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/5.7/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. List the details of all rooms at the Grosvenor Hotel, including the name of the guest staying in the room, if the room is occupied. 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 |
|    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. Create the Hotel, Room, Booking, and Guest tables using the integrity enhancement features of SQL with the following constraints:
    a. Type must be one of Single, Double, or Family.
    b. Price must be between $10 and $100.
    c. roomNo must be between 1 and 100.
    d. dateFrom and dateTo must be greater than today's date.
    e. The same room cannot be double booked.
    f. The same guest cannot have overlapping bookings.

g. The Booking relation must have appropriate foreign key constraints that specify the foreign key, what they reference, and what should be done on update and delete.

Here are some requirements for the problem:

1. You must submit a set of CREATE TABLE commands that will execute using the mysql command interpreter and create the four tables.
2. Place DROP TABLE IF EXIST statements at the top of your .sql file to DROP the tables if they already exist.
3. Don't use the CREATE DOMAIN keyword to create new types since mySQL does not currently recognize it.
4. mySql will enforce the foreign key constraints and the CHECK constraints for a-c. You should make the constraints for d-f be syntactically correct and then comment them out.

More specifically, here's what I suggest you do:

a. Parts a-d can all be written using check commands. The CHECK command for part d should be commented out because it requires you to use either CURRENT_DATE or CURDATE(), both of which are non-deterministic and hence not supported by mysql.
b. For parts e-f write a select query that returns doubly booked rooms or guests who have double bookings.
    a. In order to test these queries, you will need to insert some tuples into your relations that create double bookings and doubly booked rooms.
    b. 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. The TA will test your queries by copying and pasting them into the mysql command line interpreter and then running them to see if they correctly print out a list of doubly booked guests and doubly booked rooms.
    d. The query for doubly booked guests should print the guestName, roomNo/Hotel Names of the two rooms the guest double booked, and the booking dates for each of the two rooms. A good ordering would be

        (guestName, HotelName1, RoomNo1, dateFrom1, dateTo1, HotelName2, RoomNo2, dateFrom2, dateTo2)

    e. The query for doubly booked rooms should print the HotelName, roomNo, guestName1 and booking dates for guestName1, guestName2 and booking dates for guestName2. A good ordering would be

        (HotelName, RoomNo, GuestName1, dateFrom1, dateTo1, GuestName2, dateFrom2, dateTo2)

f. 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).
g. Since mysql does not allow sub-queries in a check 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.

c. For the foreign key constraints you should delete a booking if the hotel, room, or guest on which it is based is deleted and you should update the booking if the hotel, room, or guest is changed. Remember that foreign key constraints require that the attributes referenced by the foreign key in the parent relation be declared either as a primary key or as unique.

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.

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. Base your decision on whether a query is valid or invalid on the criteria presented in class and 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     | 1.000000000000000000000000000000 |
+----------+-----------------------------------+
```