RC25402 (WAT1308-033) August 20, 2013
Computer Science

IBM Research Report

An Introduction to SPARQL Optionals

Julian Dolby, Kavitha Srinivas
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 208
Yorktown Heights, NY 10598
USA

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Why Do We Need OPTIONALS?

Deals with sparsity in graph data. A simple example where this is useful:

```
John age 42
John lastName Smith
John spouse Mary
John homeState New_York
New_York isIn NorthEast
Todd age 42
Todd lastName Doe
Todd homeState Connecticut
Connecticut isIn NorthEast
Tom age 42
```

Say we want to find all the people who are aged 42 and live in the NorthEast, and optionally their spouses names, if its known. OPTIONAL allows you to express that.

What does it mean?

The best way to understand what an optional means is to think of it as **extending the immediately** preceding pattern.

And extending it means that it can bind variables that have not been bound in its preceding pattern; for variables that have been bound by the preceding pattern, the optional must match the value

A working dataset

```
AS mailbox   mailto:alice@example.net
AS name      Alice
AS nick      WhoMe?

BD mailbox   mailto:bert@example.net
BD name      Bert

EF mailbox   mailto:eve@example.net
EF nick      DuckSoup
```

Many of the examples here are from Jena's ARQ suite

The SIMPLEST CASE

```
SELECT ?x ?mbox ?name
   {
     ?x mailbox ?mbox .
     OPTIONAL { ?x name  ?name } .
   }
```

Simplest case: optional pattern extends bindings from the preceding pattern, and provides binding for 'name' in the OPTIONAL pattern.

ResultSet:

x	mbox	name
AS	mailto:alice@example.net	Alice
BD	mailto:bert@example.net	Bert
EF	mailto:eve@example.net	NULL

Multiple OPTIONALS

```
SELECT ?x ?mailbox ?name ?nick
   {
     ?x mailbox ?mbox .
     OPTIONAL { ?x name  ?name } .
     OPTIONAL { ?x nick  ?nick }.
   }
```

Optional **left-associative** (like minus), so **a** optional **b** optional **c** is really ((**a** optional **b**) optional **c**).

ResultSet:

x	mbox	name	nick
AS	mailto:alice@example.net	Alice	WhoMe?
BD	mailto:bert@example.net	Bert	NULL
EF	mailto:eve@example.net	NULL	DuckSoup

Multiple OPTIONALS Contd.

```
SELECT ?label
   {
     ?x mailbox ?mbox .
     OPTIONAL { ?x name  ?label } .
     OPTIONAL { ?x nick  ?label } .
   }
```

Note that the second optional binds values to a variable bound by another optional. Thus, the inner pattern will bind x, mbox and optionally label. If it binds label, then the outer pattern has to match it and cannot extend it. So result will have all mbox properties, and all name properties that have an mbox. The outer pattern extends these results. If a subject has no name property, then any nick property for it will be returned. If it does have a name property, then a nick property will be returned only if its target matches a name property for the same subject.

However, only label is projected. So then, if there is a name property, label will be bound to that. If not, then it will bind any nick property. Thus, for Alice, we should see the name and no nickname.

ResultSet

Label
Alice
Bert
DuckSoup

AS, BD, EF all have mailboxes. All should match for main.
Optional 1: Extends with name for AS and BD, null for EF.
Optional 2: Extends optional 1 result set with nickname for EF, since for the AS and BD, nick HAS to match bindings of label in optional 1.

Multiple OPTIONALS Contd.

```
SELECT ?label
   {
     ?x mailbox ?mbox .
     OPTIONAL { ?x name  ?label } .
     OPTIONAL { ?x nick  ?label } .
   }
```

```
SELECT ?label
   {
     ?x mailbox ?mbox .
     OPTIONAL {
       { ?x name  ?label } UNION
       { ?x nick  ?label } .
     }
   }
```

Multiple OPTIONAL clauses binding the same variable is similar to yet subtly different from a UNION inside the OPTIONAL. The lefthand query gets 'nick' labels for mailboxes that have no 'name'; the righthand query gets all 'nick' and 'name' labels for any mailbox.

Label
Alice
Bert
DuckSoup

Label
Alice
Bert
DuckSoup
WhoMe?

Multiple OPTIONAL clauses binding the same variable is rarely what you want

OPTIONAL WITH EMPTY

```
SELECT ?x ?nick
   {
     OPTIONAL { ?x nick  ?nick }.
   }
```

Here, no pattern to extend (i.e., the pattern is empty). An empty pattern binds no variables, so the optional 'extends' the empty result set with any of its variables. Note this is equivalent to dropping the OPTIONAL altogether.

ResultSet:

x	nick
AS	WhoMe?
EF	DuckSoup

NESTING OPTIONALS

```
SELECT ?x ?mbox ?name ?nick
   {
     ?x mailbox ?mbox .
     OPTIONAL {
       ?x nick  ?nick
       OPTIONAL { ?x name  ?name }
     }
   }
```

In the nested optional case, we have **(a OPTIONAL (b OPTIONAL c)).** Note this differs from the case where we have two optionals that are not nested. In this case, the main pattern once again matches all three individuals, but the optionals being nested means the second optional has the result of the first optional as its main pattern. Thus, the results have mbox, and optionally nick, and possibly name if it has nick too. In particular, the name of bert is not returned, since it has no nick property. Note that **b** first extends **a,** and then **c** will extend **b.**

ResultSet

x	mbox	name	nick
AS	mailto:alice@example.net	Alice	WhoMe?
BD	mailto:bert@example.net	NULL	NULL
EF	mailto:eve@example.net	NULL	DuckSoup

NO-OP OPTIONALS

```
SELECT ?nick
   {
     AS nick ?nick .
     OPTIONAL {AS name  ?nick}
   }
```

In this case, the optional does not really have any new variables to bind. ?nick is already bound to “Who Me”, so the optional is a no-op.

ResultSet

nick
WhoMe?

More complex optionals

```
SELECT ?x ?mbox ? nick ?name
   {
     ?x mailbox ?mbox .
     OPTIONAL {
       ?x nick ?nick .
       ?x name ?name } .
   }
```

The optional pattern here contains an AND. That is, it can extend bindings from the preceding pattern for ?x only when ?x has both nick and name. Only Alice will have bindings for ?nick and ?name.

ResultSet:

x	mbox	name	nick
AS	mailto:alice@example.net	Alice	WhoMe?
BD	mailto:bert@example.net	NULL	NULL
EF	mailto:eve@example.net	NULL	NULL

OPTIONAL and FILTER

```
SELECT ?x ?name
   {
     ?x mailbox ?mbox .
     OPTIONAL {
       ?x name  ?name .
       FILTER (?name = “Bert”)
     } .
   }
```

```
SELECT ?x ?name
   {
     ?x mailbox ?mbox .
     OPTIONAL {
       ?x name  ?name .
     }
     FILTER (?name = “Bert”)
   }
```

- FILTER applies only to pattern containing it
- OPTIONAL only vs. main pattern

x	name
AS	NULL
EF	NULL
BD	Bert

x	name
BD	Bert

Disconnected Optionals

```
SELECT ?x ?nick ?name
   {
     ?x nick ?nick .
     OPTIONAL { ?y name  ?name } .
   }
```

The optional pattern extends bindings from the preceding pattern but shares no variables in common. Basically this will be a cross product.

ResultSet:

x	nick	y	Name
AS	WhoMe?	AS	Alice
AS	WhoMe?	BD	Bert
EF	DuckSoup	AS	Alice
EF	DuckSoup	BD	Bert

Cross products generally perform terribly; avoid them

Disconnected Optionals And filters

```
SELECT ?x ?nick ?name
   {
     ?x nick ?nick .
     OPTIONAL { ?y name  ?name FILTER (?x=AS) } .
   }
```

The optional pattern extends bindings from the preceding pattern but shares no variables in common. Basically this will be a cross product.

ResultSet:

x	nick	y	Name
AS	WhoMe?	AS	Alice
AS	WhoMe?	BD	Bert
EF	DuckSoup	NULL	NULL

OPTIONAL AND NEGATION

- OPTIONAL can test for *absence* of triples
  - Uses FILTER and not(bound(?v))
  - FILTER for ?v to be NULL, i.e. not exist

```
SELECT ?x ?mbox
   {
     ?x mailbox ?mbox .
     OPTIONAL { ?x name ?name }
     FILTER (not(bound(?name))
   }
```

x	mbox
EF	mailto:eve@example.net

EF is the only entity with no name property

SPARQL 1.1 Negation

- SPARQL 1.1 added explicit negation
  - obviates not(bound(?v))

```
SELECT ?x ?mbox
   {
     ?x mailbox ?mbox .
     FILTER NOT EXISTS {
       ?x name ?name }
   }
```

x	mbox
EF	mailto:eve@example.net

EF is the only entity with no name property

SPARQL 1.1 MINUS

- MINUS computes set difference
  - expresses some forms of negation

```
SELECT ?x ?mbox
   {
     ?x mailbox ?mbox .
     MINUS {
       ?x name ?name
     }
   }
```

x	mbox
EF	mailto:eve@example.net

subtract entities with name from entities with mailbox