CSE 532: Project 1 - A Talent Database (TDB)

Spring 2017

Deadline: February 27

In this project, you will be designing and implementing part of a database for keeping information about talent search shows.

You will be using the Datalog version of the Flora-2 system (`http://flora.sourceforge.net`).[1] The lecture slides provide a brief introduction to the use of Flora-2. You are *not* required to build any GUI or store any data in a DBMS (we do not have the time to discuss those aspects of Flora-2). Instead, you should keep all the data, rules, *and* queries in *one* file. When the file is loaded, like this:

```
myprompt> ..../flora2/runflora

flora2 ?- [myproj1].
```

or, equivalently, `..../flora2/runflora -e "[myproj1]."`

All queries are to be run as part of this single loading command and all answers to the queries are to be printed out like this (or something similar) automatically:

```
  ---- Query 1 -----
answer1
answer2
...

  ---- Query 2 ----
answer1
answer2
...

etc.
```

A way to do this is explained in a later part of this assignment.

1 General Description

The TDB database contains information about *shows*, *auditions*, *contestants*, *judges*, and *artworks*. It is assumed that you are familiar with talent search shows in general terms but here is a brief reminder.

An *artwork* can be a dance, a song, etc. A *show* is a show held on a particular date where contestants audition by performing artwork in front of the judges (usually 2 to 5). An *audition* is an event in a show when a contestant performs a particular artwork. We assume one can perform several artworks during the same show, but it is reasonable to also assume that the same artwork cannot be auditioned more than once by the same contestant during the same show. Each *judge* rates each audition on the scale of 1 to 10.

[1] You can also use the free 3 months trial of Ergo `http://coherentknowledge.com/free-trial/`, which is a commercial version of Flora-2. It has many extensions (which you probably won't need), but it also has an IDE, which you might find useful.

2 Required Data

The data items required by your system roughly fall into these categories:

- *Information about shows* like the name, the date, the judges, the contestants.
- *Information about contestants and the judges* such as the name, Id, etc.
- *Information about artworks*, including the name, the genre, Id.
- *Information about auditions*, which says who performed which artwork and when.

To represent the dates, you can use integers of the form YYYYMMDD. This way you will be able to compare dates simply by using the ">" builtin for integers.[2]

Note that the data model underlying Datalog is essentially relational, so the design of the relations for Project 1 should follow the best practices of relational database design.[3] The structure of the above information is such that naïve ways of organizing the data in relations will cause violation of 4NF (Fourth Normal Form). So, you should normalize the schema.

3 Queries

You are to implement the following queries. For some queries you would need to use negation (`\naf`), aggregate operators (e.g., `sum`{... | ...}), quantifiers (`forall`(...)^... and `exist`(...)^...), and recursion. You might also need to use some builtins like =, ! =, >=, `\is`, @ >, @ <, etc.[4]

In the queries below, a "pair of contestants" always means a pair of *different* contestants.

1. Find all pairs of contestants who auditioned the same artwork on the same date and got the same score from at least one judge (not necessarily the same judge).

   This is a warm-up rule.

2. Find all pairs of contestants who happened to audition the same artwork (in possibly different shows) and got the same maximal score and the same minimal score *for that audition* (from possibly different judges).

   This query involves **aggregates**.

3. Find all pairs of contestants who auditioned the same artwork in (possibly different) shows that had the same number of judges and the two contestants received the same average score *for that audition*.

   This query also involves **aggregates**.

[2] Alternatively, you can use the builtin data type *\date* described in the Flora-2 manual `http://flora.sourceforge.net/docs/floraManual.pdf`, but this is not required and would be an overkill in this case.
If you chose to use Ergo, the documentation is at `http://coherentknowledge.com/ergo-documentation/`. Although the Flora-2 manual will suffice for your project for either system, the above Ergo documentation page includes a tutorial, which you might find useful.

[3] Flora-2 also supports the object-oriented and mixed object-oriented/relational models, but in this project we will not use these advanced features.

[4] To remind, @ > and @ < compare strings lexicographicly. Some of the queries below are symmetric, which means you would be getting both (a,b) and (b,a) as answers. To avoid this, use the aforesaid lexicographic comparisons.

4. Find all pairs of contestants (by name) such that the first contestant in each pair performed in all the shows in which the second contestant did (possibly performing different artworks).

   Write this query using *explicit* quantifiers (`forall` and *exists*). **All** variables that do not occur in the query's rule head **must** be quantified explicitly.

5. Find all close rivals. The *close rivals* relation is the transitive closure of the following binary relation: X and Y are *direct close rivals* iff they both performed the same artwork in the same show and their overall average scores are within 0.2 of each other.[5]

   This query involves **recursion** and aggregation.

Note that Queries 1, 2, 3, 5 are *symmetric*, i.e., if a pair (A,B) is an answer then so is (B,A). Your answers should include **only** the pairs in which the first name is lexicographicly less than (@ <) the second, i.e., redundancy due to symmetry should be eliminated.

4 Code Repository

The source code of your project **must** be maintained in a **private** cloud code repository on *Bitbucket*, so get your free Bitbucket user Id **now**.
Note: Only Bitbucket is acceptable, as we have no time to fish out the different repositories out of multiple clouds.

It is **important** to make frequent check-ins to the repository both as a demonstration of professionalism and also to document your history of project development. **Projects with no history will be rejected outright.** Those with insufficient history will get lower grade. We will be checking your repository and your commit logs to make sure that substantial activity has been taking place over a period of time.

It is strongly recommended to not use command line interface to Git. Instead, use a graphical tool such as SourceTree (Windows, Mac), TortoiseGit (Windows), Smartgit or GitKraken (both work on Windows, Mac, Linux). All these tools are free for academic use and some are open source.

Important: Your repository **must** be named following this naming schema:

```
CSE532p1-YourBitbucketId-YourLastName
```

It **must** be shared (in **read** mode) with me—and **nobody else**![6] My Bitbucket Id is:

- **kifer-sbu**

Note: Share ONLY with the above Id. I have one other Id on Bitbucket, but only the above will be checked for project submissions.

5 Documentation and Submission Instructions

Your code must be well-structured, documented, properly indented, and there must be **no compilation warnings**. These factors will be taken into account in grading.

Your project should include a short document, which should appear at the top of the program file between the comment markers /* ... */. The document should detail the database schema used for

[5] This condition can be written as $0.2 >= abs(?Sc1 - ?Sc2)$, where $?Sc1$, $?Sc2$ are variables, which your query (or rule) should bind to the average scores of various contestants in the talent search show.

[6] The TA will not be involved in Project 1.

the project (i.e., relation names, the meaning of the columns, and their types). Also, state what you expect to be the keys in each relation. You do not need to express them in Datalog (we did not discuss the representation of the schema). Use some kind of informal description (not SQL—this would be too verbose). For instance,

```
Contestant(Id, Name, ...).
Key: Id.
```

Tables must be in 4NF (4th Normal Form). If a lower normal form is chosen, the choice must be justified.

In order to express some queries you might need to include rules that define intermediate relations. In some cases, the intent of those rules may not be obvious and you would have to explain their meaning with a comment line or two. But the best practice is to convey the meaning via the names of the relations and the variables so that the rules would be **self-explanatory.**

Pay attention to the aesthetics. Poorly formatted/designed works will be penalized. A typical layout for Datalog rules is:

- Comments precede the respective rules.
- Single line rules, must be shorter than 80 characters.
- Multi-line rules. These have the following layout:

```
// This is a multi-line rule.
rulehead :-
    predicate_1,
    predicate_2,
    ...,
    predicate_n.
```

You will submit the project via the Blackboard system: go to the *Assignments* area and follow the instructions, which can be found at
`http://it.stonybrook.edu/sites/it.stonybrook.edu/files/kb/6105/assignments_students91_1.pdf`
The data, the rules, and the queries should be in **one** `.flr` file. Upload the file as described at the above URL. The file must be directly loadable and executable in Flora-2 without any editing, i.e., your program should be runnable by simply typing

```
..../runflora  -e "['yourFile']."
```

Here is one of the possible templates for `yourFile.flr`:

```
// Code
Query1(?X,?Y) :- ....
........
Query2(?A,?B) :- ....
........
........
........
// Queries
?- writeln('\n--- Query 1 answers ---')@\io,
```

```
     Query1(?F,?G).
?- writeln('\n--- Query 2 answers ---')@\io,
     Query2(?F,?G).
.......
```

The presence of the queries embedded in the file, as shown above, will ensure that the queries will run automatically after the file is loaded and the answers will be shown on the screen like this:

```
--- Query 1 answers ---
?F = 1
?G = 3

?F = 21
?G = 7
...
4 solutions ...
Yes

--- Query 2 answers ---
?F = 3
?G = 9
....
8 solutions ....
Yes
```

In the assignment submission textbox on Blackboard, provide the URL for your **private** git repository, which should be **shared with me** (in read mode).

6 Teaming

This project must be done **individually** — no partners.
At the top of the program file (in a comment block) include your name, student Id, email, and this statement:

> **I pledge my honor that all parts of this project were done by me individually and without collaboration with anybody else.**

7 Planning Your Work

This is not a difficult project, but, since Datalog and Flora-2 are likely to be new to you, you will encounter numerous problems trying to get things done. Therefore, start right away and do not delay.

A test dataset is provided. However, you might need to add more information to ensure that your queries have enough data to work with. Note: obtaining correct answers on the test data does *not* guarantee that your queries are *logically* correct. It should be obvious that one can *always* write a *wrong* query that will give the right answer for any fixed given input (but not for all valid inputs).

8 A Note on Copy/Pasting from PDF Documents

If you try to run pieces of code by copy/pasting them from PDF or Word documents, such as this document or the Flora-2 manual, you may discover that the system issues parser errors. This happens because some PDF characters, such as space, quotes, double quotes, minus, etc., may *look* like their ASCII keyboard equivalents but, in fact, they may be completely different Unicode, Latin-1, or CP-1252 characters. So, you may have to erase and retype those characters.