MIF14-BDD: Project
Implementing and Testing a Stratified Datalog Engine
Part II: Translating Datalog → SQL and Testing under Oracle

Abstract

This project will let you delve into the details of the implementing and evaluating a stratified Datalog engine. The project consists of two main parts.
The first part (about 12 points) consists of **implementing** an engine that computes all the facts that can be infered from a stratified Datalog program, given as input.
The second part (about 8 points) consists of **testing** the performance of your engine. To this end, you should check the correctness of its output by comparing it with that obtained by **either**:

- (about 2 points) *manually* translating your program into DES **and** SQL and evaluating using both DES **and** Oracle
- (about 8 points) writing an *automatic* translation of your program to SQL with recursion and evaluating using Oracle

This project is to be performed by one or two students. The programming language is `Java`. A small report (at most 2 pages, without appendix) must be added to your project.

The project is due on May 21, 2017 at 11pm.
You have to submit it on Spirals by this deadline (no email is accepted).
In order to submit, you have to go here:
http://spiralconnect.univ-lyon1.fr/webapp/activities/activities.jsp?containerId=6689641

1 Translating from Datalog to SQL

Every *base* predicate (EDB) in your Datalog program corresponds to a table (relation) name in SQL. As such, for each Datalog predicate, you should give an SQL table creation statement, such that:

- the number of attributes (column names) equals the computed predicate arity
- the type of all attributes is `VARCHAR(150)`

```
link(Charpennes,Perrache)
link(PartDieu,Charpennes)
link(Debourg,PartDieu)
link(PartDieu,Debourg)
link(X,Y)→metro(X).
link(X,Y)→metro(Y).
link(X,Y)→reachable(X,Y).
link(X,Z),reachable(Z,Y)→reachable(X,Y).
```

```
CREATE TABLE link (
   link.1 VARCHAR(150),
   link.2 VARCHAR(150)
);
```

Next, Datalog *facts* are translated as SQL insertion statements, as illustrated next:

```
INSERT INTO link VALUES (Charpennes,Perrache);
INSERT INTO link VALUES (PartDieu,Charpennes);
INSERT INTO link VALUES (Debourg,PartDieu);
INSERT INTO link VALUES (PartDieu,Debourg);
```

1) Non-recursive Datalog rules capture the expressivity of SQL "SELECT-FROM-WHERE" queries. The notion of a *defined* predicate (IDB) corresponds to that of a SQL *view* SQL, whose syntax is [1]:

```
CREATE VIEW vname AS
SELECT table_1.1,table_2.2,...,table_n.k
FROM   table_1,table_2,...,table_n
WHERE  [conditions]
```

The view can then be queried as: `SELECT * FROM vname`.

To exemplify, consider the following ways to define the *unreachable* predicate in Datalog.

Example 1.1. The below non-recursive and positive Datalog rule translates to SQL as:

```
metro(Charpennes)
metro(Perrache)
metro(PartDieu)
metro(Debourg)
creachable(Charpennes,Charpennes)
creachable(Charpennes,PartDieu)
creachable(Charpennes,Debourg)
creachable(Perrache,Charpennes)
creachable(Perrache,PartDieu)
creachable(Perrache,Perrache)
creachable(Perrache,Debourg)
metro(X),metro(Y),creachable(X,Y)→unreachable(X,Y).
```

```
CREATE or REPLACE VIEW unreachable AS
SELECT creachable.1, creachable.2
FROM metro as m1, metro as m2, creachable
WHERE creachable.1 = m1.1
AND   creachable.2 = m2.2
```

Example 1.2. The below non-recursive Datalog rule with (stratified) negation translates to SQL as:

```
metro(Charpennes)
metro(Perrache)
metro(PartDieu)
metro(Debourg)
reachable(Charpennes,Perrache)
reachable(PartDieu,Charpennes)
reachable(Debourg,PartDieu)
reachable(PartDieu,Debourg)
reachable(PartDieu,Perrache)
reachable(Debourg,Charpennes)
reachable(Debourg,Debourg)
reachable(PartDieu,PartDieu)
reachable(Debourg,Perrache)
metro(X),metro(Y),NEG reachable(X,Y)→unreachable(X,Y).
```

```
CREATE or REPLACE VIEW unreachable AS
SELECT reachable.1, reachable.2
FROM metro as m1, metro as m2, reachable
WHERE reachable.1 = m1.1
AND   reachable.2 <> m2.2
OR (reachable.1 <> m1.1
    AND reachable.2 = m2.2)
```

[1] We can, of course, select values corresponding to any position of a given table, in any given order; the syntax is simplified for clarity purposes.

2) Recursive Datalog rules can be translated via SQL queries WITH...RECURSIVE.

To exemplify, consider the following definition for the *reachable* predicate in Datalog:

Example 1.3. The below recursive and positive Datalog rule translates to SQL as:

```
link(Charpennes,Perrache)
link(PartDieu,Charpennes)
link(Debourg,PartDieu)
link(PartDieu,Debourg)
link(X,Y)→metro(X).
link(X,Y)→metro(Y).
link(X,Y)→reachable(X,Y).
link(X,Z),reachable(Z,Y)→reachable(X,Y).
```

```
CREATE or REPLACE VIEW reachable AS
WITH RECURSIVE rec_reachable AS (
  SELECT link.1, link.2 FROM link
  UNION ALL
   (SELECT link.1, rec_reachable.2
    FROM link, rec_reachable
    WHERE link.2 = rec_reachable.1)
SELECT * FROM rec_rechable;
```

Example 1.4. The below recursive Datalog rule with (stratified) negation translates to SQL as:

```
link(Charpennes,Perrache).
link(PartDieu,Charpennes).
link(Debourg,PartDieu).
link(PartDieu,Debourg).
link(X,Y)→metro(X).
link(X,Y)→metro(Y).
link(X,Y)→reachable(X,Y).
link(X,Z),reachable(Z,Y)→reachable(X,Y).
metro(Y),metro(Y), NEG reachable(X,Y)→unreachable(X,Y).
```

```
CREATE or REPLACE VIEW unreachable AS
WITH RECURSIVE rec_reachable AS (
  SELECT link.1, link.2 FROM link
  UNION ALL
   (SELECT link.1, rec_reachable.2
    FROM link, rec_reachable
    WHERE link.2 = rec_reachable.1))
SELECT *
FROM metro as m1, metro as m2
WHERE m1.1 <> m2.2
AND m1.1, m2.2 NOT IN
 (SELECT rec_reachable.1, rec_reachable.2 FROM rec_reachable);
```

2 Workflow

- Guided by the above example, you should first translate your input Datalog rules into their SQL analogues. As explained above, you have the choice whether to do it manually or implement the transformation yourself.
- Next, you should execute the obtained SQL statements in Oracle. Again, you have the choice whether to do it directly (manually) or whether to connect to the server via JDBC [2] (automatically, from your Java program).
- Finally, you should *compare* whether the results obtained running your Java implementation are consistent with those outputted by JDBC.

[2] `http://www.oracle.com/technetwork/java/javase/jdbc/`