Report for Yadi–Work Package 2

Brian AMPWERA, Yandi LI, Mulu Weldegebreal ADHANA

January 28, 2013

1 Introduction

Yet Another Datalog Interpreter (YADI) is a Datalog-to-SQL converter that transforms a Datalog query to an Abstract Syntax Tree to an SQL statement to an answer set of tuples.

2 Using Yadi

2.1 Starting Yadi

In order to start the yadi line interpreter, the user should type the executeable type at the prompt with the connection information.

```
./yadi "host=XXXX port=XXXX user=XXXX password=XXXX dbname=XXXX"
./yadi "host=localhost port=5432 user=postgres password=1 dbname=mydb"
```

At the command line

```
yadi$
```

2.2 List Existing Relations

To view the exisiting relations: \./

Example

```
yadi$ \./
table_name|
----------+
 q        |
 help     |
 movies   |
 actors   |
```

List the Columns of a Relation: \{name-of-relation}./

Example

```
yadi$ \Movies./
 count   |
---------+
 2       |
```

2.3 Getting help

To view the help incorporated in the application

```
?./
```

the command will provide a detail list of help commands and their options

```
?general./
?query./
?syntax./
?ddl./
?dbinfo./
?agg./
?anonymous./
?symbol./
?keyword./
?insert./
?delete./
?drop./
```

2.4 Exit

```
yadi$ exit./
```

3 Query

3.1 Syntax

A query consists of several rules that define some idbs/edbs[1], followed by a query of one of the idb/edb. The syntax should obey the following specification. A predicate name is formatted as an uppercase letter followed by a series of lowercase letters; a variable is a series of lowercase letters; an interger is automatically recognized as contant value; strings are those enclosed in double quotes; `'_'` stands for an anonymous variable; negation is represented as a tilde before a predicate; and operators for comparison can be recognized.

Here is a example as demonstration,

[1] intensional database/extensional database

```
Legal predicate: A(_,_), Predicate(_,_)
Illegal predicate: PAR(_,_), A1(_,_)
Legal variable: variable, x
Illegal variable:  goHome, x1
Integer constant: 12
String constant: "hello", "yadi-8", "12"
Anonymous variable: _
Negation: ~A(_,_)
Comparison operator: >, <, =, <>, >=, <=
```

In the rule that defines an idb, there can be several predicates in the body, seperated either by `','` or by `and`, interpreted as a conjunction of predicates. Each sentence/rule ends with a dot `'.'`. Multiple lines of rules together with a query lauched by symbol `'?-'` are combined to be a query program. Here gives an example of a query program,

```
Q(x):-Q(y) and Schedule(y,x).
Q(x):-Schedule(1,x), ~x=2.
?-Q(x)./
```

One could note that, YADI takes the symbol `':-'` as a seperator of the head and the body of a rule; and it takes a slash `'/'` to commit the program.

3.2 Expressive power

3.2.1 Conjunctive query

Join/Intersection/And/Cross-product A join relation is implicitly called in the following example,

```
Q(x,y):- R(x,z), S(z,y).
?-Q()./
```

where we want to join R and S on their common variables.

The SQL result of the example above is

```
WITH RECURSIVE
Q("1","2") AS
((SELECT R1."1", S1."2"
FROM S as S1 , R as R1
WHERE S1."1"=R1."2"))
SELECT * FROM Q
```

As shown above, natural numbers are used to name the columns for all the underlying tables, which is a must as an edb for YADI. This makes sense because columns have no name in Datalog. Also note that, tables are renamed with a postfix of a natural number . This strategy takes effect especially when a predicate shows several times in a rule.

Comparison One can make comparison of a variable to a string, an integer, or another variable. For instance,

```
Q(x,y,z):-R(x,x,m,y) and S(t,_,_,m) and x<>"hello the WORLD!" and t<m and T(z) and 3>z.
?-Q(x,y,z)./
```

will be translated into

```
WITH RECURSIVE
Q("1","2","3") AS
((SELECT R1."1", R1."4", T1."1"
FROM T as T1 , S as S1 , R as R1
WHERE R1."1"=R1."2" AND S1."4"=R1."3" AND 3>T1."1" AND S1."1"<S1."4" AND R1."1"<>'hello the WORLD
    !'))
SELECT "1","2","3" FROM Q
```

Values as predicate argument and anonymous variables These features make the program more like a Datalog. Equality constraints can be directly written as an arguments of a predicate. Anonymous variables can be used when the value of the variable doesn't matter. For example,

```
Q(x,y):-R("hello",x,y,1) and S("1",_,_,y) and x<>"hello the WORLD!".
?-Q()./
```

```
WITH RECURSIVE
Q("1","2") AS
((SELECT R1."2", S1."4"
FROM S as S1 , R as R1
WHERE S1."1"='1' AND R1."1"='hello' AND R1."4"=1 AND S1."4"=R1."3" AND R1."2"<>'hello the WORLD!'
    ))
SELECT * FROM Q
```

Aggregate functions Aggregate functions have their meanings when appearing in the head of the rule. Three aggregate functions are supported, namely `sum(),count(),avg()`. As shown in the second column below, the aggregation can be interpreted as, for each value x, get the sum of values in y column corresponding to the same x. The same goes to the third column, where we want to know the number of occurence for each value of x. In geneal, aggregration is: for each value of the variable outside the aggregate function, do the calculation.

```
?-Schedule()./
SELECT * FROM Schedule
 1       | 2       |
---------+---------+
 1       | 2       |
 1       | 3       |
 2       | 4       |
 3       | 4       |
 4       | 5       |
 4       | 6       |
 4       | 7       |
 6       | 7       |
 7       | 8       |
 7       | 9       |
 10      | 11      |
 12      | 13      |
```

```
Q(x,sum(y)):-Schedule(x,y).
?-Q()./

WITH RECURSIVE
Q("1","2") AS
((SELECT Schedule1."1",
  SUM(Schedule1."2")
FROM Schedule as Schedule1
GROUP BY Schedule1."1"
ORDER BY Schedule1."1"))
SELECT * FROM Q

 1       | 2       |
---------+---------+
 1       | 5       |
 2       | 4       |
 3       | 4       |
```

```
Q(x,count(x)):-Schedule(x,_).
?-Q()./

WITH RECURSIVE
Q("1","2") AS
((SELECT Schedule1."1",
  COUNT(Schedule1."1")
FROM Schedule as Schedule1
GROUP BY Schedule1."1"
ORDER BY Schedule1."1"))
SELECT * FROM Q

 1       | 2       |
---------+---------+
 1       | 2       |
 2       | 1       |
 3       | 1       |
```

```
|  4  | 18  |        |  4  | 3  |
|  6  | 7   |        |  6  | 1  |
|  7  | 17  |        |  7  | 2  |
|  10 | 11  |        |  10 | 1  |
|  12 | 13  |        |  12 | 1  |
```

Aggregate functions are compatible with all first-order-logic query, but not with recursive query, excluded from the SQL standard.

Constants in the head One can put a constant in the head of the rule, to concatenate a column of constant value into the query. As an example, where each value of x in R is paired with an 1 and then selected,

```
Q(x,1):-R(x,_).
?-Q()./

WITH RECURSIVE
Q("1","2") AS
((SELECT R1."1", 1
FROM R as R1))
SELECT * FROM Q
```

```
 1  | 2  |
----+----+
 1  | 1  |
 2  | 1  |
 3  | 1  |
 4  | 1  |
 6  | 1  |
 8  | 1  |
 10 | 1  |
 12 | 1  |
 13 | 1  |
```

```
?-R()./
SELECT * FROM R
 1  | 2  |
----+----+
 1  | 2  |
 2  | 3  |
 3  | 4  |
 4  | 5  |
 6  | 7  |
 8  | 9  |
 10 | 11 |
 12 | 13 |
 13 | 14 |
```

3.2.2 First-order logic query

Union/Or In Datalog syntax, one may express the logic concept "or" by putting several lines of rules with the same idb as heads. Thus, any tuple that satisfies any of the rules will be selected. For instance,

```
Q(x,y):-R(x,m) and S(y,m).
P(x):-Q(x,_) and T(x).
P(y):-R(y,_) and Q(y,z).
?-P()./
```

```
WITH RECURSIVE
Q("1","2") AS
((SELECT R1."1", S1."1"
FROM S as S1 , R as R1
WHERE S1."2"=R1."2")) ,
P("1") AS
((SELECT Q1."1"
FROM Q as Q1 , R as R1
WHERE Q1."1"=R1."1")
UNION
(SELECT T1."1"
FROM T as T1 , Q as Q1
WHERE T1."1"=Q1."1"))
SELECT * FROM P
```

Negation w/o recursion One can query for tuples that are not belonging to a certain predicate. In this case, negation of predicate will be a handy function. Note that in order for the query to be safe, each variables in the head of the rule must occur in at least once in a positive predicate. Cases of such safety exceptions will be given later. An example of negation together with union is,

```
Q(x,y):- R(x,z), T(y), ~S(y,z), ~z=3.
Q(x,1):- R(x,1).
?-Q(x,y)./
```

```
WITH RECURSIVE
Q("1","2") AS
((SELECT R1."1", 1
FROM R as R1
WHERE R1."2"=1)
UNION
((SELECT R1."1", T1."1"
FROM T as T1 , R as R1)
EXCEPT
(SELECT R1."1", T1."1"
FROM T as T1 , R as R1 , S as S1
WHERE T1."1"=S1."1" AND R1."2"=S1."2" AND R1."2"=3)))
SELECT "1","2" FROM Q
```

3.2.3 Recursive query without negation

Due to the limitation of postgresSQL system, only linear and non-mutual recursion can be implemented. "Linear" means, in each rule an idb can call itself at most once. "Non-mutual" means, it is forbidden to have two idbs each of who call the other idb, which forms a loop. The next example gives a demonstration,

```
Q(x):-Schedule(2,x).
Q(x):-Q(y), Schedule(y,x).
?-Q(x)./
```

```
WITH RECURSIVE
Q("1") AS
((SELECT Schedule1."2"
FROM Schedule as Schedule1
WHERE Schedule1."1"=2)
UNION
(SELECT Schedule1."2"
FROM Schedule as Schedule1 , Q as Q1
WHERE Schedule1."1"=Q1."1"))
SELECT "1" FROM Q
```

```
| 1       |
|---------+
| 4       |
| 5       |
| 6       |
| 7       |
| 8       |
| 9       |
|         |
|         |
```

```
| ?-Schedule()./
| SELECT * FROM Schedule
|  1       | 2       |
| ---------+---------+
|  1       | 2       |
|  1       | 3       |
|  2       | 4       |
|  3       | 4       |
|  4       | 5       |
|  4       | 6       |
|  4       | 7       |
|  6       | 7       |
|  7       | 8       |
|  7       | 9       |
|  10      | 11      |
|  12      | 13      |
```

3.2.4 Stratified negation

Stratified recursion with negation is also supported in YADI, which means no predicate will call itself negatively. Note that now one can do linear, non-mutual, stratisfiable, recursion with negation. The next example illustrates this feature.

```
Q(x):-Q(y), Schedule(y,x).
Q(x):-Schedule(1,x) and ~x=2.
?-Q(x)./
```

```
WITH RECURSIVE                                  | 1       |              | ?-Schedule()./
Q("1") AS                                       |---------+              | SELECT * FROM Schedule
(((SELECT Schedule1."2"                         | 3       |              |  1        | 2        |
FROM Schedule as Schedule1                      | 4       |              | ---------+---------+
WHERE Schedule1."1"=1)                          | 5       |              |  1        | 2        |
EXCEPT                                          | 6       |              |  1        | 3        |
(SELECT Schedule1."2"                           | 7       |              |  2        | 4        |
FROM Schedule as Schedule1                      | 8       |              |  3        | 4        |
WHERE Schedule1."1"=1 AND Schedule1."2"=2))     | 9       |              |  4        | 5        |
UNION                                           |                        |  4        | 6        |
(SELECT Schedule1."2"                           |                        |  4        | 7        |
FROM Schedule as Schedule1 , Q as Q1            |                        |  6        | 7        |
WHERE Schedule1."1"=Q1."1"))                    |                        |  7        | 8        |
SELECT "1" FROM Q                               |                        |  7        | 9        |
                                                                         |  10       | 11       |
                                                                         |  12       | 13       |
```

3.3 Special cases & Exceptions

3.3.1 Special cases

A query with a empty list of argument will select all tuples of the relation. Edbs can be queried without rules.

```
Q(x):-R(x,y).      |  ?-R()./
?-Q()./            |
                   |  SELECT * FROM R
WITH RECURSIVE     |
Q("1") AS          |
((SELECT R1."1"    |
FROM R as R1))     |
SELECT * FROM Q    |
```

3.3.2 Exceptions handling

```
Q(x,y):- S(x,_,_,_).
?-Q()./
Failure("Unbounded variable in the head")
-------------------------------------------------------------------------------------------
?-Q()./                          (*Q is idb*)
```

```
Fatal error: ERROR: relation "q" does not exist
LINE 3: SELECT * FROM Q
-----------------------------------------------------------------------------------------------
Q(x):- ~S(x,_,_,_).
?-Q()./
Failure("Rule with empty body: Unsafe query")
-----------------------------------------------------------------------------------------------
Q(x,y):-Schedule(x,y).
Q(x,y):-Q(x,z),Q(z,y).
T(x):-Q(2,x).
?-T()./
Fatal error: ERROR: recursive reference to query "q" must not appear more than once
LINE 8: FROM Q as Q2 , Q as Q1
-----------------------------------------------------------------------------------------------
Q(x):-R(x),T(x).
R(x):-Q(x), S(x).
?-Q()./
Failure("The query is a mutual recursion, which is not supported in postgres SQL")
-----------------------------------------------------------------------------------------------
Q(x):- R(x).
R(x):- ~Q(x).
?-Q()./
Failure("The query is a mutual recursion, which is not supported in postgres SQL")
-----------------------------------------------------------------------------------------------
Q(x,y):- R(x), ~Q(1,x).
Q(x,y):-S(x,y).
?-Q()./
Failure("The query is not stratisfiable.")
```

4 Other Data Manipulation Language

4.1 Insert

Format of the command: +{NameOfRelation}({Comma-separated-valuesOfAtrributes})./

Multiple tuples can be inserted collectively.

```
+R(1,2).+R(2,3).+R(3,4).
+Parent(Kate, Bob).+Parent(Nicolas,Bob).      (*a series of letters will
+Parent(a,b).+Parent(c,d).                    be recognized as string constant*)
+Movies("Reservoir Dogs","Q. Tarantino").
+Movies("Death Proof","Q. Tarantino").
+Movies("Kill Bill 1 ","Q. Tarantino").
+Actors("Jamie Foxx", "Django").
+Actors("Kerry Washington ", "Django")./
```

Inserting from a file: <<{name_of_file}

```
<< "myActors" ./
```

Note. The implementation of the above mentioned insertion is from a datalog perspective, When the interpreter recevies a fact, an attempt is made to insert the fact into the an existing relation. However, in the event that the table doesn't exist then a table is created.

Limitations. The maximum size of the fact in each tuple is 15 characters. The current version only supports variables of the type string and int.

4.2 Delete

Format of the command: -NameOfRelation({comma-separated-valuesofAttributes}).

```
-S(45,67).
-Parent(Mia,Bob).
-Movies("Reservoir Dogs","Q. Tarantino").
-Movies("Death Proof","Q. Tarantino").
-Movies("Kill Bill 1 ","Q. Tarantino").
-Actors("Jamie Foxx", "Django").
-Actors("Kerry Washington ", "Django").
```

4.3 Update

This version does not include a direct implementation of the update query as in sql. However, an update can be performed by deleting the exact tuple and re-inserting into the table.

5 Data Definition Language

5.1 Create

In order to create a table in datalog, would require to insert a tuple into the database relation, therefore this is acheived by inserting a tuple.

5.2 Drop

Format of the command: !{Name-of-Relation}.

```
!Actors.
```

6 Live example

An example that combines all Data Manipulation Language and Data Definition Language is given as follows,

```
Enter for Help: ?.
yadi$ \.
table_name|
---------+
 r       |
```

```
 help    |
 loves   |
 actors  |

yadi$ \Loves./
 count   |
---------+
 2       |

yadi$ ?-Loves()./
 1       | 2       |
---------+---------+
 Bob     | Jane    |
 James   | Gates   |
 James   | Jane    |

yadi$ << "test"./
Reading from test
!Loves.
+Loves(Kate,James).+Loves(Bob, Jane).+Loves(James,Jane).+Loves(Jane,Gates).
+Loves(Benjamin,Kate).+Loves(Mike,Jane).+Loves(Benjamin,James).+Loves(Kate,Kate).
Jealous(x,y):- Loves(x,z), Loves(y,z), x<>y.
?-Jealous()./

 1       | 2       |
---------+---------+
 Benjamin| Kate    |
 Mike    | Bob     |
 James   | Bob     |
 Mike    | James   |
 Bob     | James   |
 Kate    | Benjamin|
 James   | Mike    |
 Bob     | Mike    |
 Kate    | Benjamin|
 Benjamin| Kate    |

yadi$ ?-Loves()./
 1       | 2       |
---------+---------+
 Kate    | James   |
 Bob     | Jane    |
 James   | Jane    |
 Jane    | Gates   |
 Benjamin| Kate    |
 Mike    | Jane    |
 Benjamin| James   |
 Kate    | Kate    |
```