

STATISTICS SECTION

The statistics section of our project was split into 3 main jsp pages: selectStatistic.jsp, newStatistic.jsp, and statisticErr.jsp.

selectStatistic.jsp

selectStatistic.jsp was the “main statistics home page”, where the user has the option to select a start date and end date they want to view results for, as well as the statistic they want to view. There are 4 options for statistics to view:

1. Given a specific time period, compute the highest rated room type for each hotel
2. Given a specific time period, compute customers with the 5 highest total money spent
3. Given a specific time period, compute highest rated breakfast type across all hotels
4. For a specific time period, compute the highest rated service type across all hotels

The input values are fairly simple, there are 3 main things to input: the start date, end date, and type of statistic.


```
<select name="startmonth">
  <option value="1">January</option>
  <option value="2">February</option>
  <option value="3">March</option>
  <option value="4">April</option>
  <option value="5">May</option>
  <option value="6">June</option>
  <option value="7">July</option>
  <option value="8">August</option>
  <option value="9">September</option>
  <option value="10">October</option>
  <option value="11">November</option>
  <option value="12">December</option>
</select>
</span>
```

As you can see from the screenshot above, all the options are accessible via dropdown menus. To the left, there's also a screenshot of the implementation in the .jsp file. Every option is assigned a numerical value (which makes it easier to process the errors).

After the user presses the submit button, the page is redirected to newStatistic.jsp, which generates the SQL queries as well as checks for any errors in the input.

newStatistic.jsp

This is the jsp page where the program connects to our database and does the query.

First, the program takes all the input from selectStatistic.jsp and converts it into usable Java variables. Because we want to compare dates but our HTML inputs were all in string form, I first converted everything into integers using Integer.parseInt().

After that, the program checks for errors: for the inputs given, there are only 3 possible types of errors:

Error 1: The start date was not valid

Error 2: The end date was not valid

Error 3: The end date came before the start date, which is also not valid

Because the form accepts all days 1-31 for every month, if the user chooses, for example, February 29th or November 31st, these dates should not be accepted. The code to check this is implemented below.

```
if (startmonth==4 || startmonth==6 || startmonth==9 || startmonth==11){
    //only 30 days in the month
    if (startday >30){
        statisticCheck[0] = "-1";
        error = true;
    }
}
else if (startmonth==2){
    //only 27 days
    if (startday >28){
        statisticCheck[0] = "-1";
        error = true;
    }
}
```

There are 2 checks: one for February, which usually only has 28 days (this calendar assumes no leap years), and one for the months where there are only 30 days.

This check works for both error 1 and error 2.

Checking for error 3 (the end date occurs before the start date) is fairly simple since all the date values are converted into integers. I used nested if loops, first to check if the years were accepted, then to check if the months were accepted, then to compare the days. If the end date ended up occurring before the start date, statisticCheck[2] (error 3) is set to -1 and the error boolean is set to true.

```
if (endyear < startyear){
    //error, year before startyear
    statisticCheck[2] = "-1";
    error = true;
}
else if (endyear == startyear){
    //check month
    if (endmonth < startmonth){
        //error, endmonth before startmonth
        statisticCheck[2] = "-1";
        error = true;
    }
    else if (endmonth == startmonth){
        //check day
        if (endday < startday){
            //error, endday before startday
            statisticCheck[2] = "-1";
            error = true;
        }
    }
}
```

The code to check if the end date occurs before the start date is shown in the screenshot of code to the left.

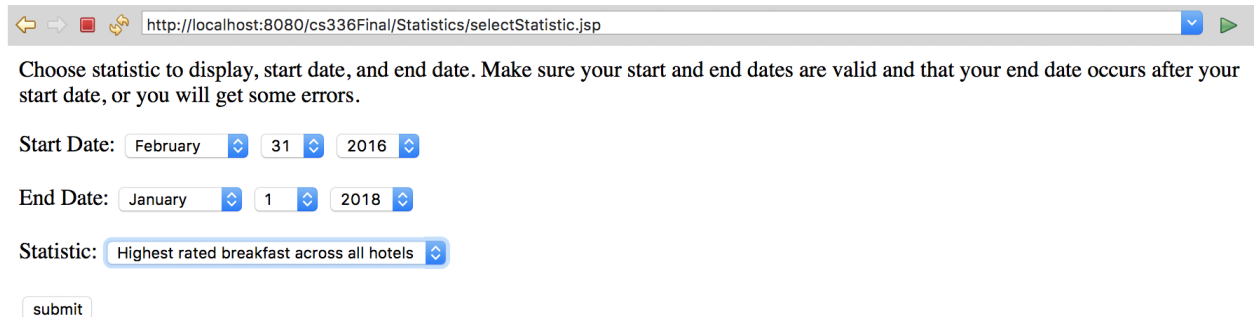
After all these checks, the program will either redirect to statisticErr.jsp if any kind of error occurs, or continue on to the query portion of newStatistic.jsp.

Since statisticErr.jsp is fairly simple, we'll go over that implementation first before returning to newStatistic.jsp to see how the SQL queries were implemented.

statisticErr.jsp

This jsp page only occurs when there's an error in the input of selectStatistic.jsp. It displays the error explanation, then offers a redirect back to the main page.

For example, using the following input on selectStatistic.jsp:



Will return the following error page:



If multiple errors happen at once, the page returns explanations for every error that occurs. For example, if we enter an invalid start date and end date which hypothetically occurs in the wrong order, like below:



We will get the following error page:



The implementation for this is fairly simple, since all of the actual error checking is done in newStatistic.jsp.

```
Statistic error(s):
<br/>
<br/>

String[] errors = (String[])session.getAttribute("errors");

if(errors[0].equals("-1")){
    out.write("Start date: not a valid date <br/> <br/>");
}
if(errors[1].equals("-1")){
    out.write("End date: not a valid end date <br/> <br/>");
}
if(errors[2].equals("-1")){
    out.write("The end date occurs before the start date <br/> <br/> ");
}

%>

<br/>
Try again:
<form method = "get" action = "../selectStatistic.jsp">
    <input type = "submit" value="Statistic">
</form>
```

Back to newStatistic.jsp!

```
INSERT INTO HotelDatabase.Hotel VALUES
(1, '123 Phoenix St', 'Phoenix', 'AZ', 'United States', 12345),
(2, '2 Ann Arbor Ct', 'Ann Arbor', 'MI', 'United States', 92834),
(3, '100 College Ave', 'New Brunswick', 'NJ', 'United States', 08901),
(4, '130 Jersey Rd', 'Trenton', 'NJ', 'United States', 09720);

INSERT INTO HotelDatabase.Room VALUES
(100, 1, 250.25, 4, 1, '4 person basic room', 'double', '2017-01-02', '2017-01-20', 0.30),
(101, 1, 250.25, 4, 1, '4 person basic room', 'double', '2017-01-02', '2017-01-20', 0.30),
(102, 1, 150, 2, 1, '2 person basic room', 'standard', '2017-01-02', '2017-03-03', 0.10),
(103, 1, 150, 2, 1, '2 person basic room', 'standard', '2017-01-02', '2017-01-20', 0.20),
(200, 1, 225, 2, 2, '2 person deluxe room', 'deluxe', '2017-01-02', '2017-02-02', 0.00),
(201, 1, 405, 2, 2, '2 person deluxe room', 'deluxe', '2017-01-02', '2017-02-02', 0.15),
(202, 1, 1200, 4, 2, '4 person deluxe room', 'deluxe', '2017-01-01', '2017-02-01', 0.00),
(300, 1, 2500, 4, 3, 'beautiful penthouse', 'suite', '2017-01-01', '2017-12-01', 0.00),
(301, 1, 2500, 8, 3, 'beautiful penthouse pt2', 'suite', '2017-05-01', '2017-05-20', 0.10),
(302, 1, 2000, 6, 3, 'beautiful penthouse yes', 'suite', '2017-05-10', '2017-06-10', 0.10);

INSERT INTO HotelDatabase.Room VALUES
(11, 2, 100, 4, 1, 'beautiful room yes', 'double', '2017-05-01', '2017-05-30', 0.20),
(12, 2, 100, 4, 1, 'beautiful room yes', 'double', '2017-12-01', '2017-12-15', 0.15),
(13, 2, 150, 2, 1, 'please reserve me', 'standard', '2017-02-01', '2017-02-13', 0.40),
(14, 2, 150, 2, 1, 'please reserve me', 'standard', '2017-02-01', '2017-02-13', 0.40),
(15, 2, 180, 2, 1, 'yes very pretty room', 'double', '0000-00-00', '0000-00-00', 0),
(16, 2, 180, 4, 1, 'please thank you', 'double', '2017-05-01', '2017-05-30', 0.10),
(17, 2, 400, 4, 1, 'super deluxe yay', 'deluxe', '2017-01-01', '2017-06-15', 0.05),
(18, 2, 400, 4, 1, 'springbok deluxe delivery', 'deluxe', '2017-01-01', '2017-01-31', 0.10),
(19, 2, 400, 4, 1, 'springbok bok bok', 'deluxe', '2017-01-01', '2017-01-05', 0.20),
(21, 2, 800, 8, 2, 'the best room ever', 'suite', '0000-00-00', '0000-00-00', 0);

INSERT INTO HotelDatabase.Room VALUES
(100, 3, 115, 4, 1, 'ew new brunswick', 'standard', '2017-01-01', '2017-01-20', 0.10),
(101, 3, 115, 4, 1, 'ew new brunswick', 'standard', '2017-01-01', '2017-01-20', 0.10),
(102, 3, 115, 4, 1, 'ew new brunswick', 'standard', '2017-01-01', '2017-01-20', 0.10),
(103, 3, 115, 4, 1, 'ew new brunswick', 'standard', '2017-01-01', '2017-01-20', 0.10),
(104, 3, 115, 4, 1, 'ew new brunswick', 'standard', '2017-01-01', '2017-01-20', 0.10),
(105, 3, 115, 4, 1, 'ew new brunswick', 'standard', '2017-01-01', '2017-01-20', 0.10),
(200, 3, 115, 4, 2, 'ugh new brunswick', 'standard', '2017-02-01', '2017-02-20', 0.10),
(201, 3, 115, 4, 2, 'ugh new brunswick', 'standard', '2017-02-01', '2017-02-20', 0.10),
(202, 3, 115, 4, 2, 'ugh new brunswick', 'standard', '2017-02-01', '2017-02-20', 0.10),
(203, 3, 115, 4, 2, 'ugh new brunswick', 'standard', '2017-02-01', '2017-02-20', 0.10),
(204, 3, 115, 4, 2, 'ugh new brunswick', 'standard', '2017-02-01', '2017-02-20', 0.10),
(205, 3, 115, 4, 2, 'ugh new brunswick', 'standard', '2017-02-01', '2017-02-20', 0.10),
(206, 3, 115, 4, 2, 'ugh new brunswick', 'standard', '2017-02-01', '2017-02-20', 0.10);

INSERT INTO HotelDatabase.Room VALUES
(100, 4, 600, 4, 1, 'ew new brunswick', 'deluxe', '2017-01-01', '2017-01-20', 0.10),
(101, 4, 600, 4, 1, 'ew new brunswick', 'deluxe', '2017-01-01', '2017-01-20', 0.10)
```

At this point, we have confirmed that we have valid inputs for our statistics, so now we can query everything.












To test out this section, I had to create fake data to test everything on. To do this, I manually inputted information into the database, which can be found in testpopulate.sql.

A shrunken version of some of the information manually entered can be found in the screenshot to the left.

To ensure that there was enough data to work with, I tried to randomize information. I also had to make sure that all the foreign key constraints matched, otherwise the values wouldn't insert.

A screenshot of the summary of information in HotelDatabase is also copied below. There are 4

hotels and at least 10 rooms in each hotel, as well as 10+ fake customers with one or more reservations each.

Name	Engine	Version	Row Format	Rows
 Breakfast	InnoDB	10	Compact	11
 BreakfastReview	InnoDB	10	Compact	38
 Contains	InnoDB	10	Compact	28
 CreditCard	InnoDB	10	Compact	8
 Customer	InnoDB	10	Compact	11
 Hotel	InnoDB	10	Compact	4
 HotelPhones	InnoDB	10	Compact	0
 Includes	InnoDB	10	Compact	38
 Reservation	InnoDB	10	Compact	14
 Reserves	InnoDB	10	Compact	14
 Room	InnoDB	10	Compact	46
 RoomReview	InnoDB	10	Compact	14
 Service	InnoDB	10	Compact	16
 ServiceReview	InnoDB	10	Compact	28

After all the information was manually entered into the database, I created SQL queries in MySQLWorkbench with placeholder dates that returned the values that the selectStatistic.jsp page would eventually return.

For example, one of the statistics to calculate was the top rated room type for each hotel. I first created a query to return all room types and ratings, grouped by hotel, then queried the top room type for each hotel, by average rating:

```

/* Returns table of HotelID, RoomType, and average table ratings as AvgRating */
SELECT temptable3.HotelID, temptable3.RoomType, avg(temptable3.Rating) AS AvgRating
FROM(
  SELECT HotelDatabase.RoomReview.HotelID, HotelDatabase.Room.RoomType, HotelDatabase.RoomReview.Rating,
  HotelDatabase.Reserves.InDate, HotelDatabase.Reserves.Outdate
  FROM HotelDatabase.RoomReview, HotelDatabase.Reserves, HotelDatabase.Reservation, HotelDatabase.Room
  WHERE HotelDatabase.RoomReview.HotelID = HotelDatabase.Room.HotelID AND HotelDatabase.RoomReview.RoomNumber = HotelDatabase.Room.RoomNumber
  AND HotelDatabase.RoomReview.CustomerID = HotelDatabase.Reservation.CustomerID AND
  HotelDatabase.Reserves.InvoiceNumber = HotelDatabase.Reservation.InvoiceNumber AND
  HotelDatabase.Reserves.HotelID = HotelDatabase.Room.HotelID AND HotelDatabase.Reserves.RoomNumber = HotelDatabase.Room.RoomNumber) temptable3
GROUP BY HotelID, RoomType;

SELECT temp11.HotelID, temp11.RoomType, max(temp11.AvgRating) AS AvgRating
FROM(
  SELECT temptable3.HotelID, temptable3.RoomType, avg(temptable3.Rating) AS AvgRating
  FROM(
    SELECT HotelDatabase.RoomReview.HotelID, HotelDatabase.Room.RoomType, HotelDatabase.RoomReview.Rating,
    HotelDatabase.Reserves.InDate, HotelDatabase.Reserves.Outdate
    FROM HotelDatabase.RoomReview, HotelDatabase.Reserves, HotelDatabase.Reservation, HotelDatabase.Room
    WHERE HotelDatabase.RoomReview.HotelID = HotelDatabase.Room.HotelID AND HotelDatabase.RoomReview.RoomNumber = HotelDatabase.Room.RoomNumber
    AND
    HotelDatabase.RoomReview.CustomerID = HotelDatabase.Reservation.CustomerID AND
    HotelDatabase.Reserves.InvoiceNumber = HotelDatabase.Reservation.InvoiceNumber AND
    HotelDatabase.Reserves.HotelID = HotelDatabase.Room.HotelID AND HotelDatabase.Reserves.RoomNumber = HotelDatabase.Room.RoomNumber
    AND HotelDatabase.Reserves.OutDate
    BETWEEN '0000/00/00' AND '2020/01/01') temptable3
  GROUP BY HotelID, RoomType) temp11
GROUP BY HotelID;

```

HotelID	RoomType	AvgRating
1	deluxe	6.5
1	double	10
1	suite	7.5
2	deluxe	5.75
2	double	6.5
3	standard	5.375
4	deluxe	8
4	suite	4.5

HotelID	RoomType	AvgRating
1	deluxe	10
2	deluxe	6.5
3	standard	5.375
4	deluxe	8

Which returned the following results printed to the left. The 2nd table, with only 4 rows, corresponding to the 2nd screenshotted SQL query, was what I wanted to hypothetically return on the selectStatistic.jsp page.

To convert the SQL query into the .jsp format, I first connected to the database, then created a string with the query, minus the date limit.

Since there were 4 different possible query formats, I had a conditional statement that did a different query for each statistics option. The logic used was the same for all 4 options, so I'll only go through how I implemented 1 query.

```
if (statoption2 == 1){
    // return highest rated room type for each hotel
    List<String> list = new ArrayList<String>();
    try {

        Statement queryStatement = con.createStatement();

        String getStat1 = "SELECT temptable3.HotelID, temptable3.RoomType, max(temptable3.AvgRating) AS AvgRating " +
            "FROM(SELECT temptable3.HotelID, temptable3.RoomType, avg(temptable3.Rating) AS AvgRating " +
            "FROM(SELECT RoomReview.HotelID, Room.RoomType, RoomReview.Rating, Reserves.InDate, Reserves.Outdate " +
            "FROM RoomReview, Reserves, Reservation, Room " +
            "WHERE RoomReview.HotelID=Room.HotelID AND RoomReview.RoomNumber = Room.RoomNumber AND " +
            "RoomReview.CustomerID=Reservation.CustomerID AND Reserves.InvoiceNumber=Reservation.InvoiceNumber AND " +
            "Reserves.HotelID=Room.HotelID AND Reserves.RoomNumber=Room.RoomNumber ";

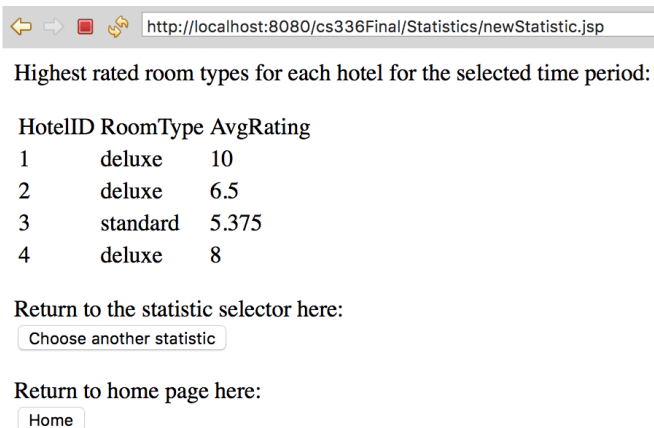
        String startDate = Integer.toString(startyear) + "/" + Integer.toString(startmonth) + "/" + Integer.toString(startday);
        String endDate = Integer.toString(endyear) + "/" + Integer.toString(endmonth) + "/" + Integer.toString(endday);

        String dateSelector = "AND Reserves.OutDate between '" + startDate + "' and '" + endDate + "'" + temptable3 GROUP BY HotelID,
            "GROUP BY HotelID";

        getStat1 = getStat1 + dateSelector;
    }
```

Since the start date and end date information was in integer format at this point (we converted it from a string to an integer at the beginning of newStatistic.jsp so that we could compare dates to check for errors), I converted them back to strings and concatenated them in the correct date format YYYY/MM/DD as startDate and endDate strings.

After inserting the dates into the SQL query string, the program sends the request to the connected database and parses the values into a table.



HotelID	RoomType	AvgRating
1	deluxe	10
2	deluxe	6.5
3	standard	5.375
4	deluxe	8

Return to the statistic selector here:

[Choose another statistic](#)

Return to home page here:

[Home](#)

As displayed on the left, a simple description of the statistic being viewed and options to return to the statistics page or the home page are displayed.

This particular query was for dates between 01/01/2013 and 01/01/2018 (so all valid entries in the database since all the dates were in 2017), and as you can see, it matches the SQL query created in MySQLWorkbench for the same date conditions.

A quick note about the date range: there was some confusion about what values should be returned if the selected date range to view statistics fell in the middle of a reservation. For example, if I wanted to view the highest rated rooms for 01/01/2017 through 01/08/2017 but there was a reservation for 01/04/2017 through 01/10/2017, should it be included in the average rating? I decided not to include them by simply comparing the date range with the OutDate of the reservation.

Since service, breakfast, and room reviews all have an InDate and OutDate, this is directly applicable to 3 of the 4 statistics. For calculating the top 5 customers, I referred to the date of reservation on the invoice.

Another important thing to note is for both the breakfast and service reviews, I returned every type of breakfast and service in the database as well as the ratings, sorted in descending order. Although the project description said to display the top rated breakfast and service (separately), sorting them in descending order makes it easy to see the top rated service since it's the first entry on the table.

It would be fairly easy to change this to just displaying the #1 rated breakfast/service by simply adding a "LIMIT 1" onto the already sorted SQL queries.