Printer-Friendly Version (PDF)

The RDBMS

CS236 Project Four

Relational Databases

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* (too-pul) 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*.

Project Description

Write an RDBMS-based interpreter for a DatalogProgram. The grammar below is simpler than the grammar we used for Project 2. You shouldn't need to modify how your Project 2 gramatically works. Given a DatalogProgram, your interpreter should print the result for each query. The queries should be evaluated in the order given from the file. First, print the query and a space. If the query's resulting relation is empty, print "No". If the resulting relation is not empty, output "Yes(x)" where x is the number of Tuples. Then, if there are free variables in the query, print the tuples of the resulting relation alphabetically on separate lines (indent them by 2 spaces). Each line or tuple should display the free variables from the query (in the same order as the query), and, for each variable, give the corresponding value from the tuple. The Tuple's data is comma space separated.

Notes

A DatalogProgram can be represented by a Database. Each *scheme* in a DatalogProgram defines a Relation in the Database. The *scheme name* defines the name of the relation and the *attribute list* of the scheme defines the schema of the relation. Each *fact* in the DatalogProgram defines a Tuple in a Relation. The *fact name* identifies a Relation to which the Tuple belongs. The basic data structure for Project 4 (and Project 5) 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 Project 4 (and Project 5). Inheritance and Polymorphism are always useful. Each data structure implementation has its benefits, drawbacks and obstacles to overcome.

Since we ignore Rules for Project 4, you need only worry about converting Datalog Queries into Database queries. One way to do this is to convert every Datalog query into a sequence of rename, select, and project operations. The *predicate name* of the query identifies the relation we are to operate on. First, we make a copy of the relation (we don't want to change the original). (Advice: write a copy constructor for Relation.) Then we rename the schema of the new relation so that they have the corresponding names found in the query. You may have to violate the rule that no two attributes can have the same name. Don't worry about this. We will fix it when we perform the projection. After renaming, use the select operation to select tuples from the resulting relation that match the Datalog query. A tuple, *t*, from resulting relation *R*, matches the Datalog query if

1) for every constant *c* in the query corresponding to attribute *A* of *R*, value *v* in the tuple *t* corresponding to attribute *A* of *R* is equal to *c*; and

2) for every variable var_1 in the query corresponding to attribute *A* of *R*, if there exists a variable var_2 in the query corresponding to attribute *B* of *R* and $A \neq B$ and $var_1 = var_2$ then the constant c_1 of *t* corresponding to attribute *A* equals the constant c_2 of *t* corresponding to attribute *B*.

After selecting the appropriate tuples, we perform a projection by keeping those columns from the resulting relation that corresponds to variables in the query. If a variable appears more than once, we keep only one of the columns with that variable. We then print each of the resulting tuples. The variables of the tuples should be printed in the same order as the variables in the query (as read from left to right). The tuples' order should be alphabetical (by first variable's value and if need be up to nth variable's value). It shouldn't be hard to sort them. (Advice: make Tuple implement Comparable and write a compareTo method. Then dump the tuples into a TreeSet which will remove duplicates and sort for you.) Variables' column order should be the same as the query.

Notes About Pass-off Files

The focus of Project 4 is on interpreting 236-Datalog (without rules), and not on building an industrial-strength interpreter. Thus, the following guidelines have been adopted in creating passoff files:

- 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. You do not have to check for this.

New Grammar

<Datalog Program>	->	Schemes : <Scheme List> Facts : <Fact List> Rules : <Rule List> Queries : <Query List> <EOF>
<Scheme List>	->	<Scheme><Scheme List Tail>
<Scheme List Tail>	->	<Scheme List> | ε
<Scheme>	->	<Scheme Name> (<Attribute List>)
<Scheme Name>	->	<Identifier>
<Attribute List>	->	<Attribute> <Attribute List Tail>
<Attribute List Tail>	->	, <Attribute List> | ε
<Attribute>	->	<Identifier>
<Fact List>	->	<Fact> <Fact List> | ε
<Fact>	->	<Fact Name> (<Constant List>) .
<Fact Name>	->	<Identifier>
<Constant List>	->	<String> <Constant List Tail>
<Constant List Tail>	->	, <Constant List> | ε
<Rule List>	->	<Rule> <Rule List> | ε
<Rule>	->	<Head> :- <Predicate List> .
<Head>	->	<Scheme>
<Predicate List>	->	<Predicate> <Predicate List Tail>
<Predicate List Tail>	->	, <Predicate List> | ε
<Predicate>	->	<Predicate Name> (<Argument List>)
<Predicate Name>	->	<Identifier>
<Argument List>	->	<Argument> <Argument List Tail>
<Argument List Tail>	->	, <Argument List> | ε
<Argument>	->	<String> | <Identifier>
<Query List>	->	<Query> <Query List Tail>
<Query List Tail>	->	<Query List> | ε
<Query>	->	<Predicate> ?

Requirements (Get the Design Right)

Your design must include the following classes: *Database*, *Relation*, *Schema*, and *Tuple*. In addition, the implementation of the relational operations (*rename*, *select*, and *project*) must be methods in the *Relation* class. It may be beneficial to include a *print* or *toString* method in the *Relation* class.

Examples

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

Example 40 Input: ex40.txt

```
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)?
```

Example 40 Output: out40.txt

```
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'
```

Example 41 Input: ex41.txt

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

Example 41 Output: out41.txt

```
S(A,'a',B)? Yes(1)
  A='a', B='a'
```

Example 42 Input: ex42.txt **Example 42 Output: out42.txt**

```
Schemes:
   snap(S,N,A,P)
   csg(C,S,G)
   cdh(C,D,H)
   cr(C,R)
Facts:
   snap('12345','C. Brown','12 Apple','555-1234').
   snap('67890','L. Van Pelt','34 Pear','555-5678').
   snap('22222','P. Patty','56 Grape','555-9999').
   snap('33333','Snoopy','12 Apple','555-1234').
   csg('CS101','12345','A').
   csg('CS101','67890','B').
   csg('CS101','33333','A-').
   csg('EE200','12345','C').
   csg('EE200','22222','B+').
   csg('EE200','33333','B').
   csg('PH100','67890','C+').
   cdh('CS101','F','9PM').
   cdh('EE200','M','10AM').
   cdh('EE200','W','10AM').
   cdh('PH100','Tu','11AM').
   cr('CS101','Auditorium').
   cr('EE200','25 Ohm Hall').
   cr('PH100','Newton Lab').
Rules:
   DoNothing(Z) :- Stuff(Z).
Queries:
   snap(S,'Snoopy',A,P)?
   csg(Course,'33333',Grade)?
   cr('CS101','Auditorium')?
   cdh('EE200','F',Hour)?
   csg(Course,Id,Grade)?
   cdh(Course,Day,Hour)?
   cr(Course,Room)?
```

```
snap(S,'Snoopy',A,P)? Yes(1)
  S='33333', A='12 Apple', P='555-1234'
csg(Course,'33333',Grade)? Yes(2)
  Course='CS101', Grade='A-'
  Course='EE200', Grade='B'
cr('CS101','Auditorium')? Yes(1)
cdh('EE200','F',Hour)? No
csg(Course,Id,Grade)? Yes(7)
  Course='CS101', Id='12345', Grade='A'
  Course='CS101', Id='33333', Grade='A-'
  Course='CS101', Id='67890', Grade='B'
  Course='EE200', Id='12345', Grade='C'
  Course='EE200', Id='22222', Grade='B+'
  Course='EE200', Id='33333', Grade='B'
  Course='PH100', Id='67890', Grade='C+'
cdh(Course,Day,Hour)? Yes(4)
  Course='CS101', Day='F', Hour='9PM'
  Course='EE200', Day='M', Hour='10AM'
  Course='EE200', Day='W', Hour='10AM'
  Course='PH100', Day='Tu', Hour='11AM'
cr(Course,Room)? Yes(3)
  Course='CS101', Room='Auditorium'
  Course='EE200', Room='25 Ohm Hall'
  Course='PH100', Room='Newton Lab'
```

Passoff Requirements

Build your project using the cs236Bundle. Your source code goes in the src/project4 directory. All classes must be package project4 and import project1, and project2. You must implement the body method in Project4 class. Its only task is to construct a Database and return an output string. This class receives a filename from the command line arguments. Other requirements for all cs236 projects can be found on the policies page.