

Homework 3

assigned November 6th; due November 20th 5:00pm

You can work in groups of up to 3 people. When you submit the assignment, do not forget to indicate the section you are in. Groups of students from different sections are **not allowed**. If such an assignment is submitted, it will be arbitrarily assigned to one section, and student(s) from the other section will get no marks for it.

You must submit electronically (one submission per group). The electronic submission must contain the following files (with these exact name):

- group.txt (text file with names and student numbers of all members of the group)
- a3.ddl (ddl instructions to create the database)
- populate.sql
- query1.sql
- query2.sql
- query3.sql
- query4.sql
- query5.sql
- update.sql
- Makefile

You must also submit a hardcopy of the solution for the Assertions, Functional dependencies and Normalization.

Evening Section No pencil assignment will be marked. You must use the front page that can be found on the web page.

1 Embedded SQL (75 points)

This part of the assignment must be done on db2.

The database schema

You will be using part of the schema from assignment #2.

```
person(sin, name, dob)
doctor(name, department, hospital, salary)
patients(sin, disease, hospital, fee, entry_date, exit_date)
treat(doctor_name, patient_name)
```

The attributes `dob`, `entry_date` and `exit_date` are of the type `DATE` and `fee` and `salary` are of the type `REAL`. To keep it simple, we make the rest of the attributes of the type `CHAR(10)`.

You are to write a number of programs using Embedded SQL in C. Your programs will insert, update and query the underlying database. Your programs will take input parameters and output the answer to `STDOUT` as a single column of values.

Create the database

This part is worth **5 points**.

You need to create the four tables listed in the schema. Make sure you define the following key constraints:

- `person.sin` is a primary key of `person`.
- `doctor.name` is a primary key of `doctor`.
- `patients.sin` is a foreign key of `person.sin`.
- `treat.doctor_name` is a foreign key of `doctor.name`.

You can simply create the tables using SQL.

SQC Programs

Each is worth **10 points**.

1. `populate.sqc` :

You need to write an embedded SQL C program `populate.sqc` which is used to populate the tables.

You need to compile the `.sqc` file into an executable `populate`. Its syntax is

Syntax:

```
populate <table> <val> <val> <val> ...
```

For instance, to insert a row into `treat`, one can call your program:

```
populate treat "Dr. Who" "J. Doe"
```

and to insert into the table `patient`,

```
populate patients 2001 Sleepy "HSC" 100.00 "1/1/2003" "1/3/2003"
```

2. `query1.sqc`:

It prints the names of patients with the specified disease.

Syntax:

```
query1 <disease>
```

The output format (when successful) is simply:

```
name1  
name2  
name3  
...
```

3. `query2.sqc`:

It prints the names of patients who are treated by the specified doctor.

Syntax:

```
query2 <doctor_name>
```

The output format (when successful) is simply:

```
name1  
name2  
name3  
...
```

4. `query3.sqc`:

It prints the names of doctors who treat themselves. This program does not take any input parameters. The output format (when successful) is simply:

```
name1  
name2  
name3  
...
```

5. `query4.sqc`:

It prints the number of patients whose stay is between a specified period, and also the average length of their stay. Round up the average to the nearest integer, so if the answer is 9.23, report 10.

Syntax:

```
query4 <begin_date> <end_date>
```

So only these patients with the `entry_date ≥ begin_date` and `exit_date ≤ end_date` are considered.

Output:

```
<total number of patients>  
<average duration of their stay>
```

6. `query5.sql`:

An idle period for a hospital is the days in which no patients entered that hospital, and the number of such days is the duration of the idle period. Given a hospital, find the average length of the idle periods of that hospital in days (round up to the nearest integer).

Syntax:

```
query5 <hospital>
```

Output:

```
<average length of the idle periods>
```

7. `update.sql`:

This program increases the fees of each patient in the `patients` table by a specified amount.

Syntax:

```
update <amount>
```

Since this is an update, it has no output.

Important remarks

Part of the marking procedure will be automated, so it is especially important that you use the correct parameters and output the results in single columns.

Make sure that you respect the foreign key / primary key constraints. You do not need to enforce these constraints inside your `.sql` programs, instead specify these constraints when you create the table. When an invalid instruction is being executed, your programs should catch and display the error.

2 Assertions (30 points)

Write the following assertions in SQL. Each is worth **10 points**. You do **not** have to implement this part of the assignment on db2.

1. Each doctor has (or had in the past) at least one patient.
2. The total amount of fees a hospital charges each patient is less than \$100,000.
3. Doctors do not treat themselves, unless their salaries exceed \$100,000.

3 Functional dependencies (25 points)

Problem 1 (10 points). Consider the following set of FDs on $ABCDEFG$: $A \rightarrow CE, EF \rightarrow ADG, BG \rightarrow CEF, BF \rightarrow AD, D \rightarrow ABG$.

Compute the closures of the following sets:

- D ;
- AB ;
- CFG .

Find the candidate keys for this set of FDs.

Problem 2. Let (U, F) be a schema. Recall that a set $V \subseteq U$ is *closed* if $V^+ = V$. Prove that

- the intersection of two closed sets is closed (**10 points**), and
- no candidate key is closed, unless $F = \emptyset$ (**5 points**).

The total length of both proofs should not exceed 15 lines (if it does, you will automatically get zero points).

4 Normalization (20 points)

Consider the schema of Problem 1 in the **functional dependencies** section.

Find a lossless BCNF decomposition of this schema. If it is not dependency-preserving, find a 3NF decomposition as well. You must explain why your BCNF decomposition is (or is not) dependency-preserving.