Homework 2

- These exercises use the hotel schemas shown with the Chapter 4 exercises.
- For each problem, formulate the appropriate mysql commands to solve the problem
- Please use the same naming conventions for Homework 2 that you used for Homework 1. To recap those conventions:
    a. Each query exercise in the homework should be in a separate .sql file.
    b. The name of the sql files should be the number of the question.
- If you need to write any sentences (or anything that's not valid SQL) in the answers, like in question 7, write them as SQL comments, otherwise the grading script will see it as a syntax error. You can look at this link for more information about commenting in MySQL: https://dev.mysql.com/doc/refman/5.1/en/comments.html.
- Some of you included the output to your SQL queries in your solution files. That is not valid SQL code so please do not do that—we have to edit it out and this time we will deduct points if we have to do so.
- When you test your queries, you should add or delete additional sample data to the Hotel schemas as needed. For example, when I was testing my queries for guest and room overbooking, I added sample data that overbooked guests and rooms, and when I was creating an archival table for the booking table, I inserted older dated booking tuples into the Booking relation.
- **Make sure that when we specify the order and names of columns in a view, that you use the same order and names. Since we will be doing script-based grading, it is essential that you do so in order for the outputs to match. If you have any doubt about the order of the columns and the names of the columns, look at the example output given with the problem.**

1. 6.18: Make sure that you print the details for all rooms, even unoccupied ones. If the room is occupied, then print the name of the guest staying in that room. **Hint: I made two views for this query and then used a certain type of join between the views to get my final result.**

```
+--------+---------+-----------+--------+--------------------+
| roomNo | hotelNo | type      | price  | guestName          |
+--------+---------+-----------+--------+--------------------+
|    100 |       1 | double    |  94.49 | Brad Vander Zanden |
|    100 |       1 | double    |  94.49 | Minnie Mouse       |
|    200 |       1 | family    | 115.49 | Daffy Duck         |
|    300 |       1 | king      | 142.28 | NULL               |
|    400 |       1 | penthouse | 944.99 | NULL               |
|    110 |       1 | double    |  36.75 | NULL               |
+--------+---------+-----------+--------+--------------------+
```

2. 7.11: mySql will not enforce the constraints, but it will check them syntactically. You can check to see whether or not your constraints might work in practice by running them independently of the create table command and seeing whether they would detect the prohibited condition. For example, you can run your overbooking query at the command line to check whether or not it would return overbooked rooms. More specifically, here's what I suggest you do:
    a. Parts a-d can all be written using check commands. These should not be commented out because mysql will parse them. However it will not enforce them.

b. For parts e-f write a select query that returns doubly booked rooms or guests who have double bookings.
    i. In order to test these queries, you will need to insert some tuples into your relations that create double bookings and doubly booked rooms.
    ii. Your queries will probably need to join the booking relation with itself in order to check for doubly booked rooms or guests who have double bookings. The reason is that you must find a way to compare two bookings, and a join will allow you to do so.
c. Once your select query works, couch it in a constraint using NOT EXISTS as shown on page 18 of the SQL DDL slides (http://web.eecs.utk.edu/~bvz/cs465/notes/Ch07-SQL-DDL.pdf).
d. Since mysql does not recognize the constraint command, leave it in your create table command but comment it out. You can look at this link for more information about commenting in MySQL: https://dev.mysql.com/doc/refman/5.1/en/comments.html.

3. This problem requires that you write three commands:
    a. Write a command that creates a separate table called **BookingOld** with the same structure as the hotel Booking table to hold archive records.
    b. Using the INSERT statement, copy the records from the Booking table to the archive table relating to bookings before 1st January 2003. Only move bookings where the guest has already checked out before 1st January 2003. If the guest is still staying at the hotel on 1st January 2003, do not copy them to the new table. The INSERT command has a form:

        INSERT INTO relation (SELECT …);

        where all tuples returned by the select query are inserted into the relation.
    c. Write a query that deletes all bookings before 1st January 2003 from the Booking table. Only delete a booking if the guest has checked out before 1st January 2003.

4. Create a view named **CurrentGuestCount** that contains a count of the number of guests currently staying at each hotel. Your view should contain columns for the hotelNo and the guest count as follows:

```
+---------+------------+
| hotelNo | guestCount |
+---------+------------+
|       1 |          3 |
|       2 |          2 |
+---------+------------+
```

5. Create a view named **HotelData** containing the names of all guests currently staying at one of our hotels and the names of the hotel at which they are staying. Order the results by hotel name. For example:

```
+--------------------+-----------------+
| guestName          | hotelName       |
+--------------------+-----------------+
| Minnie Mouse       | Grosvenor Hotel |
| Daffy Duck         | Grosvenor Hotel |
| Brad Vander Zanden | Grosvenor Hotel |
| Winnie The Pooh    | Holiday Inn     |
| Cinderella         | Holiday Inn     |
```

```
+--------------------+-----------------+
```

6. Create a view named **CheckingOutToday** that contains the account information for each guest at the Grosvenor Hotel who is checking out today. The account information should include the guest number, guest name, guest address, room number being checked out of, number of days spent in the room, and the total cost of the stay. You will need to use the DATEDIFF function to calculate the number of dates spent in the room and you will need to do some arithmetic in the field that represents the total cost of the stay. Finally you should remember that the checkout date is really equal to the dateTo field + 1.

```
+---------+--------------------+---------------+--------+---------+-----------+
| guestNo | guestName          | guestAddress  | roomNo | numDays | totalCost |
+---------+--------------------+---------------+--------+---------+-----------+
|      20 | Brad Vander Zanden | Knoxville, TN |    100 |       4 |    359.96 |
|      30 | Daffy Duck         | Knoxville, TN |    200 |       4 |    439.96 |
+---------+--------------------+---------------+--------+---------+-----------+
```

7. Consider the following view defined on the Hotel schema:

```
CREATE VIEW RoomBookingCount (hotelNo, roomNo, bookingCount)
AS    SELECT b.hotelNo, r.roomNo, COUNT(*)
        FROM Room r, Booking b
        WHERE r.roomNo = b.roomNo AND r.hotelNo = b.hotelNo
        GROUP BY b.hotelNo, r.roomNo;
```

For each of the following queries, state whether the query is valid and for the valid ones show how each of the queries would be mapped onto a query on the underlying base tables. Base your decision on whether a query is valid or invalid on the criteria presented in class and in the book, not on whether the query runs in mysql. All three queries shown below will run in mysql, but they may or may not be valid according to the SQL standard.

(a)
```
SELECT hotelNo, roomNo
FROM RoomBookingCount
WHERE hotelNo = 1;
```

(b)
```
SELECT hotelNo, SUM(bookingCount)
FROM RoomBookingCount
GROUP BY hotelNo;
```

(c)
```
SELECT *
FROM RoomBookingCount
ORDER BY bookingCount;
```

8. Required for graduate students, extra credit (15 points) for undergraduate students: Write an SQL query that prints each branch (i.e., the branchNo) whose staff count exceeds the average staff count for a branch, and print the amount by which each such branch exceeds the average staff count. Use the branch and staff relations from the DreamHome case study in the book. You should probably use an SQL variable to help simplify the query. You will need to set the SQL variable first, then execute the query.

```
+----------+-----------------------------------+
| branchNo | staffDiff                         |
+----------+-----------------------------------+
| B003     | 1.00000000000000000000000000000000 |
+----------+-----------------------------------+
```