# CSE 532: Project 1 - A Pharmaceutical Database (PharmDB)

Fall 2021

Deadline: September 27

In this project, you will be designing and implementing part of a database for keeping information about clients of a pharmacy.

You will be using the Datalog subset of the Flora-2 system (`http://flora.sourceforge.net`). The lecture slides provide a brief introduction to the use of Flora-2.[1] 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

PharmDB contains information about customers, drug prescriptions, prices, and interactions among drugs. The system supports the following functions:

1. Enable the pharmacy to keep track of the drugs prescribed to each client.
2. Enable analytic querying, such as, for example, to help prevent harmful interactions among the drugs prescribed to the clients; to help find substitute drugs; etc.

[1] Just in case you need more info, the Flora-2 manual is at `http://flora.sourceforge.net/docs/floraManual.pdf`. You might also want to take a look at the ErgoAI tutorial `https://sites.google.com/coherentknowledge.com/ergoai-tutorial/ergoai-tutorial/?authuser=0`. ErgoAI is an extension of Flora-2, but most of that tutorial is also applicable to Flora-2.

## 2 Required Data

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

- *Information about PharmDB clients, their prescriptions, and the drugs they take.*
- *Information about the drugs, including their interactions and the medical conditions they treat.*

Every client in the system has

- *Name*
- *Id*
- *Address*
- *Drugs prescribed to that person, including the medical condition the drug is intended to treat (for that particular person) and the start/end dates when each drug can be dispensed to the client. Cost is also part of the prescription.*

  For simplicity, in this project you can represent time with integers of the following form: YYYYMMDD. For instance, 20160213 for February 13, 2016. This way you can do date comparison by simply using $<$ and $>$ for integers.

- *Drugs that the client actually takes.*

  We omit the start/end dates for this data item as these dates are immaterial for the queries below. Note: a person can be taking some of the drugs prescribed to that person, but not necessarily all (one may decide to not take some of the prescribed drugs).

A drug has the following information:

- *Name*
- *Medical condition(s) that the drug treats.*
- *For each medical condition, a list of drugs that interact with the given drug in a harmful way.*

  Harmfully interacting drugs are described via triples of the form $(mc, d1, d2)$, which says that the drugs $d1$ and $d2$ cannot be taken together by a patient who has the medical condition $mc$.

  Note that harmful interactions are *not* transitively closed: $d1$ may interact harmfully with $d2$, $d2$ with $d3$, but it does not mean that $d1$ and $d3$ are incompatible drugs.

Control dates:

- Some queries refer to *control dates.*

  A sample list of control dates is provided. These dates are used in various tests in our queries and are introduced to simplify the implementation of some of the queries and to avoid the need for more advanced features of Flora-2 (such as constraint solvers, for which we have no time).

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.[2] 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.

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

## 3 Queries

You are to implement the following queries. For some queries you would need to use negation (`\naf`), aggregate operators (e.g., `sum`{... | ...}, `max{...| ...}`), quantifiers (`forall`(...)^..., `exist`(...)^...), and recursion. You might also need to use some builtins like =, ! =, >=, `\is`, @ >, @ <, etc.[3]

1. Find all client names who have a prescription for Tuberculosis that is valid on one of the control dates. You can define a predicate, *control_date*, that would list all of those control dates. Control dates are provided as part of the sample data.

2. Find all clients who have prescriptions that harmfully interact with each other *or* who have wrong prescriptions (at least one).

   The harmfully interacting prescriptions must be in effect for the respective clients on one of the control dates.

   A *wrong* prescription is one where the drug is prescribed for a condition that the drug is not intended to treat.

   The output to the first part of the query should contain, for each person's name, the pairs of harmfully interacting prescribed drugs. The output for the second part of the query should, for each person, list the wrongly prescribed drugs.

3. Find all clients (names) who have *no* harmfully interacting prescriptions (on **any** of the control dates) and the total cost of all the drugs prescribed to each of those clients is > \$200.

4. Find all clients who take all the drugs prescribed to them except those that are prescribed wrongly (see Query 2 for the definition of the wrongly prescribed drugs). This query *must* use the `forall` quantifier.

5. Find all pairs of indirectly interacting drugs.

   For the purposes of this query, a pair of drugs is interacting indirectly if there is a chain of drugs $d_1$, ..., $d_n$, such that

   - each adjacent pair $d_i, d_{i+1}$ interacts directly through some condition (i.e., there is a harmful interaction $(mc, d_i, d_{i+1})$ for some $mc$; the $mc$'s may be different for different $i$);
   - both $d_i$ and $d_{i+1}$ were prescribed to some client irrespective of the dates (the client may be different for different $i$'s);
   - $d_1$ and $d_n$ do *not* interact directly.

   This query requires recursion.

## 4 Code Repository

The source code of your project **must** be maintained in a **private** cloud code repository on *Github Classroom*, so please set up a Github account if you do not already have one. Make sure that your Github profile lists **your full name** as it is **officially known on SOLAR**.

[3] 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.

To get a repository for Project 1 in the classroom, log into your Github account (which, as explained above, must match your SBU name) and follow this URL:
https://classroom.github.com/a/UZjinDBd.
**Note:** No other repositories will be accepted.

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 – unless you are an expert. 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.

## 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 while grading.

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

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

Tables must be in 4NF (4th Normal Form). If a lower normal form is chosen, it must be well-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-documented.**

**Aesthetics.** 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.
```

**Blackboard.** 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

In the assignment submission textbox on Blackboard, **provide your Github Id** to make it easier to find your repository in the Github classroom roster.

**Format of submission.** The data, the rules, and the queries should be in **one** `.flr` file. Upload the file into the Project 1 assignment on Blackboard, 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
```

## 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:

> **Name: ....................... (as in SOLAR)**
> **SBU ID: .....................**
> **I pledge my honor that all parts of this project were done by me individually, without collaboration with anyone, and without consulting any external sources that provide full or partial solutions to a similar project.**
> **I understand that breaking this pledge will result in an "F" for the entire course.**

If in doubt—ask.

## 7 Planning Your Work

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

A test dataset is provided. **Note**: obtaining correct answers on the test data does *not* guarantee that your queries are 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). Your queries will be checked for correctness manually.

## 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/Word characters, such as space, quote, double quote, 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.