CS145 Written Assignment #5

Due Thursday May 13*

You are hired by Mr. Burns to design a personnel database for the Springfield Nuclear Power Plant (SNPP). After a briefing by Smithers, you quickly came up with the following schema:

```
Dept(DNo, name, budget)
Emp(SSN, name, salary, DNo)
Mgr(SSN, assistantSSN)
```

SNPP is divided into multiple departments. Each department has a name and a budget, and is uniquely identified by a department number (`DNo`). Each employee has a name and a salary, and is uniquely identified by a social security number (`SSN`). Each employee works for exactly one department. Certain employees are managers. Managers manage the departments they work for, and each manager has exactly one administrative assistant.

1. Your relational design basically followed the E/R-style translation of the subclass relationship between managers and employees. Another approach is to use the NULL-style translation of subclasses, and replace `Emp` and `Mgr` with one table `AllEmpInfo(SSN, name, salary, DNo, assistantSSN)`, where `assistantSSN` is `NULL` for employees who are not managers.

   You explained the tradeoff between the two approaches to Mr. Burns. He absolutely loathed the second approach because it would store meaningless `NULL`'s and waste his precious disk space. Nevertheless, he still wanted to write queries over `AllEmpInfo` instead of `Emp` and `Mgr`! What can you do to keep your job? (*Hint:* scene, prospect, extent or range of vision, etc.)

2. There is a relational algebra operator called *outerjoin* that we have not covered in class. The (full) outerjoin between two relations R and S, denoted $R \mathbin{\supset\!\!\bowtie\!\!\subset} S$, is defined as follows: First, we take the natural join $R \bowtie S$. For each "dangling tuple" in R (i.e., a tuple that does not join with any tuple in S), we "pad out" this tuple with `NULL`'s in those attributes belonging only to S, and then add it to the result. Similarly, for each dangling tuple in S, we also pad it with `NULL`'s and add it to the result. For example:

   R:

A	B
a_1	b_1
a_2	b_2

   S:

B	C
b_1	c_1
b_3	c_3

   $R \mathbin{\supset\!\!\bowtie\!\!\subset} S$:

A	B	C
a_1	b_1	c_1
a_2	b_2	NULL
NULL	b_3	c_3

   In a *left outerjoin* between R and S, only the dangling tuples in R are padded with `NULL`'s; in a *right outerjoin* between R and S, only the dangling tuples in S are padded with `NULL`'s.

   (a) In Problem #1, `AllEmpInfo` is really just the left outerjoin between `Emp` and `Mgr`. So you have already figured out how to do a left outerjoin in SQL. Now, given two relations $R(A, B)$ and $S(B, C)$, please write the the *full* outerjoin $R \mathbin{\supset\!\!\bowtie\!\!\subset} S$ in SQL.

*Please refer to CS145 Course Information Page (`http://www.stanford.class/cs145/info.html`) for submission instructions and late policy.

(b) (Complete this part if you want to get a "+" for this assignment.) Is the outerjoin operator associative? That is, given any three relations R, S, and T, does $(R \mathbin{⟗} S) \mathbin{⟗} T = R \mathbin{⟗} (S \mathbin{⟗} T)$ always hold? If yes, sketch a simple proof; otherwise, show a counterexample.

3. You are now ready to create the database schema in SNPP's Megatron 2000† RDBMS. Here are the basic `CREATE TABLE` commands:

```
CREATE TABLE Dept(DNo INTEGER PRIMARY KEY,
                  name CHAR(30), budget FLOAT);
CREATE TABLE Emp(SSN CHAR(9) PRIMARY KEY,
                 name CHAR(30), salary FLOAT, DNo INTEGER);
CREATE TABLE Mgr(SSN CHAR(9) PRIMARY KEY,
                 assistantSSN CHAR(9));
```

(a) "Assistants to managers are also employees." Show how to modify the `CREATE TABLE` commands to encode this referential integrity constraint.

(b) "Each department has a unique name." Unfortunately, Megatron 2000 does not support `UNIQUE` key declarations. Therefore, you need to write an attribute-based check to encode this constraint.

(c) "Those who work in Department 13 have a salary cap of $20,000." Write a tuple-based check to encode this constraint.

(d) "If a department has a budget less than $1,000,000, then it must have no more than two managers." Write a SQL assertion to encode this constraint.

(e) "The guy named Homer Simpson will never get a raise!" Write a trigger to enforce Mr. Burns' wish.

(f) Just when you are about to enter your schema, Smithers informs you that Megatron 2000 does not support `FOREIGN KEY` declarations. To correctly implement the referential integrity constraint in (a), which of the following alternatives can you use?

   i. An attribute-based check on `Mgr.assistantSSN`
   ii. An attribute-based check on `Mgr.assistantSSN` together with an attribute-based check on `Emp.SSN`
   iii. A general assertion
   iv. Triggers

For each alternative, if your answer is no, give a short explanation. For (i)–(iii), if your answer is yes, please also provide an implementation. For (iv), if your answer is yes, please specify all possible triggering events; however, you do not need to show the full implementation.

4. (Complete this part if you want to get a "+" for this assignment.) Views in SQL are *virtual*, meaning that they are not stored in the database but rather their definitions are used to translate queries referencing views into queries over base relations. One disadvantage of this approach is that views may effectively be computed over and over again if many queries reference the same view. An alternative approach is to *materialize* views: the contents of a view are computed and the result is stored in a database table, so that a reference to the view in a query can simply access the stored table. However,

† To learn more about Megatron 2000, take CS245!

when contents of a base relation referenced in the view change, then the contents of the materialized view must be modified accordingly.

Mr. Burns has asked you to create a materialized view `DeptStat` showing the number of employees and their average salary for each department in SNPP. Initially, `DeptStat` is created and populated using the following SQL statements:

```
CREATE TABLE DeptStat(DNo INTEGER PRIMARY KEY,
                      numEmps INTEGER, avgSal FLOAT);
INSERT INTO DeptStat
  (SELECT   DNo, COUNT(*), AVG(salary)
   FROM     Emp
   GROUP BY DNo);
```

Now when we refer to `DeptStat` in queries we obtain the desired result. Obviously, when `Emp` is modified, `DeptStat` must be modified accordingly. Write a trigger to modify `DeptStat` after an `INSERT` statement is executed on `Emp`. You may assume that there is a `NOT NULL` constraint on `Emp.salary`. You should try to make your trigger as efficient as possible. Simple recomputation of the entire `DeptStat` view is not efficient enough. *Hint:* Your trigger does not need to access `Emp`; it only needs to access the newly inserted tuples and `DeptStat` itself.