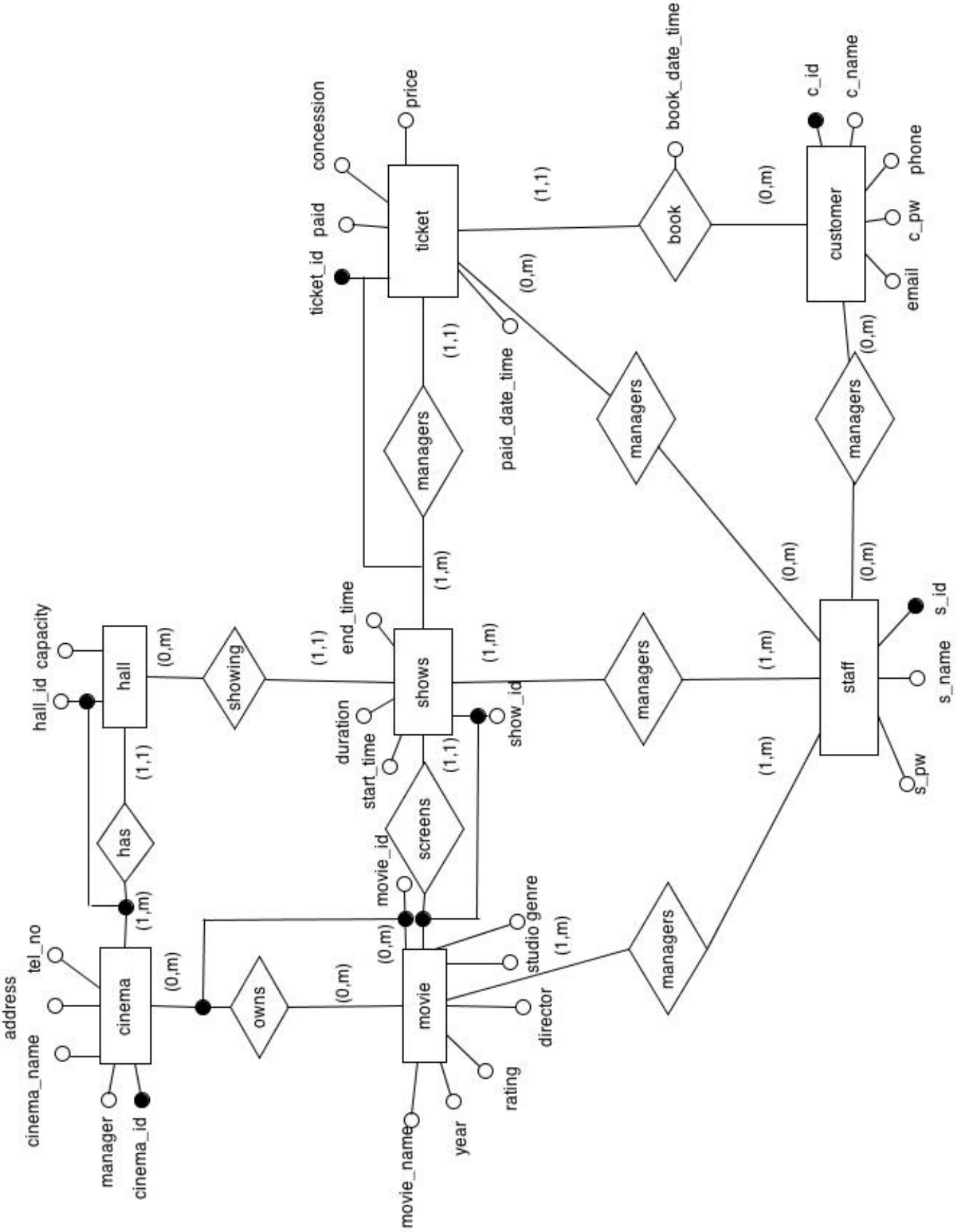
We are developing Popcorn from the following tools:

Web Server:	**Apache 2.4**
Web Server Languages:	HTML5, CSS3, PHP, JavaScript
DBMS:	MySQL 5.6.12 + phpmyadmin extension

Core Features

Customer	Admin
Book and cancel ticket.	Display all bookings made by customers
Choose to pay online or pay at the cinema counter	Look at aggregate sales, and sales categorised by movies
View past bookings	Insert, delete and update movies, show timings and hall locations
Search for movies according to movie title, date, and Cinema.	Update 'paid' status of tickets to reflect paid tickets during counter payments
Rank movies according to showing date, showing time, and ratings.	

Revised ER Diagram



Entity Table

Here is a list of our entities we used in our relation schema:

Movie	
Attribute	**Domain**
movieID (key)	INT
movieName	VARCHAR(256)
year	DATE
genre	CHAR(64)
studio	VARCHAR(64)
director	VARCHAR(64)
rating	DOUBLE

Shows	
Attribute	**Domain**
showID (key)	INT
movieID	INT
hallID	INT
duration	INT
startTime	DATETIME
endTime	DATETIME

Ticket	
Attribute	**Domain**
ticketID (key)	INT
cID	INT
showID	INT
seatNo	CHAR(3)
price	DOUBLE
concession	CHAR(5)
paid	BOOLEAN
bookDateTime	DATETIME
paidDateTime	DATETIME

Customer	
Attribute	**Domain**
cID (key)	INT
cName	VARCHAR(64)
email	VARCHAR(128)
cPw	VARCHAR(64)
phone	INT

Hall	
Attribute	**Domain**
hallID (key)	INT
cinemaID	INT
capacity	INT

Staff	
Attribute	**Domain**
sID (key)	INT
sName	VARCHAR(64)
sPw	VARCHAR(64)

Cinema	
Attribute	**Domain**
cinemaID (key)	INT
cinemaName	VARCHAR(64)
address	VARCHAR(256)
telNo	INT

Revised Relation Schema

Here is our database schema in SQL DDL code:

```
DROP TABLE IF EXISTS ticket;
DROP TABLE IF EXISTS staff;
DROP TABLE IF EXISTS customer;
DROP TABLE IF EXISTS shows;
DROP TABLE IF EXISTS hall;
DROP TABLE IF EXISTS movie;
DROP TABLE IF EXISTS cinema;


CREATE TABLE cinema(
     cinemaID INT NOT NULL AUTO_INCREMENT CHECK (cinemaID > 0
AND cinemaID < 1000) ,
     cinemaName VARCHAR(64) NOT NULL UNIQUE,
     address VARCHAR(256) NOT NULL,
     telNo INT NOT NULL,
     PRIMARY KEY(cinemaID)
) AUTO_INCREMENT=1;


CREATE TABLE movie(
     movieID INT NOT NULL AUTO_INCREMENT CHECK (movieID > 2000
AND movieID < 3000) ,
     movieName VARCHAR(256) NOT NULL,
     year DATE NOT NULL,
     genre CHAR(32),
     studio VARCHAR(256) NOT NULL,
     director VARCHAR(256),
     rating DOUBLE,
     PRIMARY KEY(movieID)
)AUTO_INCREMENT=2001;


CREATE TABLE hall(
     hallID INT NOT NULL AUTO_INCREMENT CHECK (hallID > 1000
AND hallID < 2000) ,
     cinemaID INT,
     capacity INT CHECK(capacity > 0),
     FOREIGN KEY(cinemaID) REFERENCES cinema(cinemaID) ON
UPDATE CASCADE ON DELETE CASCADE,
     PRIMARY KEY(hallID)
)AUTO_INCREMENT=1001;


CREATE TABLE shows(
     showID INT NOT NULL AUTO_INCREMENT CHECK (showID > 3000
AND showID < 4000) ,
```

```
     movieID INT,
     hallID INT,
     duration INT,
     startTime DATETIME NOT NULL,
     endTime DATETIME NOT NULL,
     FOREIGN KEY(hallID) REFERENCES hall(hallID) ON UPDATE
CASCADE ON DELETE CASCADE,
     FOREIGN KEY(movieID) REFERENCES movie(movieID) ON UPDATE
CASCADE ON DELETE CASCADE,
     PRIMARY KEY(showID)
)AUTO_INCREMENT=3001;

CREATE TABLE customer(
     cID INT NOT NULL AUTO_INCREMENT CHECK (cID > 4000 AND cID
< 5000) ,
     cName VARCHAR(256) NOT NULL,
     email VARCHAR(128) NOT NULL UNIQUE,
     cPw VARCHAR(64) NOT NULL,
     phone INT NOT NULL,
     PRIMARY KEY(cID)
)AUTO_INCREMENT=4001;

CREATE TABLE staff(
     sID INT NOT NULL AUTO_INCREMENT CHECK (sID > 5000 AND sID
< 6000) ,
     sName VARCHAR(256) NOT NULL,
     sPw VARCHAR(64) NOT NULL,
     PRIMARY KEY(sID)
)AUTO_INCREMENT=5001;

CREATE TABLE ticket(
     ticketID INT NOT NULL AUTO_INCREMENT CHECK (ticketID >
6000 AND ticketID < 7000) ,
     cID INT,
     showID INT,
     seatNo CHAR(3) NOT NULL,
     price DOUBLE NOT NULL,
     concession CHAR(5) NOT NULL DEFAULT 'ADULT', CHECK
(concession = 'CHILD' OR concession = 'ELDER' OR concession =
'ADULT') ,
     paid BOOLEAN NOT NULL DEFAULT 0,
     bookDateTime DATETIME NOT NULL,
     paidDateTime DATETIME,
     FOREIGN KEY(cID) REFERENCES customer(cID) ON UPDATE
CASCADE ON DELETE CASCADE,
     FOREIGN KEY(showID) REFERENCES shows(showID) ON UPDATE
CASCADE ON DELETE CASCADE,
     PRIMARY KEY(ticketID)
)AUTO_INCREMENT=6001;
```

Schema Notes*

1) In the Cinema, Hall, Movie and Shows relations described in the schema above, a boundary check is performed on their respective primary keys, such as this:

```
cinemaID INT(6) NOT NULL AUTO_INCREMENT CHECK (cinemaID > 0
AND cinemaID < 1000),
```

Here, in the cinema relation, we are assuming that there will never be more than 999 cinemas existing in the database. The attribute has to be an INT type (instead of CHAR), so as to accommodate auto-incrementation. Each cinemaID has an auto-incremented value starting from 1, with the maximum value being 999. The key assumptions and rationale for the check applies similarly to the movie, show and hall relations.

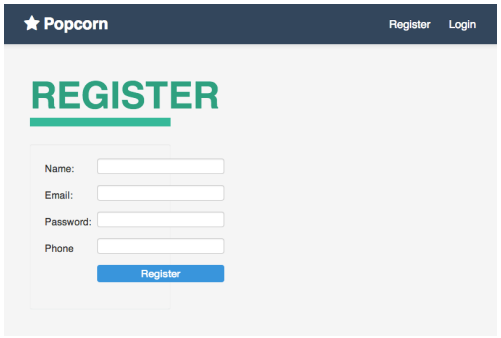
2) In the customer relation, cID refers to customer ID, which also serves as the customer's username. cPW refers to their password. Both cID and cPW are required to login to their user account in Popcorn.

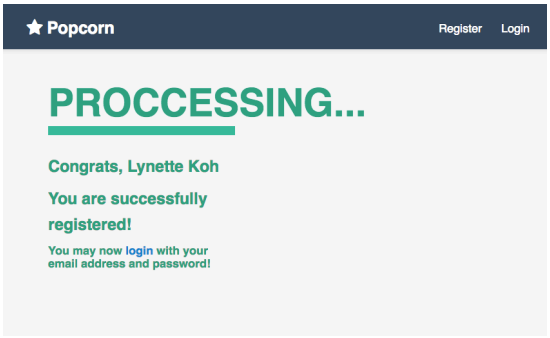
Similarly, for the staff relation, sID and sPW refer to the cineplex employees' username and password that is needed to access the admin panel.

SQL Code

1. Customer-side

1.1 Registration






```
$sql = "INSERT INTO `theatre`.`customer` (`cID`, `cName`, `email`, `cPw`, `phone`) VALUES (null, '$customerName', '$email', '$password', '$phone');";
mysql_select_db('movie');
$retval = mysql_query( $sql, $conn );
if(! $retval ){
     die('Could not enter data: ' . mysql_error());
}
echo "<h3>Congrats, $customerName</h3>";
echo "<h3>You are successfully registered!</h3>";
echo "<h4>You may now <a href='./login.php'> login </a> with your email address and password!</h4>";
mysql_close($conn);
}
```

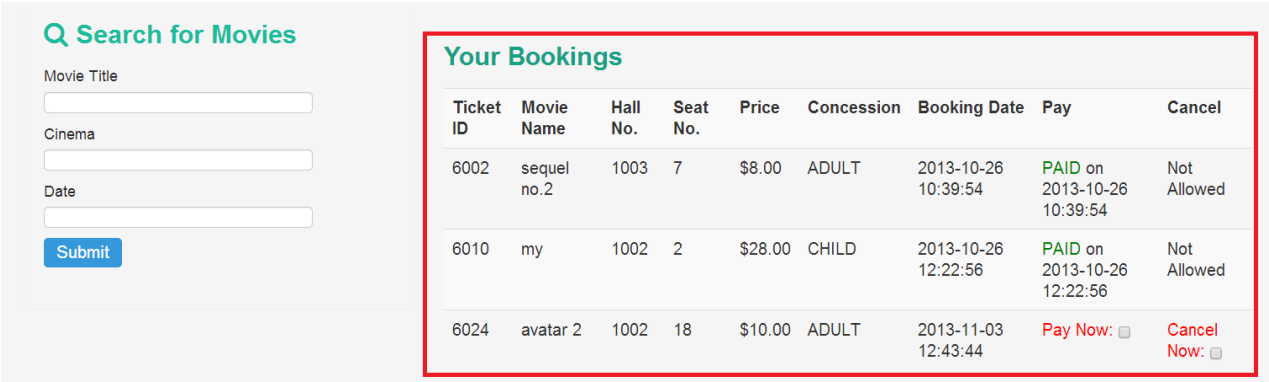
For registration, customer fills in a HTML form and submits a POST request. The customer details are inserted into the customer table. If an error occurs $retval is false and the page displays an error. If the customer has been added successfully, a success message is displayed.

1.2 Login


```
try {
     $sth = $db->prepare("SELECT cID FROM customer WHERE
email= :email AND cPw= :pw ");
               $sth->execute();
               while($row = $sth-
>fetch(PDO::FETCH_ASSOC)) {
                $cid = $row['cID'];
               }
       } catch(Exception $ex) {
        echo $ex;
       }
       if(is_null($cid)){
               echo "Invalid user or password";
       }
       else{
               header("Location: dash.php?id=".$cid);
       }
```

The login functions takes in 2 variables from the post form. A query return the row which matches both variables and the customer ID is stored into a variable as we require the ID to display the customer's past bookings. If the ID is null, the row does not exist and the invalid user or password message is displayed.

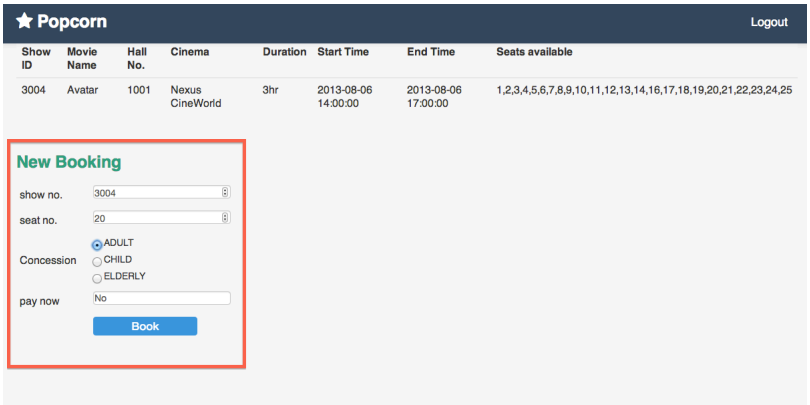
1.3 View Existing Bookings

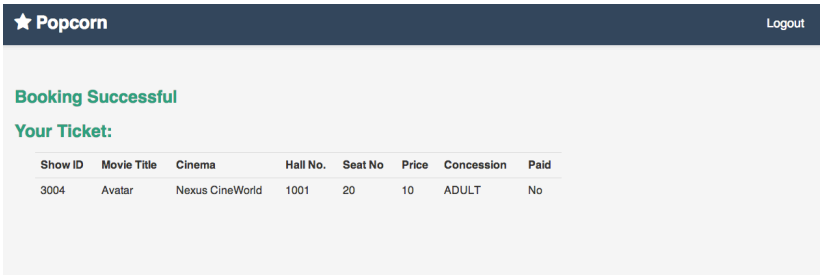



```
    $sth = $db->prepare("SELECT * FROM ticket WHERE cID=
:id");
    $sth->bindValue(':id', $cid);
    $sth->execute();
    while($row = $sth->fetch(PDO::FETCH_BOTH))  {
    echo "<tr>";
    $tid = $row[0];
    $paid = $row[6];
    echo "<td> $tid </td>";
    echo "<td> $row[2] </td>";
    ...
    echo "<td> $row[8] </td>";
    if ($paid == 1){
        echo '<td>Paid on '.$row[7].' </td>';
        echo '<td>Not Allowed</td>';
    }
    else{
        echo '<td> <input type="checkbox" value='.$tid.'
id="pay" class="cb"> </td>';
        echo '<td> <input type="checkbox" value='.$tid.'
id="cancel" class="cb"> </td>';
    }
```

The customer ID from the login form is used to retrieve past bookings made by the customer. If the customer has paid, the table cell for payment and cancelling is filled with text because the customer is not allowed to cancel payment or cancel his booking. If he has not paid, the cells contain checkboxes allowing him to pay or to cancel the booking.

1.4 Making a Booking

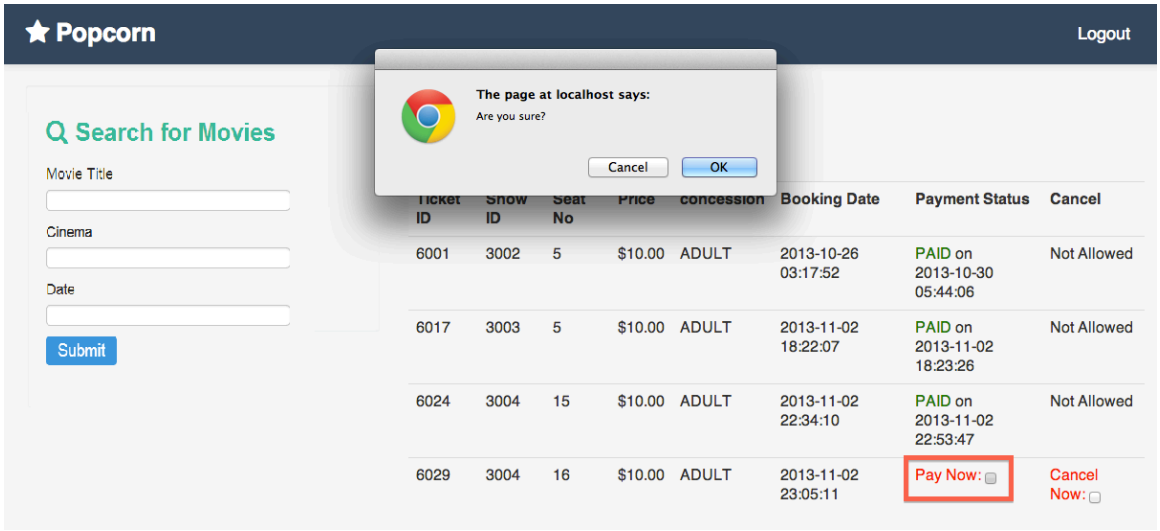


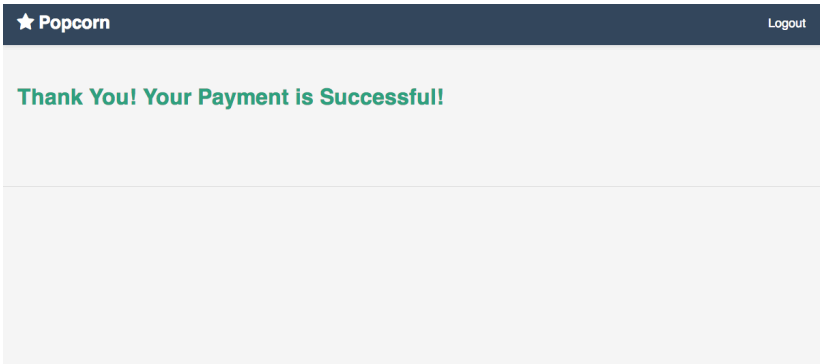



```
 if ($paid==1){
     $paiddate = date('Y-m-d H:i:s');
 }
 else{
     $paiddate = null;
 }
$sql = "INSERT INTO `theatre`.`ticket` (`ticketID`,
`cID`, `showID`, `seatNo`, `price`, `concession`, `paid`,
`bookDateTime`, `paidDateTime`)
VALUES (null, '$customerID', '$showID','$seatNo',
'$price', '$concession', '$paid', '$date', '$paiddate');";

mysql_select_db('movie');
$retval = mysql_query( $sql, $conn );
if(! $retval ){
 die('Could not enter data: ' . mysql_error());
}
echo "Entered data successfully\n";
mysql_close($conn);
}
header("Location: dash.php?id=".$customerID);
```

If a customer chooses to pay at the time of booking, the paidDateTime is set to current date, it is left as null otherwise. bookDateTime is always set as the time of the booking. retval will be false if an error occurs while inserting. An error message will then be displayed. Otherwise, the user is redirected back to the dashboard.

1.5 Payment

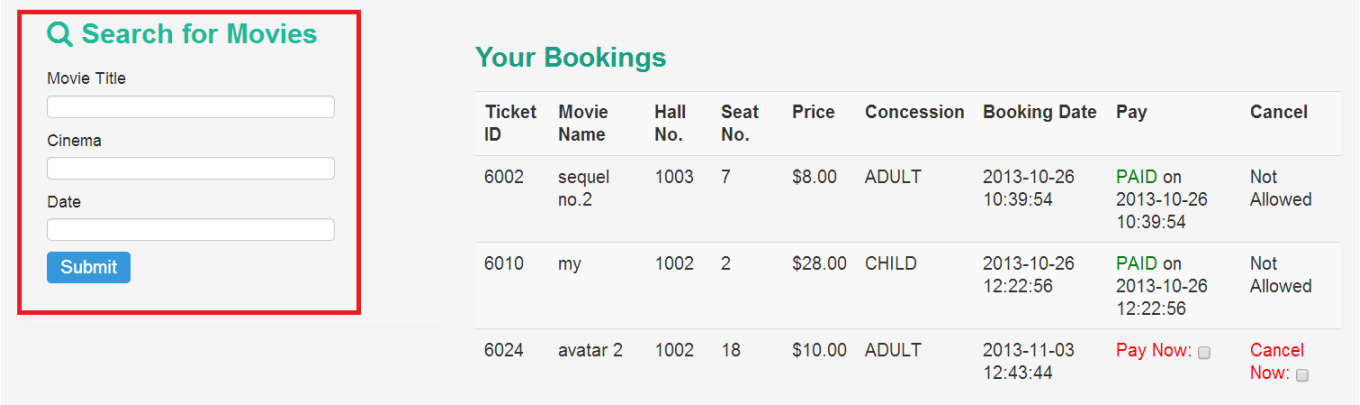


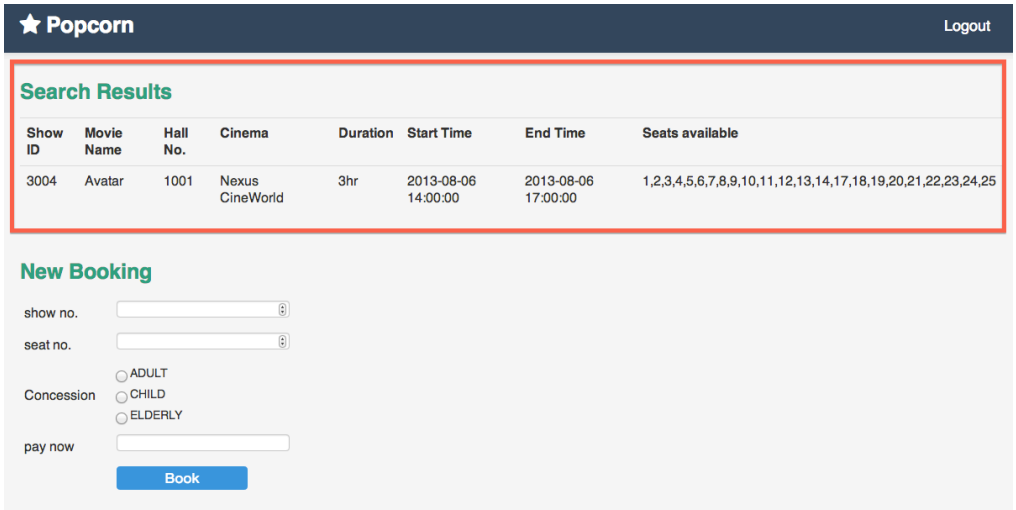



```
$paiddate = date('Y-m-d H:i:s');
echo $paiddate;
mysqli_query($con,"UPDATE ticket SET
paidDateTime='$paiddate' WHERE ticketID='$ticketID'");
mysqli_query($con,"UPDATE ticket SET paid='1' WHERE
ticketID='$ticketID'");
```

For payment, the ticket ID will be received through GET parameters. The first query sets the paidDateTime to the current time. The second sets the boolean variable paid to true. The customer is then redirected back to the dashboard.

1.6 Search movies





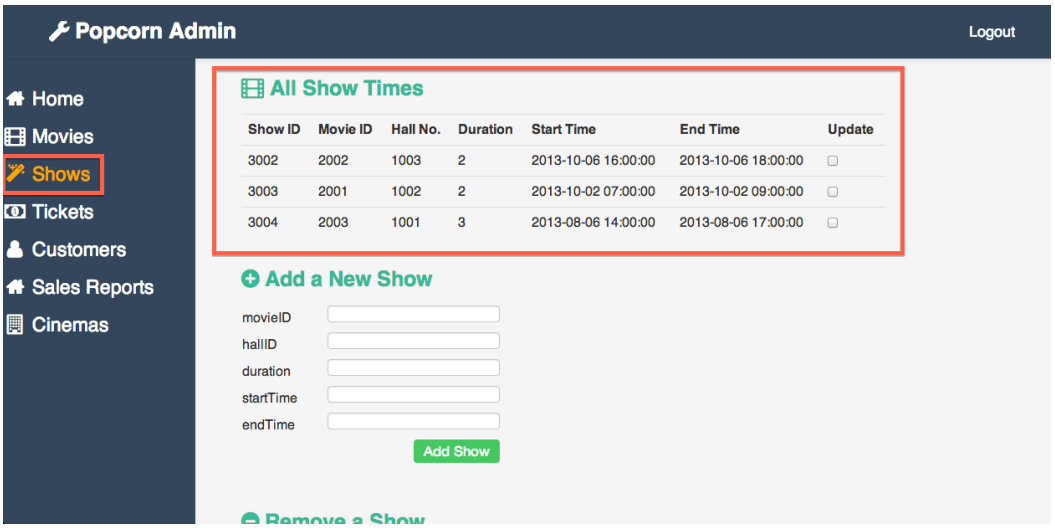

```
$sth = $db->prepare("SELECT * FROM shows, movie,
hall, cinema WHERE movieName LIKE '%$title%'
AND shows.movieID=movie.movieID
AND shows.startTime BETWEEN :date0 AND :date1
AND shows.hallID=hall.hallID
AND hall.cinemaID = cinema.cinemaID
AND cinemaName LIKE '%$cinema%'");


$sth->execute();
while($row = $sth->fetch(PDO::FETCH_BOTH))  {
…
$sth1 = $db->prepare("SELECT * FROM movie WHERE
movieID= :id1");
$sth1->execute();
…
$sth2 = $db->prepare("SELECT cinemaName FROM cinema,
hall WHERE hallID= :id2 AND hall.cinemaID =
cinema.cinemaID");
$sth2->execute();
…
```

```
$sth3 = $db->prepare("SELECT * FROM hall WHERE
hallID= :id3");
…
for($i=1;$i<=$total;$i++){
     array_push($arr, $i);
}
$sth4 = $db->prepare("SELECT seatNo FROM ticket
WHERE showID= :id2");
…
$taken = array();
while ($row2 = $sth2->fetch(PDO::FETCH_BOTH)){
      array_push($taken, $row2[0]);
}
$diff = array_diff($arr,$taken);
}
```

Statement handler sth searches for and displays shows that are screened at a certain cinema, on a certain date or are of a specific movie title. sth1 returns the movie name which will be displayed in place of the movie ID. sth2 returns the cinema name which will be displayed in place of the ID. sth3 queries for the capacity of a hall and sth4 queries for the seat number of all tickets of a particular show. The empty seats are calculated by creating an array with all the seats in the hall and taking the difference with an array of all occupied seats. The result of the all the queries are organised into a table and displayed.

2. Admin-side

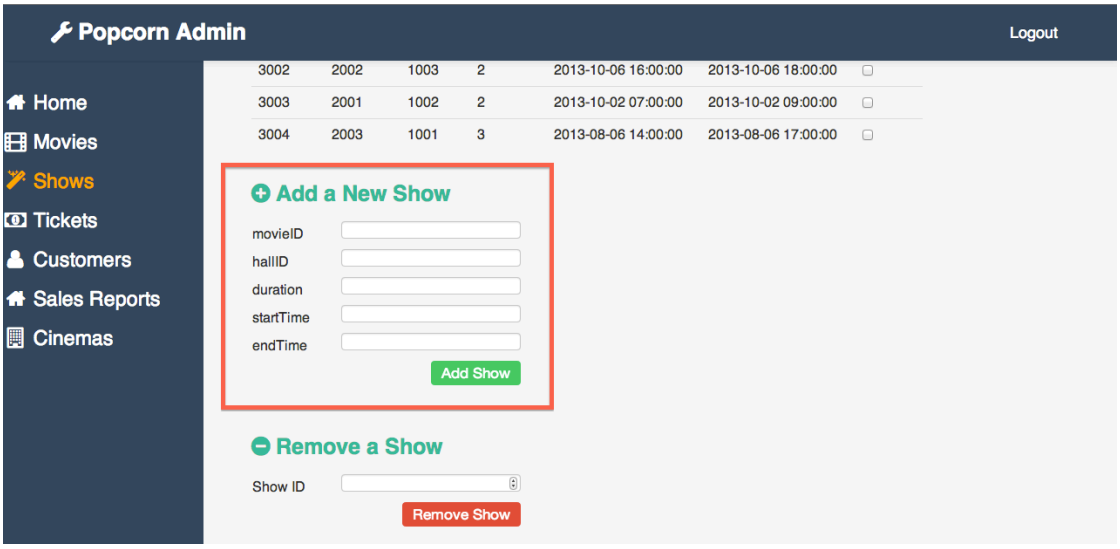
2.1 Viewing shows, movies, cinema, customer, ticket and hall information




```
$query_shows = "SELECT * FROM `shows`";
$rs = mysql_query($query_shows);
if (!$rs) {
   echo "Could not execute query: $query";
} else {
}
echo
"<table><td>showID</td><td>movieID</td><td>hallID</td><td
>duration</td><td>startTime</td><td>endTime</td>";
while ($row = mysql_fetch_row($rs)) {
   {
   echo "<tr>";
   echo "<td> $row[0] </td>";
   ...
   echo "<td> $row[5] </td>";
   echo "</tr>";
   }
}
```

For displaying show information, the select statement is used. The results are retrieved by rows and the information is displayed in table form. The code for retrieving movies, cinema, customer, ticket and hall is very similar.

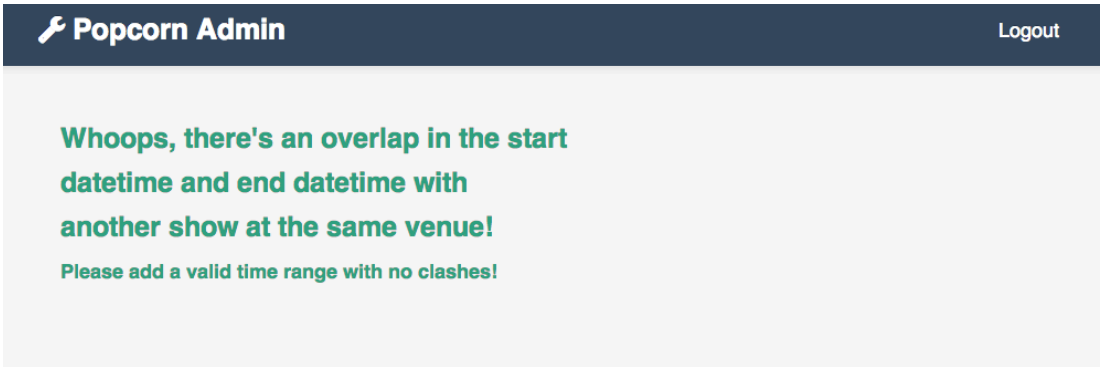
2.3 Adding shows, movies, cinema and hall information




```
$movieID = ($_POST['movieID']);
...
$endTime = $_POST['endTime'];


$sql = "INSERT INTO `theatre`.`shows` (`showID`,
`movieID`, `hallID`, `duration`, `startTime`, `endTime`)
VALUES (null, '$movieID', '$hallID', '$duration',
'$startTime', `endTime`);";


$retval = mysql_query( $sql, $conn );
if(! $retval )
{
 die('Could not enter data: ' . mysql_error());
}
echo "Entered data successfully\n";
mysql_close($conn);
}
```

In order to add new shows, the admin submits a form containing the relevant details which are stored in variables and inserted into the shows tables. retval will contain false if an error occurs. Otherwise, the admin will be redirected back to the admin panel.

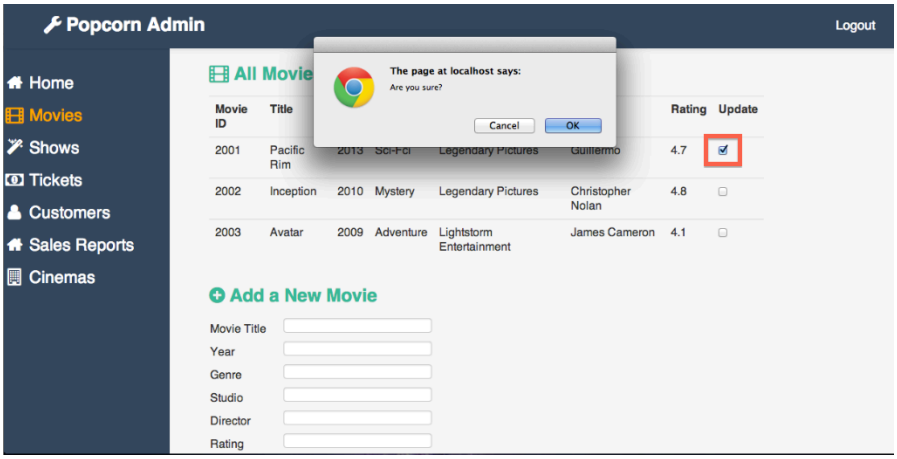
2.4 Input validation when adding new show

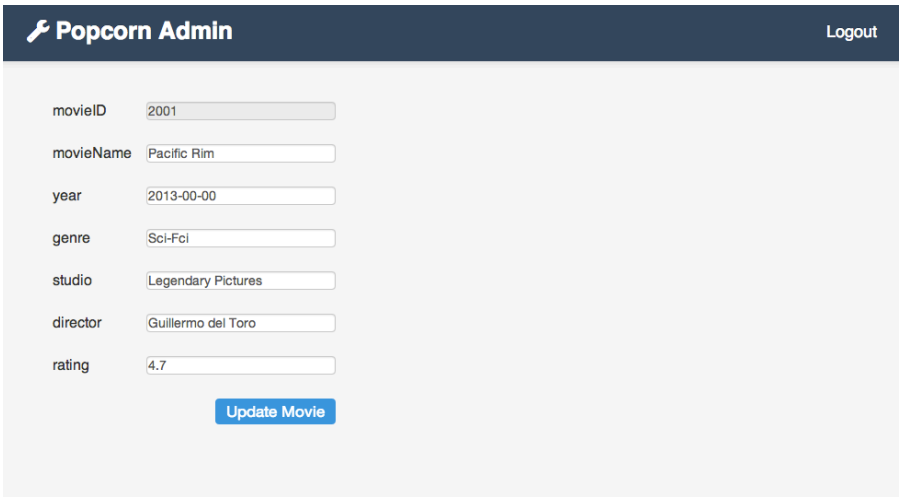



```
if ($startTime > $endTime)
                die('Start time is after end time.
Error');
if ((strtotime($endTime) - strtotime($startTime)/(60*60))
!= $duration)
                die('Duration is not Correct.');
$sth = $db->prepare("SELECT startTime,endTime FROM shows
WHERE hallID=:id AND showID<>:sid");
$sth->bindValue(':id', $hallID);
$sth->bindValue(':sid', $showID);
$sth->execute();
while($row = $sth->fetch(PDO::FETCH_BOTH)) {
        $retval =
datesOverlap($row[0],$row[1],$startTime,$endTime);
         if ($retval == 1){
                die('Overlap in dates');
         }
}
function
datesOverlap($start_one,$end_one,$start_two,$end_two) {
  if($start_one <= $end_two && $end_one >= $start_two) {
//If the dates overlap
        return 1;
  }
  return 0; //Return 0 if there is no overlap
}
```

The first if condition checks if start time is before end time, returning an error if it is not. The second condition checks that the duration is correct. The statement handler sth queries for the start and end time of shows that are screened at the same hall. The start time and end time are then compared to make sure that there is no overlap.

2.5 Updating shows, movies, cinema and hall information

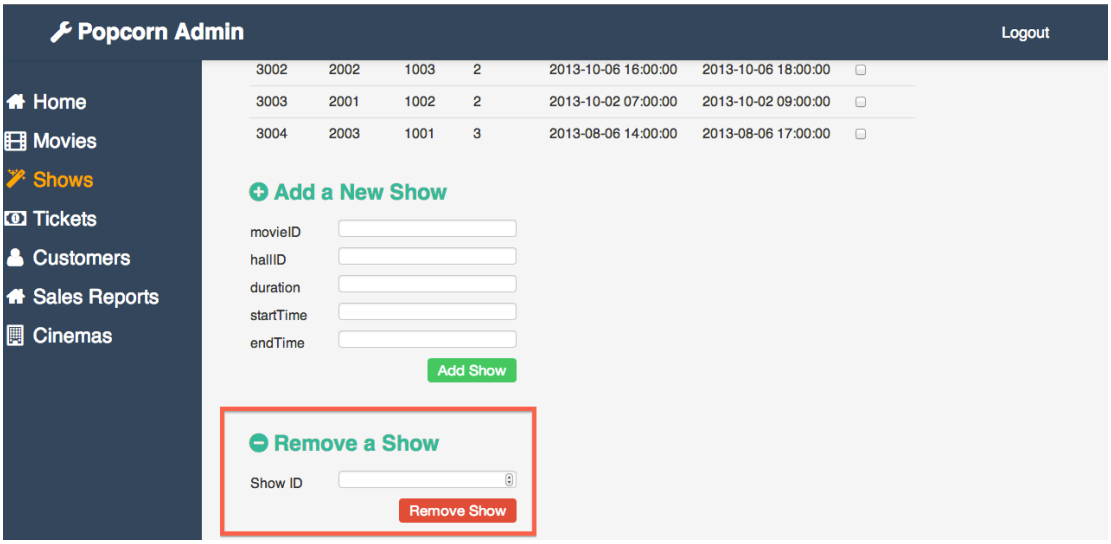





```
$movieID = ($_POST['movieID']);
...
$endTime = $_POST['endTime'];
$sql = "UPDATE `theatre`.shows SET movieID='$movieID',
hallID='$hallID', duration='$duration',
startTime='$startTime',  endTime='$endTime' WHERE
showID='$showID'";

$retval = mysql_query( $sql, $conn );
if(! $retval )
{
 die('Could not enter data: ' . mysql_error());
}
echo "Entered data successfully\n";
mysql_close($conn);
}
```

In order to update shows, the admin submits a form containing the relevant details that are stored in variables. The update statement is used to update only the row that matches the correct ID.

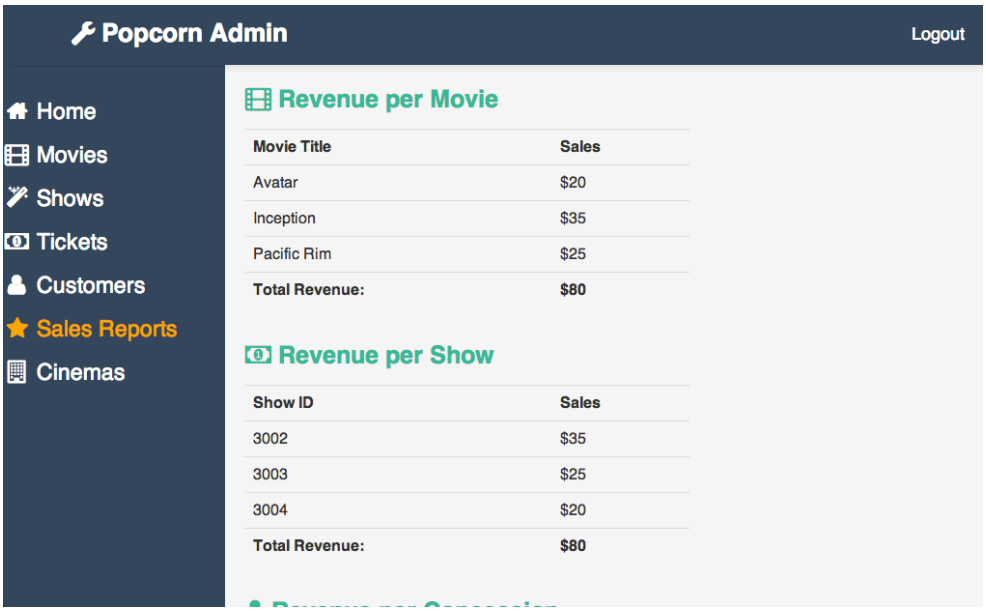
2.6 Deleting entities




```
try {
        $id = ($_POST['id']);
        if ($id < 1000){
                mysqli_query($con,"DELETE FROM cinema
WHERE cinemaID='$id'");
                mysqli_close($con);
        }
        else if ($id < 2000){
                mysqli_query($con,"DELETE FROM hall WHERE
hallID='$id'");
                mysqli_close($con);
        }
        else if ($id < 3000){
                mysqli_query($con,"DELETE FROM movie WHERE
movieID='$id'");
                mysqli_close($con);
        }
    ...
```

A single function can be used to delete all entities because the IDs for all entities are unique and fall within a certain range. An if else clause is used to check which entity type the id belongs to and the relevant delete query is executed.

2.7 Viewing sales reports




```
$query = "SELECT movie.movieName, SUM(price) FROM
                ticket, shows, movie WHERE
                ticket.showID=shows.showID AND
                shows.movieID=movie.movieID GROUP BY
                movie.movieName";
   $rs = mysql_query($query);
   if (!$rs) {
       echo "Could not execute query: $query";
       trigger_error(mysql_error(), E_USER_ERROR);
   } else {
   }
   echo '<div class="span6">';
   echo "<h3 class='folio-title'><span class='main-
                color'><i class='fa-icon-film'></i>
                Revenue per Movie</span></h3>";
   echo "<table class='table'><tr>
  <th>Movie Title</th>
  <th>Sales</th>
  </tr>";
   while ($row = mysql_fetch_row($rs)) {
       {
       echo "<tr>";
       echo "<td> $row[0] </td>";
       echo "<td> $$row[1] </td>";
       echo "</tr>";
       }
   }


   $query = "SELECT SUM(price) FROM `ticket`";
   $rs = mysql_query($query);
```

```
if (!$rs) {
    echo "Could not execute query: $query";
    trigger_error(mysql_error(), E_USER_ERROR);
} else {
}
echo '<tr>';
$row = mysql_fetch_row($rs);
echo "<td><span style='font-weight:bold'>Total
            Revenue:</span></td>
     <td><span style='font-
            weight:bold'>$$row[0]</td></span></tr
            >";
echo "</table>";
echo "</div>";
```

Sales are displayed per movie, show and concession types. To displays the sales for each movie, we query the ticket, show and movie table, grouping the results by movie name. The results are then displayed in table form. The total revenue is obtained by summing the prices of all entries in the ticket table. Lastly, to obtain the sales per show, we group the results by showID and display the results in table form.