University of Sussex Autumn 2019
Informatics

Databases
Assignment 2 (Deadline 12.12.2019, 4pm)

This coursework must be submitted electronically as a Canvas E-submission.

You have to answer *all* 12 questions.

The submission must consist of the single file `ass2.sql` that you download from the submission point in Canvas and into which you insert your code for each question after the corresponding question comment.

Don't write your name anywhere in the file but include your candidate number in the first line.

YOU MUST WORK ON THE ASSIGNMENT ON YOUR OWN! The standard rules for collusion and plagiarism apply and any cases discovered will be reported and investigated.

Detailed Instructions (follow carefully)
This assignment refers to an implementation of the hospital database as designed in the the first assignment (see Canvas).

To be able to answer the questions of this second assignment you must first run the SQL script `hospitalSetup.sql` that defines the tables that your code needs to rely on[1]. It is available from our Canvas site.

For the completion of this assignment it may be necessary to inspect the code in this script and understand how it implements the requirements outlined in the first assignment. Do not modify the structure of the tables in the given script when you write your answers.

Note that only a small number of data records have been inserted into the tables. In order to adequately test your code you will need to insert additional sample data. However, *do not include any of the test data or the corresponding insert statements in your submission*.

Also, you must *not* include the code of `hospitalSetup.sql` in your answer.

[1]Note that the implementation is a translation of the E/R model given in the model answers of Assignment 1, with a few minor deviations regarding the sub-entity types.

All your code *must run on our ITS server* where it will be tested for marking purposes. Use comments where appropriate which can be included e.g. like this:

```
-- this is a one-line comment.
```

You will automatically receive 0 marks for any answer that contains a syntax error[2], so please test your code carefully before you submit. If you are unable to answer a question write a comment or just skip the question. Format your code so that it is readable (this means in particular avoid putting long queries on one line).

Important: For questions 1-11 terminate your SQL statement for each question with a semicolon. For question 12 terminate your function declaration with the provided termination symbol $$.

1. Write a *single* SQL statement to set up a table according to the following (relation) schema:

   Hospital_MedicalRecord(recNo, patient, doctor, enteredOn, diagnosis, treatment)
   PRIMARY KEY(recNo,patient)
   FOREIGN KEY patient REFERENCES Hospital_Patient(NINumber)
   FOREIGN KEY doctor REFERENCES Hospital_Doctor(NINumber)
   diagnosis NOT NULL
   enteredOn NOT NULL

   Your code must execute without error assuming that all other tables have been set up by running script `hospitalSetup.sql`. The data types (domains) you choose for the columns should be most appropriate for the data they will contain. You must also accommodate the following requirements:

   (a) For table and column names you *must* pick the names used in the schema above (otherwise you will lose marks).

   (b) There are never more than 65,535 medical records for a single patient. The numbering of those records only uses positive integers.

   (c) The *enteredOn* column records date and time of when the medical record has been entered. It should have as default value the current date and time!

[2] This will be different in exams where you do not have an SQL tool to test your queries.

(d) The *diagnosis* column can contain some long text, but never more than 2^{24} bytes.

(e) The *treatment* column contains text, but is never more than one thousand characters long.

(f) Equip the FOREIGN KEY constraints, and only those, with constraint names `FK_patient` and `FK_doctor`, respectively.

(g) When a patient is deleted from the database, all their medical records shall be automatically deleted too.

(h) On the other hand, it should be possible to delete doctors who provided medical records without automatically deleting their medical records.

(i) Changing the NI number of a patient or doctor should not be permitted if they have or have produced, respectively, a medical record.

[16 marks]

Instructions for Queries

For Questions 2–11 specified below, write *one single* SQL query, respectively, that solves the task. You can use nested queries (also known as subselects or inner queries including subqueries) wherever you like. Note, however, that unnecessarily contrived answers might attract (mild) penalties, even if they are correct.

Important: It is essential that you choose column headings according to the exact instructions of the question and that you list columns in the exact order mentioned in the question. Where no explicit headers are mentioned, use the original column name.

Note that your queries must work correctly with any data (according to the schema) in the tables, not just the few records provided. All references to time, when not explicit, are relative and 'today' *always refers to the time of running the query.*

2. Add to the table *Hospital_MedicalRecord*, defined in Question 1, a column *duration* that stores the duration of the consultation that led to the record being taken. The duration is expressed in hours, minutes and (possibly) seconds. Choose an appropriate type for this column. You must not create a new table here. [5 marks]

3. Decrease the salaries of all doctors with any expertise in *ear* related matters by 10 percent. This means that the new salary must be $\frac{9}{10}$ of the old salary.

[6 marks]

4. List all the patients who live in a city which contains the string *right* (all lowercase!). List their first name, last name (with original column headings) and the (four digit) year of their birth with column heading *born*. Sort the result table alphabetically by last name. Rows with equal last name should be ordered alphabetically by first name.

[7 marks]

5. For all patients who have not had their 30th birthday yet list their national insurance number, first name, last name (all with original column headings) and their body mass index rounded up to *three* digits after the decimal point with heading *BMI*. The body mass index of a person equals $\frac{weightInKg}{heightInMetres^2}$. Please note that on our database the height is recorded in centimetres and the weight in kilograms.

[8 marks]

6. Compute how many doctors the hospital has. The heading of the single column in the result table must be *number*. [4 marks]

7. For each doctor list how many operations they have carried out this calendar year. The result table should contain *three columns*, the national insurance number of the doctor, the last name (both with original headings) and the number of operations with heading *operations*. Sort the result by the number of operations with the highest number appearing on top. [8 marks]

8. List all doctors who are not mentored by anybody, but are mentoring someone themselves. The result table should have *three* columns: the doctors' national insurance number (with original column heading), the (uppercase!) initial of their first name (one letter) with column heading *init*, and their last name (with original column heading).

[8 marks]

9. In the table `Hospital_Operations` the primary key guarantees that no two operations in the same theatre start at the same time. However, this does not automatically guarantee that two operation do not overlap. Therefore, list all pairs of operations that overlap.

The resulting table must have *three* columns: theatre number and start date&time for the first operation with headings *theatre* and *startTime1*, respectively, and the start time (only time, no date!) for the second (overlapping) operation in the same theatre with heading *startTime2*. Note that *startTime1* must be before *startTime2* to avoid duplicate listing of the same overlap pair. So, for instance, if your result table contains a row:

```
2 2016-12-02 9:00 11:11
```

then the result table must *not* include the (symmetric) row:

```
2 2016-12-02 11:11 9:00.
```

Hint: You may want to check out the single row functions for date and time in the MySQL manual. [8 marks]

10. For each operating theatre used for at least one operation, find out which day(s) had the most operations in it. The result table should have *five* columns: the theatre number (with original column heading), the day of the month (as number) with heading *dom*, the (English) name of the month (as string with first character capitalized) with heading *month*, the 4-digit year with heading *year* and the number of operations in the theatre on that day with heading *numOps*. So the result table may contain a row:

    ```
    34 3 October 2016 6
    ```

    if operating theatre 34 had 6 operations on the 3rd of October 2016 and never more than 6 on any other day. Note that a theatre can appear several times in this result table if there were several days that have been equally maximally busy. Sort the result table by theatre number (smallest first) and for equal theatre numbers chronologically by date (earliest first).

    Hint: You may need to include each of the different elements of the date appearing in the columns of the results table (i.e. day, month and year) in the relevant GROUP BY clause. [9 marks]

11. List those operating theatres that have seen more operations this May (meaning the month of May of this year) compared to last May (meaning the month of May of the previous year). The result table should have *four* columns: the operating theatre's number with original column name, its number of operations last year with heading *lastMay*, the number of operations this year with heading *thisMay*, and the corresponding increase with heading *increase*. The result table may, for

instance, contain rows that look like this:
`4 40 43 3`
if operating theatre 4 had 40 operations last May, 43 operations this May and thus an increase of 3. Sort the result table by the increase in operations with the highest increase on top. [9 marks]

Instructions for Stored Procedures

For the following creation of stored procedures/functions the delimiter has to change. This has been done for you already in the template, so use delimiter $$ specified there. Don't forget to test your function. Even if the declaration is successful the function might still throw an error during execution. Any function that throws an SQL runtime error may receive 0 marks. *Do not include any test code in the submission though, just the function declaration.*

12. Write a stored function `usage_theatre` that, given a theatre number and a year (specified by a positive four digit integer number), computes the total time the given theatre has been occupied for (due to operations) in the specified year. The total time computed should be returned as a string. You must accordingly choose an appropriate result type for your procedure. The result string must look like this: `125days 9hrs 32mins` where the concrete numbers of course depend on the data.

    There are a number of 'unwanted' cases that you need to deal with. If the given year is in the future the result string must be: `The year is in the future` (no period at the end). If it is not in the future but the operating theatre, say 42, does not exist the result string must be: `There is no operating theatre 42`. If, however, the year is not in the future and the operating theatre exists, but there are no operations in the year specified, then the result string must be: `Operating theatre 42 had no operations in 1066` (in case the theatre was 42 and the year was 1066). Please make sure you use the exact spelling and spacing of the result strings as given in the examples above.

    [12 marks]