Database Systems, CSCI 4380-01
Homework # 6
Due Monday October 29, 2018 at 11:59 PM

Introduction.

This homework is worth 4% of your total grade. If you choose to skip it, Midterm #2 will be worth 4% more.

This homework is on writing procedural SQL and insert/update/delete statements. It requires you to take a complex task, break it into computationally inexpensive steps and put all of it into a single procedure. You are free to use any SQL construct, any number of steps and statements. There is no penalty for long or short solutions, just get the job done.

Your homework will use a database of tables (a different subset of the jeopardy database). Since you are changing data, you will use your own database (`db_username`), created already for you and populated with this data.

There are only a few rules:

- Do not create tables or procedures in any other database, other than your own.
- Do not change the already given tables as it will be costly for me to recreate them. Do all your work by creating new database objects as described below.
- Only create the tables and procedures described and nothing else. If you need additional tables for your work, use temporary tables that are erased when the session is over. This will make testing easier for us.

Homework Description

The data model is the same for this homework as in the previous two homeworks.

Before you start. Add to your solution script the following commands that create copies of two tables and alters one of them.

```
create table scores2 as select * from scores ;
create table contestants2 as select * from contestants ;
alter table contestants2 add iswinner boolean ;
```

Problem 1. We are tired of finding who is the winner of a game, something that does not change over time. So, it is best to store this directly and use it in queries, instead of recomputing it every time.

Update the `contestants2` table you just created so that iswinner has value `True` if that contestant is the winner of the game and `False` otherwise. To find a winner, find contestants such that there is no other player with a higher final score. Also, players who do not have a final score in the database will count as winners.

Note that if two players are tied, they will both count as winners for the purposes of this problem.

Problem 2. We also would like to compute which games were consecutive. We do not know this for sure because of missing data. However, there are a few things we can compute:

- If two dates are consecutive, then clearly one came after the other.

- If one date is a friday and the other one is the following monday, then they are consecutive. (You can use `extract(isodow from airdate)` to find the airdate.

- If two dates `d1, d2` are not consecutive according to the above rules, but there is no other game between these dates and one of the contestants from the game in date `d1` is also competing in date `d2`, then these games are consecutive. Note that to reduce complexity, we will only consider games less than 60 days apart.

Based on these rules, create and populate a new relation with the schema:

`follows(gameid1, gameid2)`

where `gameid2` is the id of the game directly following the game given by `gameid1`.

Problem 3. Now create an after trigger called `scores_trigger` on update of the `scores2` table. Whenever a score is updated, if it is the final round score, then you must update `contestants2` table to make sure that the `iswinner` attributes are correct for the contestants of this game.

To create this trigger, you will need to create a function in the Postgresql syntax. Name this function `scores_trigger_f`.

Problem 4. I guess my parsing was not so good as we are missing a lot of data. To understand where to start debugging, I would like to find out where my parsing has failed the worst.

To help me, write a single pl/pgsql procedure called:

`parsingreport(input_year)`

that takes as input an integer representing a specific year. Your procedure should return a report regarding what data is missing in each year:

1. How many contestants have zero responses (to any regular and final clue) in the database for a game they participated in.
2. How many total clues are missing (including regular and final clues).

   Note that each game should have a total of 1+2x6x5= 61 clues. Use this to compute how many clues there should be.

   It is possible that a clue is missing because it was never revealed, but it is much more likely to be parser error.
3. How many responses to final jeopardy clues are missing

   Each contestant with a positive score (above zero) coming into the final jeopardy (i.e. in Round 3) must have a response for the final clue in the database. Exclude in your computation any response from a contestant with a missing final score or a negative score in round 3.

   If a final clue is missing, exclude any responses to that clue in your computation as well.
4. Your function should return a single string that reports all of these statistics.

Note that for this question, you can use the original tables as you will be querying them but not changing them.

Submission Instructions.

Submit a single ASCII text file named `username_hw6ans.sql` that contains all your queries to SUBMITTY. It should have the same format as previous homeworks, as shown below.

```
-- Print your answer and RCS id first
SELECT 'Student: Sibel Adali (adalis@rpi.edu)';

create table scores2 as select * from scores ;
create table contestants2 as select * from contestants ;
alter table contestants2 add iswinner boolean ;

SELECT 'Problem 1';

-- Insert your solution to Problem 1 here

SELECT 'Problem 2';

-- Insert your solution to Problem 2 here
-- Your solution should only create a table
-- follows(gameid1, gameid2)
-- and populate it, and nothing else.

SELECT 'Problem 3';

-- Insert the code for creating your trigger for Problem 3 here.
-- Your code should only create a trigger named
-- scores_trigger and a function named scores_trigger_f
-- and nothing else

SELECT 'Problem 4';
-- Insert the code for creating your function for Problem 4 here.
-- Your code should only create a single function named
-- parsingreport(input_year)
-- and nothing else
```

Database Schema

```
-- Each game is in a season, given by id
CREATE TABLE games
     ( id INT  -- season id
       , gameid INT
       , airdate DATE
       , PRIMARY KEY (gameid)
     ) ;

-- Each contestant is identified by a shortname, which is unique for a
-- game.

CREATE TABLE contestants
     ( gameid INT
       , fullname VARCHAR(100)
       , description VARCHAR(255)
       , shortname  VARCHAR(100)
       , PRIMARY KEY (gameid, shortname)
       , FOREIGN KEY (gameid) REFERENCES games(gameid)
     ) ;

-- The overall scores of each contestants after different rounds
-- of the game.
-- Rounds '1', '2' are in the first stage called the 'Jeopardy' stage,
-- Round '3' is after 'Double Jeopardy' before 'Final Jeopardy'.
-- Round 'Final Score' is the actual score of each person
-- Round 'Coryat Score' is the hypothetical score without the bets
-- Round '6' is an error, which needs to be identified later.

CREATE TABLE scores
      ( gameid INT
        , shortname VARCHAR(100)
   , score INT
   , round VARCHAR(20)
   , PRIMARY KEY (gameid, shortname, round)
   , FOREIGN KEY (gameid, shortname)
     REFERENCES contestants(gameid, shortname)
      ) ;

-- Each game has many clues, clue is the question, and correct_answer is the answer
-- value is the dollar value of the clue: amount player wins/looses
-- for correct, incorrect answers
-- category is the named of the category
-- cat_type is one of: 'J': 'Jeopardy' round and 'DJ': 'Double Jeopardy' round
-- isdd is true if the question was a double jeopardy question

CREATE TABLE clues
     ( gameid INT
       , clueid INT
       , clue   TEXT
       , value  INT
       , category VARCHAR(255)
       , cat_type VARCHAR(10)
       , isdd     BOOLEAN
       , correct_answer VARCHAR(255)
       , PRIMARY KEY (gameid, clueid)
       , FOREIGN KEY (gameid) REFERENCES games(gameid)
```

```
     ) ;

-- Each contestant can answer a clue, if the answer is wrong,
-- another contestant can answer. This relation stores all
-- contestants who gave a response (but not what they said).
-- If there is no correct answer for a question here, it means
-- that no contestant answered the question correctly.

CREATE TABLE responses
     ( gameid INT
       , clueid INT
   , shortname VARCHAR(255)
   , iscorrect BOOLEAN
   , PRIMARY KEY (gameid, clueid, shortname)
   , FOREIGN KEY (gameid, clueid) REFERENCES clues(gameid, clueid)
   , FOREIGN KEY (gameid, shortname)
     REFERENCES contestants(gameid, shortname)
     ) ;

-- At the end of the game, there is a single question/clue called
-- the 'Final Jeopardy'. This relation stores the clues for this
-- specific round. There is no dollar value attached to these questions.

CREATE TABLE final_clues
     ( gameid INT
       , clue   TEXT
       , category VARCHAR(255)
       , correct_answer VARCHAR(255)
       , PRIMARY KEY (gameid)
       , FOREIGN KEY (gameid) REFERENCES games(gameid)
     ) ;


-- For the 'final jeopardy', all contestants give an answer and a bet
-- The bet is the dollar amount the contestant will win/loose if they
-- answer correctly. Only contestants with positive winnings/scores
-- at round '3' can participate. This relation stores the bets and
-- whether each person scored correctly or not.

CREATE TABLE final_responses
     ( gameid INT
   , shortname VARCHAR(255)
   , iscorrect BOOLEAN
   , bet  FLOAT -- VARCHAR(10)
   , PRIMARY KEY (gameid, shortname)
   , FOREIGN KEY (gameid, shortname)
     REFERENCES contestants(gameid, shortname)
     ) ;

CREATE TABLE states
     (  name VARCHAR(50)
        , population INT
     ) ;
```