Query Language on Ontologies : SparSQL

Emma Di Pasquale

27 May 2008

Preliminaries of SparSQL

• **What is an ontology?**

It's a conceptualization of a domain of interest expressed in logic.

Feature: It allows access in a conceptual way to data astracting the logic structure.

• **In which way is it expressed?**

A family of languages to express ontologies is DL-Lite.

• **Services related to ontologies**

The most important service is the Query Answering i.e. to establish if a statement follows or not follows logically from a knowledge base

Problem: Incomplete information $\Rightarrow$ Open World Assumption (OWA)

Effect: FOL/SQL queries on ontologies are undecidable

Solution: We can use CQs and UCQs that are decidable but have an expressive power limitated compared to the FOL.

Hence?: We want to identify a query language as expressive as possible, decidable and with computational complexity acceptable

Incomplete information problem

- The kwoledge bases rarely know all the facts concerning the world covered, so we can say that in them there is incomplete information.
- The databases model the incomplete information in a programmatic way and not declaratory because the value NULL does not have a precise meaning

Example:

Person

Name	Age
Paolo	25
Luca	34
Mario	30
Luisa	null

q(x):- (Person(x,e) $\wedge$ e<30) $\vee$ (Person(x,e) $\wedge$ e≥30)

Answer: {Paolo, Luca, Mario}

Incomplete information problem

- To overcome this problem the ontolgies adopt an OWA.

Example:

Person

Name
Paolo
Maria
Luisa

Student

Name
Luca
Maria

$\Rightarrow$

q(x):- (Person(x) $\wedge$ $\neg$Student(x))

Answer in CWA case: {Paolo, Luisa}
Answer in OWA case : { }

SparSQL

- SparSQL = SPARQL + SQL
- SparSQL is a concrete query language on DL-Lite ontologies.
- SparSQL implements the EQL-Lite (UCQ) language which adopts the following principle:
  on what you know you have a complete information $\Rightarrow$ CWA
  $\Rightarrow$ FOL queries are decidable
- SparSQL allows to operate a dynamic closure of the knowledge in a controlled way by the user and to recover an expressive capacity comparable to FOL remaining decidable.

Syntax of SparSQL

SELECT ListAttributesOrExpressions

FROM (sparqltable (<Query Sparql>) alias)+

[where Conditions]

[group by ListAttributesOfGrouping]

[having AggregatesConditions]

[order by ListAttributesOfSorting]

SPARQL

- Recursive acronym:
  SPARQL Protocol and RDF Query Language
- W3C standardization from January 2008
- Query Language for RDF data.

SPARQL (2)

Data in RDF

Predicate

Subject

Object

```
@prefix dc: <http://www.dis.uniroma1.it/dc/>
 _:a dc:title    'DL-Lite' .
 _:a dc:creator  'Mario'.
 _:b dc:creator  'Mario'.
 _:b dc:title    'Mastro'
```

Query: "Find other papers by the authors of a given paper."

```
PREFIX dc: <http://www.dis.uniroma1.it/dc/>
SELECT ?title
WHERE
{
    ?doc      dc:title    'DL-Lite' .
    ?doc      dc:creator  ?c .
    ?docOther rdf:type    'Article'
    ?docOther dc:creator  ?c .
    ?docOther dc:title    ?title
}
```

Basic Graph Pattern

N.B. Here there can't be variables that aren't present in the SPARQL query

rdf:type is a way to extract all concept instances

Query Result:

title
DL-Lite
Mastro

SPARQL (3)

SPARQL is used in SparSQL because:

- it's a way to express UCQs. SparSQL uses only the syntax of SPARQL that allows to express the UCQs;
- it allows to highlight the query syntax that goes to extract knowledge from DB
- it allows to refer to the attributes of the tables through variables and this is useful because it is not said the user has had access to datasource and knows the exact name of the attributes

In SparSQL the SPARQL triple must have the following structure:

Subject ⇨	variable - constant - triple
Predicate ⇨	rdf:type - role name - attribute role name - concept attribute name
Object ⇨	variable - constant - concept name – role attribute name

Semantic of SparSQL

DEF: A SparSQL query is the following:

$$q(\vec{x}) :- T\,(K\ ucq_1, .., K\ ucq_n)$$

where:

- T is a SQL query.
- ucq_i is one UCQ expressed in SPARQL.
- $\vec{x}$ is the vector of free variables to return

i.e. the sparql tables of the SparSQL queries are the subjective queries in EQL with K operator.

Example of query in SparSQL (1)

Query: Return the number of articles published by every professor who are less than 35-years-old and has published more than 10 articles

```
SELECT professor.z, count(article.x)
FROM sparqltable (SELECT ?x ?y ?z
                  WHERE {?x rdf:type 'Professor'.
                         ?x age ?y.
                         ?x name ?z}) professor,
     sparqltable (SELECT ?x ?y
                  WHERE {?x rdf:type 'Article'.
                         ?x publicationAuthor ?y}) article
WHERE professor.x = article.y and
      professor.y < 35
GROUP BY professor.x
HAVING count(article.x) > 10;
```

We can use an aggregate function

The queries SPARQL express UCQs and extract knowledge by the ontology

q(x,y,z) :- K(Professor(x) ∧ age(x,y) ∧ name(x,z))

We can use an aggregate operator

Join between sparqltables

Example of query in SparSQL (2)

Query: Return the professors whose address is 'ROMA' and who aren't full professors

```
SELECT professor.x
FROM sparqltable (
          SELECT ?x
          WHERE {?x rdf:type Professor.
                 ?x address ' ROMA'})
         professors

EXCEPT

SELECT fullP.x
FROM sparqltable (
        SELECT ?x
        WHERE {?x rdf:type 'FullProfessor'})
    fullP;
```

```
SELECT professors.x
FROM sparqltable(
          SELECT ?x
          WHERE {?x rdf:type 'Professor'.
                 ?x address ' ROMA' })
          professors
WHERE professors.x not in (
      SELECT fullP.x
      FROM sparqltable(
        SELECT ?x
        WHERE {?x rdf:type 'FullProfessor' })
        fullP);
```

We can use the set operators like EXCEPT, UNION, INTERSECT

Like an UCQ we can considered a CONSTANT

The nested query is again a SparSQL query

Example of query in SparSQL (3)

Query: Return the students' name who are not graduated from 1995 to 1997 at "Sapienza" University

```
SELECT sd.ns
FROM sparqltable(select ?y ?nu ?ns
                        where{?x rdf:type 'Student'.
                              (?x degreeFrom ?u) degreeYear ?y.
                              ?u name ?nu.
                              ?x name ?ns}) sd
WHERE sd.nu like '%SAPIENZA%' and
         sd.y not between 1995 and 1997
ORDER BY desc sd.ns;
```

Complex triple to express a role attribute

The SQL operators can be applied on ontology knowledge

Example of query in SparSQL (4)

Query: Return the number of students graduated at "Sapienza" or at "Politecnico of Torino" University and advised by a full or associate professor

```
SELECT count(stud.s)
FROM sparqltable(select ?s ?u ?a
                    where {?s rdf:type 'GraduateStudent'.
                              ?s advisor ?a.
                              ?s degreeFrom ?u}) stud
WHERE stud.u like '%sapienza%' or
          stud.u like '%politecnico%torino%' and
          stud.a in (SELECT prof.x
                    FROM sparqltable(select ?x
                                          where {{?x rdf:type 'FullProfessor'}
                                                  union
                                                {?x rdf:type 'AssociateProfessor'}}) prof);
```

Union of conjunctive queries

Example of query in SparSQL(5)

```
SELECT distinct prof.n
FROM sparqltable (select ?p ?c ?n
                  where {?p rdf:type 'Professor'.
                         ?p teacherOf ?c.
                         ?p name ?n}) prof,
     sparqltable (select ?st ?c
                  where {?st rdf:type 'Student'.
                         ?st takesCourse ?c}) stud
WHERE prof.c = stud.c and
      stud.st not in (select s.x
                      from sparqltable(select ?x ?a ?n
                                       where{?x rdf:type 'Student'.
                                             ?x address ?a.
                                             ?x name ?n}) s
                      where (s.a like '%roma%' or s.a like '%milano%' ) and
                            s.n not like '%Maria%' and
                            s.x not in (select gs.x
                                        from sparqltable(select ?x
                                                         where{?x rdf:type 'GraduateStudent'}) gs ));
```

As in SQL there may be several queries nested

Boolean queries SparSQL

- A boolean query returns true or false.
- Problem: Is there in the SQL syntax a way to express boolean queries?

There isn’t explicitly, but we can consider a SQL construct to express it. This construct is:

```
SELECT CASE WHEN <CONDITION>
         THEN 1 ELSE 0 END
 FROM (Select count(*)
         from <sistem table>) as alias;
```

Where the FROM clause is mandatory in the SQL syntax and the <system table> depends by the DBMS used (For example in MySql is “mysql.user”)

Boolean queries SparSQL (2)

• We can to express boolean queries also in the FROM clause of a SparSQL query in the case we want evaluate a query if a given condition is verified.
Example: Give me the number of Sapienza's students if there is a student who has more than 40 years.

• The boolean queries in FROM clause are expressed in SPARQL (these queries are always sparqltables i.e. UCQs). In SPARQL exists the following way to express boolean queries:

ASK {<Basic Graph Pattern>**}**

Where <Basic Graph Pattern> is a set of unions of triples (i.e. UCQ) and the following query is evaluated on the DB:

```
SELECT CASE WHEN exists (EXPANDED and UNFOLDED ground query relative to
                                   <Basic Graph Pattern>)
THEN 1 else 0 END as value
FROM (SELECT count(*)
         FROM <system table>)) as alias_sparqltable
WHERE alias_sparqltable.value = 1
```

Boolean queries SparSQL (3)

Hence a boolean sparqltable is empty if it's calcolated with a SPARQL boolean query that not is verified.
So if in the FROM clause of a SparSQL normal query (or SparSQL boolean query) there are normal sparqltables and boolean sparqltables is done the cartesian product and if at least one boolean sparqltable isn't verified then the full SparSQL query is empty (or not is verified).

Syntax of boolean SparSQL queries

The user uses the following syntax that semplify the previous syntax in SQL :

VERIFY <Condition>

Where <Condition> is the same <Condition> that can be in WHERE clause of a SparSQL query.

Example of boolean query in SparSQL (1)

Query: Is there a professor living in Rome who is not a full professor?

```
VERIFY exists( select professors.x
                from sparqltable(select ?x ?y
                                 where {?x rdf:type 'Professor'.
                                        ?x address ?y }) professors

                where professors.y like '%Rome%' and
               professors.x not in (
select fullP.                                                           from
sparqltable(                                                        select ?x
                                                                    where {?
x rdf:type 'FullProfessor'}) fullP));
```

Example of boolean query in SparSQL (2)

Query: Is there a full or associate professor that lives in Milan?

```
VERIFY exists(select prof.x
                  from sparqltable(
                           select?x
                           where{?x rdf:type 'FullProfessor'.
                                  ?x address ' Milan'}) prof
                  union
                  select prof.x
                  from sparqltable(
                               select?x
                               where{?x rdf:type 'AssociateProfessor'.
                                      ?x address ' Milan'}) prof);
```

Example of boolean query in SparSQL(3)

Query: I want to know if all those who are graduated at “Sapienza” in 2007 are less than 30-years-old and if exists a student graduated whose name is Luca

we can put all type of conditions in SparSQL boolean query not only the conditions with "EXISTS" or "NOT EXISTS".

```
VERIFY 30 > all
       ( SELECT graduates.a
         FROM sparqltable (
                  SELECT ?a ?u
                  WHERE {(?x degreeFrom ?u) degreeYear '2007'.
                          ?x age ?a}) graduates
         WHERE graduates.u like '%Sapienza%')
         and exists (SELECT g.x
                  FROM sparqltable(SELECT ?x
                                WHERE {?x rdf:type ‘GraduateStudent’.
                                        ?x name ‘Luca’}) g);
```

We can have more conditions like the WHERE clause in a SparSQL or SQL query

Example of boolean query in SparSQL(4)

Query: Return true iff there are 2 students named ‘Luca’ and ‘Mario’ who have got graduation in 2007

```
VERIFY 'Luca' in (
        SELECT allstud.n
        FROM sparqltable(
                   SELECT ?n
                    WHERE{?x rdf:type 'Student'.
                                ?x name ?n.
                               (?x degreeFrom ?u) degreeYear '2007'}) allstud,
                 sparqltable(
                    ASK {?x rdf:type 'Student'.
                            ?x name 'Mario'.
                           (?x degreeFrom ?u) degreeYear '2007'}) stud);
```

N.B. it isn't possible to do any join between normal sparqltables and boolean sparqltables.

Boolean query in FROM clause

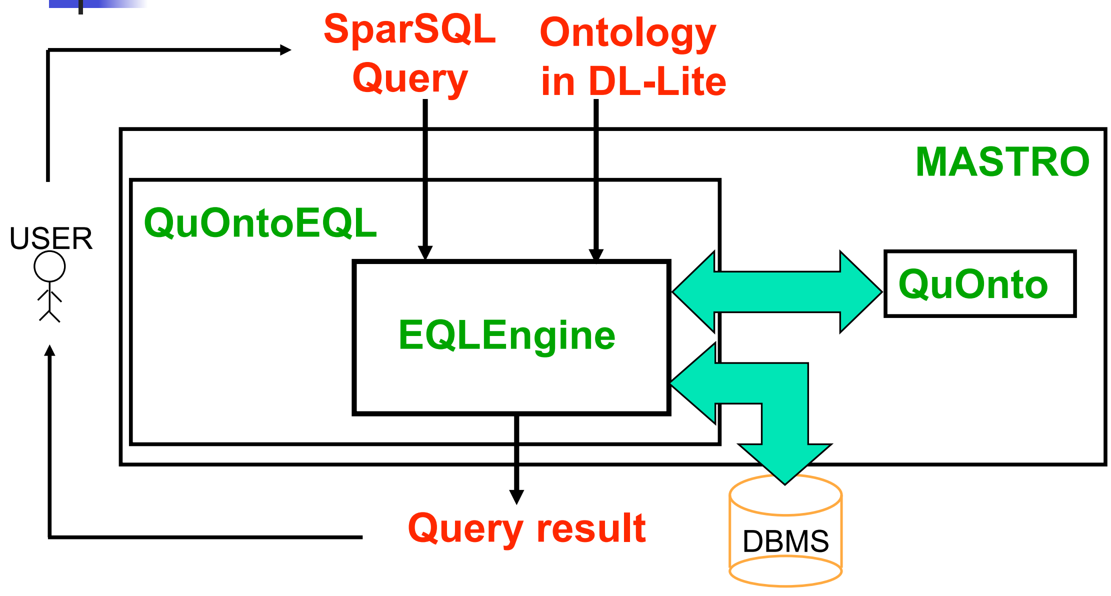
Example of boolean query in FROM clause

Query: Return all publications made on 10-10-2007 if in the same day Luca and Andrea published at least one paper, otherwise return null:

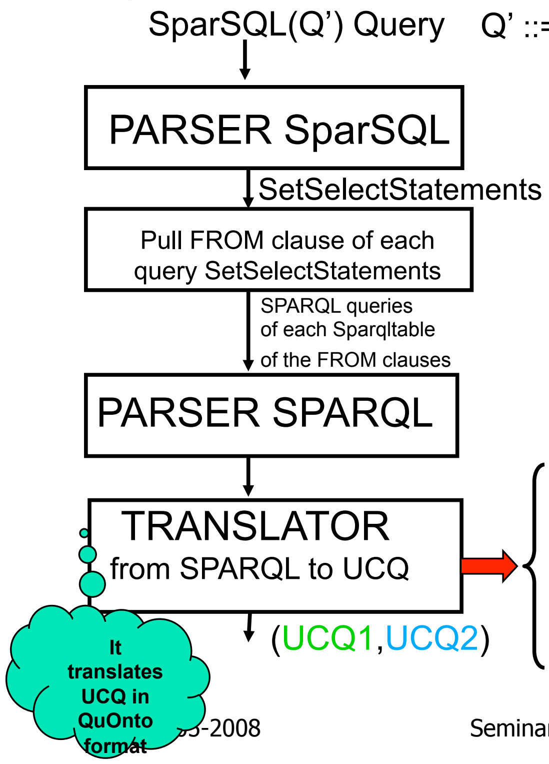
```
SELECT pub.x
FROM sparqltable(select ?x
                        where {?x rdf:type 'Publication'.
                                  ?x publicationDate '10-10-2007'}) pub,
        sparqltable(ASK{{?x rdf:type 'Publication'.
                                ?x publicationAuthor 'Luca'.
                                ?x publicationDate '10-10-2007'}
                                union
                               {?y rdf:type 'Publication'.
                                ?y publicationAuthor 'Andrea'.
                                ?y publicationDate '10-10-2007'}})
pubauthor;
```

Boolean union of conjunctive queries

Software System for to process a query SparSQL: QuOntoEQL



Software System: QuOntoEQL (2)

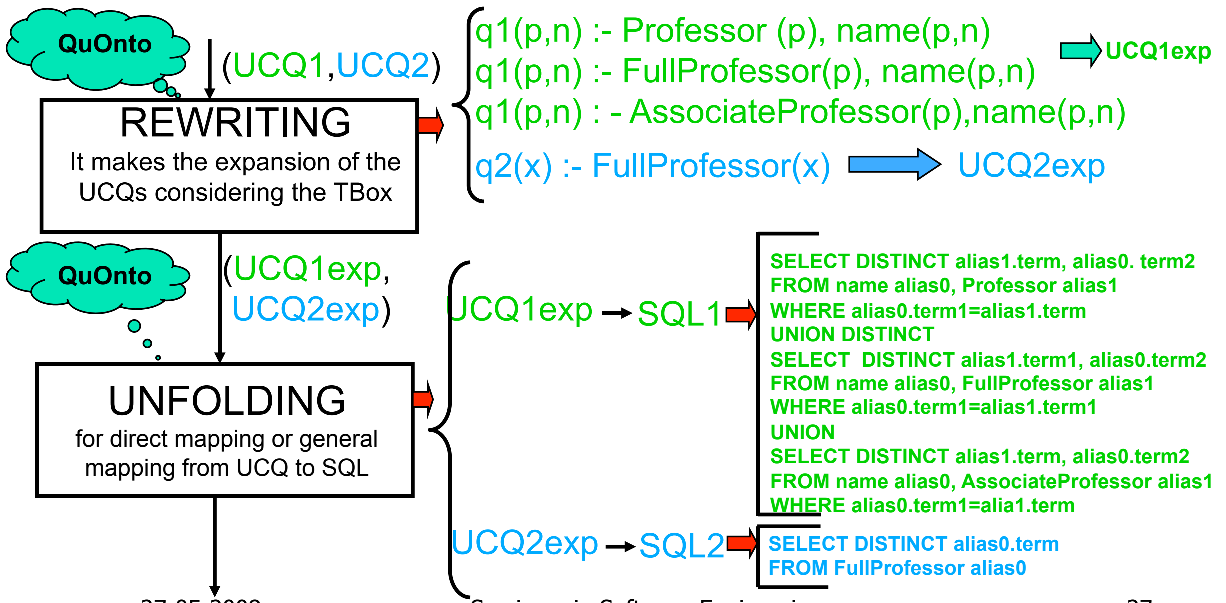

SparSQL(Q') Query
PARSER SparSQL
SetSelectStatements
Pull FROM clause of each query SetSelectStatements
SPARQL queries of each Sparqltable of the FROM clauses
PARSER SPARQL
TRANSLATOR from SPARQL to UCQ
(UCQ1,UCQ2)
It translates UCQ in QuOnto format

```
Q' ::= SELECT professor.n
       FROM SparqlTable (SELECT ?p ?n
                          WHERE {?p rdf:type 'Professor'.
                                 ?p name ?n}) professor
       WHERE professor.p NOT IN (
             SELECT FullP.x
             FROM SparqlTable( SELECT ?x
                                WHERE {?x rdf:type 'FullProfessor'})
                               FullP
```

q1(p,n) :- Professor(p), name(p,n)

q2(p,d) :- FullProfessor(x)

Software System: QuOntoEQL(3)

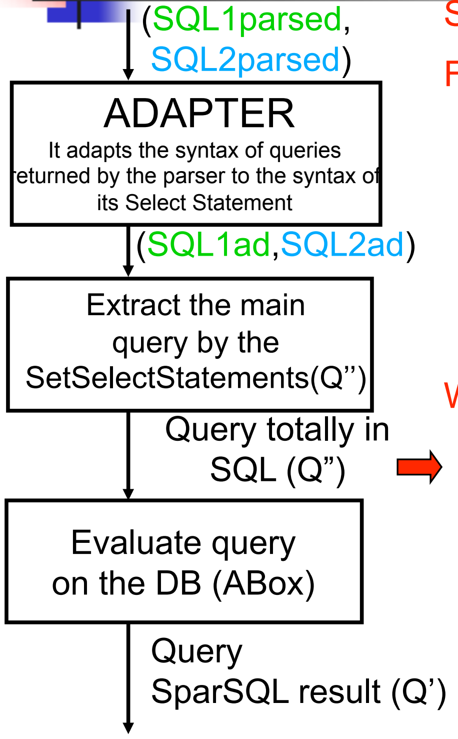

QuOnto
(UCQ1,UCQ2)
REWRITING
It makes the expansion of the UCQs considering the TBox
QuOnto
(UCQ1exp, UCQ2exp)
UNFOLDING
for direct mapping or general mapping from UCQ to SQL
UCQ1exp
UCQ2exp
UCQ1exp → SQL1
UCQ2exp → SQL2

```
q1(p,n) :- Professor (p), name(p,n)
q1(p,n) :- FullProfessor(p), name(p,n)
q1(p,n) : - AssociateProfessor(p),name(p,n)
q2(x) :- FullProfessor(x)
```

```
SELECT DISTINCT alias1.term, alias0. term2
FROM name alias0, Professor alias1
WHERE alias0.term1=alias1.term
UNION DISTINCT
SELECT  DISTINCT alias1.term1, alias0.term2
FROM name alias0, FullProfessor alias1
WHERE alias0.term1=alias1.term1
UNION
SELECT DISTINCT alias1.term, alias0.term2
FROM name alias0, AssociateProfessor alias1
WHERE alias0.term1=alia1.term
```

```
SELECT DISTINCT alias0.term
FROM FullProfessor alias0
```

Software System: QuOntoEQL (4)


(SQL1parsed, SQL2parsed)
ADAPTER
It adapts the syntax of queries returned by the parser to the syntax of its Select Statement
(SQL1ad,SQL2ad)
Extract the main query by the SetSelectStatements(Q'')
Query totally in SQL (Q'')
Evaluate query on the DB (ABox)
Query SparSQL result (Q')

```
SELECT professor.n
FROM (SELECT DISTINCT alias1.term p, alias0. term2 n
       FROM name alias0, Professor alias1
       WHERE alias0.term1=alias1.term
       UNION DISTINCT
       SELECT  DISTINCT alias1.term1, alias0.term2
       FROM name alias0, FullProfessor alias1
       WHERE alias0.term1=alias1.term1
      UNION DISTINCT
       SELECT DISTINCT alias1.term, alias0.term2
      FROM name alias0, AssociateProfessor alias1
      WHERE alias0.term1=alia1.term) professor
WHERE professor.p NOT IN
  (SELECT effettua.x
   FROM (SELECT DISTINCT alias0.term
         FROM FullProfessor alias0) fullP)
```

Computational complexity of the queries SparSQL evaluation

Respect to data	Respect to TBox	Respect to query
LOGSPACE Same complexity of FOL/SQL query on a relational database	PTIME Related to the expansions	PSPACE But the CQs on a DB are already polynomials respect to the query and LOGSPACE respect to the data

Hence SparSQL queries on ontologies have the same cost of SQL queries on a DB.