A relational database management system (RDBMS) maintains data sets called relations. A relation has a name, a schema, and a set of tuples. A schema is a set of attributes. A tuple is a set of attribute/value pairs. An attribute is the name associated with each data value in a tuple entry. Relations are used as operands in relational operations such as rename, selection, projection, union, and join. The goal of this project is to build a relational database system from a Datalog file, and then answer queries using relational algebra.

A Datalog Program can be represented by a Database. Each scheme in a Datalog Program defines a relation in the Database. The scheme defines the name of the relation, and the attribute list of the scheme defines the schema of the relation. Each fact in the Datalog Program defines a tuple in a relation. The fact name identifies a relation to which the tuple belongs. The basic data structure is a database consisting of relations, each with their own name, schema, and set of tuples. How you choose and use your data structures effects the difficulty of the project. Inheritance and polymorphism are always useful. Each data structure implementation has its benefits, drawbacks and obstacles to overcome.

Your program must convert every Datalog query into a sequence of select, project, and rename operations. As you prepare to write your program, take out a piece of paper, convert each of the queries in the given examples into a relational expression, and carry out these expressions. Your program should create these same expressions.

Project Description

Write an RDBMS-based interpreter for a Datalog Program. Given a Datalog Program, your interpreter should print the result for each query using only the facts (interpretation of the rules will be added in Project 4).

To format the output for pass-off, you must make it look like the examples below. Note the following: The queries are to be evaluated in the order listed in the Datalog Program. For each query, print the query and a space. Then, if the query's resulting relation is empty, output “No”; and if the resulting relation is not empty, output “Yes(n)” where n is the number of tuples in the resulting relation. If there are free variables in the query, print the tuples of the resulting relation, one per line and indented by two spaces, according to the following directions. Each tuple should be a comma-space separated list of pairs-(variable, string-value)-ordered according to the order of variable appearance in the query. The format of a pair is *v*= 's', where *v* is the variable and 's' is the string value. Sort the tuples alphabetically based on string values in the tuples' variables (sort with the first sort key as the first variable (in order of appearance in the query), the second sort key as the second variable, and so on).

The focus of this project is on interpreting 236-Datalog (without rules)-not on building an industrial-strength interpreter. Thus, to retain the central focus and keep the amount of coding within a reasonable bound, you may assume:

- The arity (number of parameters) of all schemes, facts, rules and queries that have the same name are guaranteed to match. You do not have to check for matching arity.
- No two attributes in the same scheme will have the same name. You do not have to check for this.
- No two schemes in the scheme list will have the same name. You do not have to check for this.
- Every scheme (or relation) referenced in a fact, rule head, predicate, or query will have been defined in the scheme section.
- Attribute names in schemes and variable names in queries have distinct name spaces: no attribute name will appear as a variable in a query.

Requirements (Get the Design Right)

Your design should include classes for the database and relation. Queries must be implemented by performing the appropriate relational operations (select, project, and rename). The implementation of the relational operators must be methods, and they must follow the mathematical definitions described in class and in the textbook. For example, the operators must not produce relations which have duplicate tuples or duplicate names in the schema.

Your project will not be accepted if it does not implement a relation object with the relational algebra operators as methods that are used to answer queries.

Examples

These are not sufficient to completely test your program. Your program must have output formatted exactly like the example outputs below. Example Input

Input

```
Schemes:
  SK(A,B)
Facts:
  SK('a','c').
  SK('b','c').
  SK('b','b').
  SK('b','c').
Rules:
  DoNothing(Z) :- Stuff(Z).
Queries:
  SK(A,'c')?
  SK('b','c')?
  SK(X,X)?
  SK(A,B)?
```

Output

```
SK(A,'c')? Yes(2)
  A='a'
  A='b'
SK('b','c')? Yes(1)
SK(X,X)? Yes(1)
  X='b'
SK(A,B)? Yes(3)
  A='a', B='c'
  A='b', B='b'
  A='b', B='c'
```

Input

```
Schemes:
        ab(A,B)
Facts:
        ab('joe','bob').
        ab('jim','bob').
        ab('joe','jim').
        ab('bob','bob').
Rules:

Queries:
        ab('joe','jim')?
        ab( who, 'bob')?
        ab('joe', anyone)?
        ab(X,X)?
        ab(X,Y)?
```

Output

```
ab('joe','jim')? Yes(1)
ab(who,'bob')? Yes(3)
  who='bob'
  who='jim'
  who='joe'
ab('joe',anyone)? Yes(2)
  anyone='bob'
  anyone='jim'
ab(X,X)? Yes(1)
  X='bob'
ab(X,Y)? Yes(4)
  X='bob', Y='bob'
  X='jim', Y='bob'
  X='joe', Y='bob'
  X='joe', Y='jim'
```

Input

```
Schemes:
        dc(D,C)
Facts:
        dc('ralph','howard').
Rules:

Queries:
        dc('ralph', X)?
        dc('bob','bob')?
        dc(X,Y)?
```

Output

```
dc('ralph',X)? Yes(1)
  X='howard'
dc('bob','bob')? No
dc(X,Y)? Yes(1)
  X='ralph', Y='howard'
```

In the next example, notice that the rules are not evaluated, but the queries are still answered using only the facts included in the file.

Input

```
Schemes:
        people(person1,person2)
        employer(boss,employee)
Facts:
        people('joe','bob').
        people('jim','bob').
        people('joe','jim').
        employer('ralph','howard').
        people('bob','bob').
Rules:
        employer(X,Y):- people(Y,X).
        employer(X,Y):- people(X,Z),employer(Z,Y).
        people(X,Y):- people(Y,X).
Queries:
        people('joe','jim')?
        people( who, 'bob')?
        people('joe', anyone)?
        people(X,X)?
        people(X,Y)?
        employer('ralph', X)?
        employer('bob','bob')?
        employer(X,Y)?
```

Output

```
people('joe','jim')? Yes(1)
people(who,'bob')? Yes(3)
  who='bob'
  who='jim'
  who='joe'
people('joe',anyone)? Yes(2)
  anyone='bob'
  anyone='jim'
people(X,X)? Yes(1)
  X='bob'
people(X,Y)? Yes(4)
  X='bob', Y='bob'
  X='jim', Y='bob'
  X='joe', Y='bob'
  X='joe', Y='jim'
employer('ralph',X)? Yes(1)
  X='howard'
employer('bob','bob')? No
employer(X,Y)? Yes(1)
  X='ralph', Y='howard'
```

FAQ

When should I select, project, or rename in a query?

Here is a simple strategy to consider when deciding how to evaluate queries.

1. Only do select operations as you traverse the parameters. The select will either be on a constant or it will be on an ID if the ID has been seen previously (i.e., the special case to avoid having two identical IDs in the scheme.
2. Once all the select operations have been completed, then rename and project.

The strategy implies that each select knows the list of IDs in the final project since it needs to check for duplicates to determine when to select on a column rather than a constant. Also in this strategy, it may be the case that no select takes place on an ID parameter (which is just fine).

Should project also re-order the tuple?

The output for the lab requires that tuples follow the order declared in the query. Although not apparent right now, consider what happens in a natural join? In such an operation, the order becomes less clear. A project-method can ensure the tuples follow the correct order as part of its operation with some little thought and foresight.

Submission

Review the project standards for creating a zip archive.

Navigate to Learning Suite [http://learningsuite.byu.edu], select 'CS 236', and click on 'Assignments' on the course home page. Click on the link to the relevant project and at the bottom of the description click on 'View/Submit'. Use the resulting upload dialog to upload your zip archive.

Pass-off

Pass-off your project directly to a TA during normal TA hours. TAs help students on a first-come/first-serve basis and are under no obligation to stay later than designated hours so plan accordingly. Please review the syllabus for the pass-off requirements.

cs-236/relational-database.txt · Last modified: 2015/02/27 22:22 by egm