Project 1: A SQL Interpreter in Prolog

For this project, you will have to implement a SQL-style system in Prolog. If your knowledge of SQL needs refreshing, you can use an online tutorial such as this one (http://www.w3resource.com/sql/tutorials.php) . We also include a reminder about key concepts below.

Requirements

We leave you a lot of liberty in the design and the implementation, but your system needs to have at least the following capability:

- **Creating a table** by supplying the name of the table and the names of the columns. To simplify the assignment, we ignore the type of values to be stored in each column.
- **Inserting rows** by supplying the values. The column names may be supplied (we do not require it), but in any case the insert should fail if not enough values are supplied to fill the row. If you allow supplying column names, make sure that they can be supplied in any order.
- **Deleting rows** when they match some predicates.
- **Dropping a table**.
- **Updating rows** that match some predicates with new values for some of their columns.
- **Selecting rows** that match some predicates. It should be possible to select only some columns from these rows.
- **Selection should work on tables joins**. In particular, you must implement cross joins and inner joins. You can make the simplifying assumption that all columns in all tables share the same namespace. This means that two columns cannot have the same name, even if they live in different tables.
- **Predicates**: your implementation must at least support checking whether the

value for a column matches a fully-instantiated (i.e. variable-less) value, as well as ordering comparison for numeric values (`>`, `<`, ...).

We do not constrain the shape in which these commands must be expressed, but we emphasize that you have to enter them **at the prompt*. Meaning that you cannot define new predicates at that time (not even using `assertz`), and that the command should not be unwieldy to type when compared to the functionality they implement. In fact, they should not be much longer than the corresponding original SQL command.

Reminder about SQL

In SQL, you have tables. Each table has an associated set of columns, and contains a set of rows. Each rows associates a single value to each column.

In SQL, a cross join of two tables creates a new (anonymous and temporary) table whose columns are the union of the columns of the two tables. The set of rows is the cartesian product of the rows in the two tables. For instance:

```
table1: x y     table2: a b     x-join: x y a b
        1 2             6 7             1 2 6 7
        3 4             8 9             1 2 8 9
                                        3 4 6 7
                                        3 4 8 9
```

An inner join is like a cross join, but filtered to only include those rows that match some predicates, typically an equality between the value for a column from table 1 and the value for a column from table 2.

When you make a `SELECT` query in SQL, the result is a set of rows. If you use `SELECT *` those will be the regular rows from the table you are querying. But you can decide to select only some columns (e.g. `SELECT x, a FROM table1 INNER JOIN table2`) and will get rows containing only these columns.

Extra Features

While the requirements above represent the *minimum* we ask of you, you can go further (and should do so if you want to get the maximum grade). Here are a few ideas (but you can add your own instead):

- Support additional predicates. How far can you push it on the way to support arbitrary predicates (i.e. predicates that look like a query you would make on the prolog shell)?
- Support SQL aggregation functions (e.g. `SUM`, `AVG`, `COUNT`, `MIN`, `MAX`). You can use these functions in the column select part of a query (adding to each row a new column whose value is the result of the function), or in predicates.
- Enable each table to have its own namespace for columns names, so that multiple tables can use the same names for their columns.
- Create a DCG grammar that mimics the real SQL syntax and interprets the queries using your system.
- Support additional SQL features such as `ORDER BY` and `GROUP BY`. Note that `GROUP BY` requires SQL aggregation functions.

Tools

You may use any built-in predicates as well as any library bundled with SWI-Prolog (however, you should not need any).

Here are predicates which are likely to interest you:

- Dynamic predicates (manipulated using `assertz`, `retract`, ...): `apropos(database)`.
- `forall/2`, `findall/3`, `bagof/3`, `setof/3`
- (Advanced) Meta-predicates, in case you want to build predicates that work a bit like the predicates in the previous point: `apropos("meta-predicate")`.
- **!!** Predicates for analyzing and constructing terms (`apropos("Analysing and Constructing Terms")`), in particular the `=..` operator.
- Meta-call predicates: `call/N`, `apply/2`.

Deliverables

Follow these instructions to the letter, or you will LOSE POINTS.

The project deadline is **the 8th of March, 23:59**.

You must post on Moodle (section: Project 1 submission page) the three following files (and **only** those, with exactly these names):

- `solution.pl` - The full implementation of your SQL framework.
- `queries.txt` - A list of queries that demonstrates your solution. See the *Report* section to know which queries to include.
- `report.pdf` - A short report, see instructions below.

Warning: We will be intransigeant towards plagiarism. We are well aware of existing solutions, and will cross check students' submissions, both manually and automatically. You may discuss the assignment between yourself, but are not allowed to share **any** code, nor to borrow any code you found online. We will find out. We always do.

Test Data

Four CSV files have been uploaded on Moodle. These contains data (one file = one table) that you can use to test your system. You will use these tables to demonstrate the capabilities of your solution (see the *Report* section).

You solution should include a predicate that will load these tables in the system when the predicate is queried. For ease of use, do not read the data from files, simply convert it to your chosen Prolog representation and include it in your `solution.pl` file. When queried, the predicate should restore the system to a clean state (all supplied tables loaded with no modifications, any extra tables dropped).

Report

We ask you to write a short report, which should contain **only** the answers to the following questions (formatted as question then answer, not as a continuous text):

1. How are your tables/rows stored in the Prolog knowledge base?
2. As a query language, Prolog is much more powerful than SQL. What is the "mismatch" between SQL and Prolog that forces us to write code to handle the differences?

3. Include examples of Prolog queries that use your framework to perform the following tasks. Also include the result of the query (if any).

   Include only one representative example per item in this list.

   i. Load/reset the example data.
   ii. Create a new table.
   iii. Insert a row in a table.
   iv. Select all rows in a table.
   v. Select all rows that match a predicate.
   vi. Select some columns from rows that match a predicate.
   vii. Select all rows from a cross join between two tables.
   viii. Select all rows from an inner join between two tables.
   ix. Delete rows that matches a predicate.
   x. Update rows that match a predicate.
   xi. Drop a table.

   You can also give additional queries to showcase extra features of your system.

   Excepted for creating a new table, your queries should run on the example data we supplied. If your extra feature cannot be showcased using the example data, you may extend the example data (but please indicate this clearly in the report).

   Please ensure that all queries can be executed sequentially.

   All queries supplied for this question (**and only those**) must appear in your `queries.txt` file (one query per line). Do not include any other data in `queries.txt` (no results, no comments, etc). This makes it easier for us to test your solutions.