SPARQL 1.1 Update

Presentation by Johann Schiffer (7602440)
for 184.729 Semantic Web Technologies

July 04, 2012

Outline

1. Introduction
2. Graph Manipulation
3. Triple Manipulation
4. Tests with Fuseki
5. Update and Entailment

Outline

1. Introduction
2. Graph Manipulation
3. Triple Manipulation
4. Tests with Fuseki
5. Update and Entailment

Can we generate and store Triples ?

Can we generate and store Triples ?

Constructing triples is possible in SPARQL 1.0, but ..

- Triples cannot be stored !!

Can we generate and store Triples ?

Constructing triples is possible in SPARQL 1.0, but ..

- Triples cannot be stored !!

SPARQL 1.1, allows to store and modify triples, i.e.

Can we generate and store Triples ?

Constructing triples is possible in SPARQL 1.0, but ..

- Triples cannot be stored !!

SPARQL 1.1, allows to store and modify triples, i.e.

- Create, Load, Clear, Move, Merge, Delete **RDF Graphs**

Can we generate and store Triples ?

Constructing triples is possible in SPARQL 1.0, but ..

- Triples cannot be stored !!

SPARQL 1.1, allows to store and modify triples, i.e.

- Create, Load, Clear, Move, Merge, Delete **RDF Graphs**
- Insert, Delete, Replace **RDF Triples**

Outline

1. Introduction

2. Graph Manipulation

3. Triple Manipulation

4. Tests with Fuseki

5. Update and Entailment

Graph Store

Graph Store is a mutable container of RDF graphs, consisting of

- Default graph $G_{default}$
- Named graphs with identifiers ($< iri_1 >$, G_1), ... , ($< iri_n >$, G_n) where
  - $n \geq 0$
  - $< iri_i > \neq < iri_j >$ for $1 \leq i < j < n$

Graph Store

Graph Store is a mutable container of RDF graphs, consisting of

- Default graph $G_{default}$
- Named graphs with identifiers $(< iri_1 >, G_1)$, ... , $(< iri_n >, G_n)$ where
  - $n \geq 0$
  - $< iri_i > \neq < iri_j >$ for $1 \leq i < j < n$

In SPARQL 1.1 Update Language Graphs are addressed by

- `DEFAULT`
- `GRAPH` iri_j
- iri_j
- `NAMED` (all named graphs)
- `ALL` (default graph and all named graphs)
- if not explicitely specified, Default Graph is addressed

Accessing an RDF Store

- Access to a graph store is defined in **Graph Store Protocols**
- Update Request contains **zero or more operations**
- Update Request is **atomic**
- execution of operations within one request must have the same effect as executing them in **lexical order**
  (according to `SPARQL 1.1 UpdateW3C Working Draft 05 January 2012`)

Operations on Graphs (1)

- **LOAD** (populate graph with triples contained in a document)

Operations on Graphs (1)

- **LOAD** (populate graph with triples contained in a document)

```
LOAD <file:///C:/SPARQL_Tools/Fuseki/foaf.ttl> INTO
<http://example/graph1>
```

Operations on Graphs (1)

- **LOAD** (populate graph with triples contained in a document)

  ```
  LOAD <file:///C:/SPARQL_Tools/Fuseki/foaf.ttl> INTO
  <http://example/graph1>
  ```

- **CREATE** (create empty graph)

Operations on Graphs (1)

- **LOAD** (populate graph with triples contained in a document)

  ```
  LOAD <file:///C:/SPARQL_Tools/Fuseki/foaf.ttl> INTO
  <http://example/graph1>
  ```

- **CREATE** (create empty graph)

  ```
  CREATE GRAPH <http://example/graph1>
  ```

Operations on Graphs (1)

- **LOAD** (populate graph with triples contained in a document)

  ```
  LOAD <file:///C:/SPARQL_Tools/Fuseki/foaf.ttl> INTO
  <http://example/graph1>
  ```

- **CREATE** (create empty graph)

  ```
  CREATE GRAPH <http://example/graph1>
  ```

- **DROP** (remove graph)

Operations on Graphs (1)

- **LOAD** (populate graph with triples contained in a document)

```
LOAD <file:///C:/SPARQL_Tools/Fuseki/foaf.ttl> INTO
<http://example/graph1>
```

- **CREATE** (create empty graph)

```
CREATE GRAPH <http://example/graph1>
```

- **DROP** (remove graph)

```
DROP GRAPH <http://example/graph1>
```

Operations on Graphs (1)

- **LOAD** (populate graph with triples contained in a document)

  ```
  LOAD <file:///C:/SPARQL_Tools/Fuseki/foaf.ttl> INTO
  <http://example/graph1>
  ```

- **CREATE** (create empty graph)

  ```
  CREATE GRAPH <http://example/graph1>
  ```

- **DROP** (remove graph)

  ```
  DROP GRAPH <http://example/graph1>
  ```

  ```
  DROP NAMED
  ```

Operations on Graphs (2)

- **MOVE** (move triples to another graph)

Operations on Graphs (2)

- **MOVE** (move triples to another graph)

  ```
  MOVE GRAPH <http://example/graph1> TO GRAPH
  <http://example/graph2>
  ```

Operations on Graphs (2)

- **MOVE** (move triples to another graph)

  ```
  MOVE GRAPH <http://example/graph1> TO GRAPH
  <http://example/graph2>
  MOVE DEFAULT TO GRAPH <http://example/graph3>
  ```

Operations on Graphs (2)

- **MOVE** (move triples to another graph)

  ```
  MOVE GRAPH <http://example/graph1> TO GRAPH
  <http://example/graph2>
  ```

  ```
  MOVE DEFAULT TO GRAPH <http://example/graph3>
  ```

- **CLEAR** (remove all triples)

Operations on Graphs (2)

- **MOVE** (move triples to another graph)

  ```
  MOVE GRAPH <http://example/graph1> TO GRAPH
  <http://example/graph2>
  ```

  ```
  MOVE DEFAULT TO GRAPH <http://example/graph3>
  ```

- **CLEAR** (remove all triples)

  ```
  CLEAR GRAPH <http://example/graph2>
  ```

Operations on Graphs (2)

- **MOVE** (move triples to another graph)

  ```
  MOVE GRAPH <http://example/graph1> TO GRAPH
  <http://example/graph2>
  ```

  ```
  MOVE DEFAULT TO GRAPH <http://example/graph3>
  ```

- **CLEAR** (remove all triples)

  ```
  CLEAR GRAPH <http://example/graph2>
  ```

- **COPY** (insert triples and remove before)

Operations on Graphs (2)

- **MOVE** (move triples to another graph)

  ```
  MOVE GRAPH <http://example/graph1> TO GRAPH
  <http://example/graph2>
  ```

  ```
  MOVE DEFAULT TO GRAPH <http://example/graph3>
  ```

- **CLEAR** (remove all triples)

  ```
  CLEAR GRAPH <http://example/graph2>
  ```

- **COPY** (insert triples and remove before)

  ```
  COPY DEFAULT TO <http://example/graph2>
  ```

Operations on Graphs (2)

- **MOVE** (move triples to another graph)

  ```
  MOVE GRAPH <http://example/graph1> TO GRAPH
  <http://example/graph2>
  ```

  ```
  MOVE DEFAULT TO GRAPH <http://example/graph3>
  ```

- **CLEAR** (remove all triples)

  ```
  CLEAR GRAPH <http://example/graph2>
  ```

- **COPY** (insert triples and remove before)

  ```
  COPY DEFAULT TO <http://example/graph2>
  ```

- **ADD** (add triples)

Operations on Graphs (2)

- **MOVE** (move triples to another graph)

  ```
  MOVE GRAPH <http://example/graph1> TO GRAPH
  <http://example/graph2>
  ```

  ```
  MOVE DEFAULT TO GRAPH <http://example/graph3>
  ```

- **CLEAR** (remove all triples)

  ```
  CLEAR GRAPH <http://example/graph2>
  ```

- **COPY** (insert triples and remove before)

  ```
  COPY DEFAULT TO <http://example/graph2>
  ```

- **ADD** (add triples)

  ```
  ADD DEFAULT TO <http://example/graph2>
  ```

Outline

1. Introduction
2. Graph Manipulation
3. Triple Manipulation
4. Tests with Fuseki
5. Update and Entailment

Insert Triples

- `INSERT DATA [GRAPH <iri1>] <TripleData>`

  Triples from `<TripleData>` (must not contain variables) are inserted in <iri1> or default graph.

Insert Triples

- `INSERT DATA [GRAPH <iri1>] <TripleData>`

  Triples from `<TripleData>` (must not contain variables) are inserted in <iri1> or default graph.

-
  ```
  [ WITH <iri0> ]
  INSERT [GRAPH <iri1>] <TriplePattern1>
  (USING [NAMED] <iri>)*
  WHERE <TriplePattern2>
  ```

Insert Triples

- `INSERT DATA [GRAPH <iri1>] <TripleData>`

  Triples from `<TripleData>` (must not contain variables) are inserted in <iri1> or default graph.

-
  ```
  [ WITH <iri0> ]
  INSERT [GRAPH <iri1>] <TriplePattern1>
  (USING [NAMED] <iri>)*
  WHERE <TriplePattern2>
  ```

  - `WHERE` clause is evaluated,
    if specified, graphs defined in `USING [NAMED]` are used.
  - solutions are applied to `<TriplePattern1>` and the resulting triples are inserted into the graph defined by `<iri1>` (or default graph)
  - if a graph is specified by `WITH <iri0>` it is used as default graph for the `INSERT` operation

Insert Triples

- `INSERT DATA [GRAPH <iri1>] <TripleData>`

  Triples from `<TripleData>` (must not contain variables) are inserted in <iri1> or default graph.

-
  ```
  [ WITH <iri0> ]
  INSERT [GRAPH <iri1>] <TriplePattern1>
  (USING [NAMED] <iri>)*
  WHERE <TriplePattern2>
  ```

  - `WHERE` clause is evaluated,
    if specified, graphs defined in `USING [NAMED]` are used.
  - solutions are applied to `<TriplePattern1>` and the resulting triples are inserted into the graph defined by `<iri1>` (or default graph)
  - if a graph is specified by `WITH <iri0>` it is used as default graph for the `INSERT` operation

- no graphs are created
  if ground triple already contained, it is not inserted
  blank nodes are newly created and different from existing ones

Insert Triples (example)

```
prefix foaf:  <http://xmlns.com/foaf/0.1/>
prefix fb:  <http://rdf.freebase.com/ns/>
LOAD <file:///C:/SPARQL_Tools/Fuseki/foaf.ttl> into
<http://example/graph1>;
INSERT DATA
GRAPH <http://example/graph1>
fb:en.bill_gates foaf:name "Bill Gates"
```

Delete Triples

- `DELETE DATA [GRAPH <iri1>] <TripleData>`

  Triples from `<TripleData>` (must contain nor variables neither blank nodes) are deleted from <iri1> or default graph

Delete Triples

- `DELETE DATA [GRAPH <iri1>] <TripleData>`

  Triples from `<TripleData>` (must contain nor variables neither blank nodes) are deleted from <iri1> or default graph

-
  ```
  [ WITH <iri0> ]
  DELETE [GRAPH <iri1>] <TriplePattern1>
  (USING [NAMED] <iri>)*
  WHERE <TriplePattern2>
  ```

Delete Triples

- `DELETE DATA [GRAPH <iri1>] <TripleData>`

  Triples from `<TripleData>` (must contain nor variables neither blank nodes) are deleted from <iri1> or default graph

-
  ```
  [ WITH <iri0> ]
  DELETE [GRAPH <iri1>] <TriplePattern1>
  (USING [NAMED] <iri>)*
  WHERE <TriplePattern2>
  ```

  - `WHERE` clause is evaluated,
    if specified, graphs defined in `USING [NAMED]` are used.
  - solutions are applied to `<TriplePattern1>` and the resulting triples are deleted from the graph defined by `<iri1>` (or default graph).
  - If such a triple not contained in the graph, or the graph does not exist, the operation has no effect.
  - if a graph is specified by `WITH <iri0>` it is used as default graph for the `DELETE` operation

Delete Triples (ctd.)

- `DELETE WHERE <TriplePattern>`
  abbreviation for
  `DELETE <TriplePattern> WHERE <TriplePattern>`

Delete Triples (ctd.)

- `DELETE WHERE <TriplePattern>`
  abbreviation for
  `DELETE <TriplePattern> WHERE <TriplePattern>`

- blank nodes cannot be deleted explicitely by `DELETE DATA`

  no graphs are removed

  if triple shall be deleted, but is not contained in the graph, or the graph does not exist, the operation has no effect.

Delete Triples (examples)

-
  ```
  prefix foaf:  <http://xmlns.com/foaf/0.1/>
  prefix fb:  <http://rdf.freebase.com/ns/>
  DELETE DATA
  GRAPH <http://example/graph1>
  fb:en.bill_gates foaf:name "Bill Gates"
  ```

-
  ```
  DELETE WHERE
  ?x foaf:name "Axel Polleres"
  ```

Update Triples

-
  ```
  [ WITH <iri0> ]
  DELETE [GRAPH <iri1>] <TriplePattern1>
  INSERT [GRAPH <iri2>] <TriplePattern2>
  (USING [NAMED] <iri>)*
  WHERE <TriplePattern3>
  ```

Update Triples

```
[ WITH <iri0> ]
DELETE [GRAPH <iri1>] <TriplePattern1>
INSERT [GRAPH <iri2>] <TriplePattern2>
(USING [NAMED] <iri>)*
WHERE <TriplePattern3>
```

Combines DELETE and INSERT comand:

- WHERE clause is evaluated (only once)
  if specified, graphs defined in USING [NAMED] are used.
- solutions are applied to <TriplePattern1> and the resulting triples are deleted from the graph defined by <iri1> (or default graph).
- If such a triple not contained in the graph, or the graph does not exist, the operation has no effect.
- solutions of the WHERE clause are applied to <TriplePattern2> and the resulting triples are inserted into the graph defined by <iri2> (or default graph).
- in general, triples are removed from one graph and new triples are inserted in another graph.
- in particular triples in a graph can be modified .

Update Triples

```
[ WITH <iri0> ]
DELETE [GRAPH <iri1>] <TriplePattern1>
INSERT [GRAPH <iri2>] <TriplePattern2>
(USING [NAMED] <iri>)*
WHERE <TriplePattern3>
```

Combines DELETE and INSERT comand:

- `WHERE` clause is evaluated (only once)
  if specified, graphs defined in `USING [NAMED]` are used.
- solutions are applied to `<TriplePattern1>` and the resulting triples are deleted from the graph defined by `<iri1>` (or default graph).
- If such a triple not contained in the graph, or the graph does not exist, the operation has no effect.
- solutions of the WHERE clause are applied to `<TriplePattern2>` and the resulting triples are inserted into the graph defined by `<iri2>` (or default graph).
- in general, triples are removed from one graph and new triples are inserted in another graph.
- in particular triples in a graph can be modified .

Update Triples (example)

```
prefix foaf:  <http://xmlns.com/foaf/0.1/>
prefix fb:  <http://rdf.freebase.com/ns/>
WITH <http://example/graph1>
DELETE
?x foaf:name "Bill Gates"
INSERT
?x foaf:name "Bill"
WHERE ?x foaf:name "Bill Gates"
```

Outline

1. Introduction
2. Graph Manipulation
3. Triple Manipulation
4. Tests with Fuseki
5. Update and Entailment

Practical Experience

Experiments made with Fuseki :

- SPARQL server, part of the JENA project
- ongoing development, tracks W3C standards
- uses the Graph store protocol
- includes GUI
- all examples of operations given above have been exhaustively and successfully tested with Fuseki

Outline

1. Introduction

2. Graph Manipulation

3. Triple Manipulation

4. Tests with Fuseki

5. Update and Entailment

Update may cause Problems

Until now, we have only considered Update in SPARQL from **syntactic** perspective.
But ..

- after a triple was deleted in RDFS, it should not be possible to deduce it anymore.

Update may cause Problems

Until now, we have only considered Update in SPARQL from **syntactic** perspective.
But ..

- after a triple was deleted in RDFS, it should not be possible to deduce it anymore.
- after a triple was added to OWL data, the data should stay consistent with OWL rules.

Update may cause Problems

Until now, we have only considered Update in SPARQL from **syntactic** perspective.
But ..

- after a triple was deleted in RDFS, it should not be possible to deduce it anymore.
- after a triple was added to OWL data, the data should stay consistent with OWL rules.

Interesting paper concerning this topic :
Updating RDFS: from Theory to Practice by Claudio Gutierrez. et al.

Deletion in RDFS

Gutierrez. et al. consider deletion of triples from RDFS data.

- **Deletion of instances**.
  An algorithm with polynomial time is given.

Deletion in RDFS

Gutierrez. et al. consider deletion of triples from RDFS data.

- **Deletion of instances**.
  An algorithm with polynomial time is given.

- **Deletion of schema data**.
  Problem is in general not tractable.

Deletion in RDFS

Gutierrez. et al. consider deletion of triples from RDFS data.

- **Deletion of instances**.
  An algorithm with polynomial time is given.
- **Deletion of schema data**.
  Problem is in general not tractable.

  No unique solution exists.
  E.g. chain of subclasses $(a_i \; sc \; a_{i+1})$; delete $(a_1 \; sc \; a_n)$

Deletion in RDFS

Gutierrez. et al. consider deletion of triples from RDFS data.

- **Deletion of instances**.
  An algorithm with polynomial time is given.

- **Deletion of schema data**.
  Problem is in general not tractable.

  No unique solution exists.
  E.g. chain of subclasses $(a_i\ sc\ a_{i+1})$; delete $(a_1\ sc\ a_n)$

  In this case the problem can be reduced to a graph theoretical problem where the size of the graph is small

References

- Bob DuCharme: **Learning SPARQL**. O´Reilly 2011
- Claudio Gutierrez, Carlos Hurtado, and Alejandro Vaisman: **Updating RDFS: from Theory to Practice**. ESWC 2011 http://users.dcc.uchile.cl/ cgutierr/papers/eswc2011.pdf
- http://www.w3.org/TR/sparql11-update/
- http://www.w3.org/2009/sparql/docs/update-1.1/Overview.xml
- http://jena.apache.org/documentation/serving_data/index.html