0.1 Persisting Ontologies

RDF Databases

On the web, RDF data can be stored in

- RDF files,

- HTML, embedded into RDFa annotations,

- RDF stores available through a SPARQL endpoint,
    - provide means to efficiently manage large RDF data,
    - considered NoSQL databases,
    - most of them are **relational RDF stores**, i.e. implemented on top of relational databases.

0.1.1 RDF Stores

RDF Store Fundamentals

Classification of relational RDF stores

vertical table stores store each triple in a three-column table (s, p, o)[1]

TRIPLES		
subject	**predicate**	**object**
John	loves	Mary
John	hates	Bob
Mary	loves	John
...		

property table stores store triples with the same subject as a n-ary table row, where predicates are modeled as table columns

TRIPLES_PER_SUBJECT		
subject	loves	hates
John	Mary	Bob
Mary	John	null
...		

horizontal table stores store triples with the same property in one table

loves		hates	
subject	**object**	**subject**	**object**
John	Mary	John	Bob
Mary	John	...	
...			

Existing triple stores

---

[1]The words in `Courier` mean IRIs, e.g. `John` means `http://e.cz/John`.

triple store	web site	type
4store	`http://4store.org`	property tables [1]
Allegro-Graph	`http://franz.com/agraph`	?
BigData	`http://bigdata.com`	vertical
TDB (Fuseki)	`http://jena.apache.org`	vertical
SDB	`http://jena.apache.org`	vertical (RDBMS)
Mulgara (Kowari)	`http://www.mulgara.org`	vertical
Oracle Spatial and Graph	`http://www.oracle.com`	vertical (RDBMS)
OWLIM	`http://www.ontotext.com/owlim`	vertical
Redland RDF Library	`http://www.ontotext.com/owlim`	vertical
OpenRDF Sesame	`http://www.openrdf.org`	vertical
StarDog	`http://stardog.com`	?
Virtuoso	`http://virtuoso.openlinksw.com`	vertical (RDBMS)

Transactional Processing in Triple Stores

Generally, it is difficult to provide ACID for triple stores. Most triple stores accept BASE instead:

ACID:

Atomicity – if an operation within a transaction fails, the whole transaction rolls back

Consistency – no constraint is violated in steady state (when no tx is running)

Isolation – one transaction does not see intermediate data of another transaction

Durability – after commit, a transactional data are kept persistent, preventing power loss, crashes, etc.

BASE:

BAsic availability – high data replication to prevent their loss on crash/system failure, etc,

Soft state – the data are soft, their consistency is the responsibility of the application developer,

Eventual consistency – data will converge to a consistent state at some point ...

Indexing in Triple Stores

- quad stores extend vertical triple stores with one more column for representing the context (named graph) in which the triple resides, i.e. (S,P,O,C),

- vertical stores typically create B-tree indexes on S,P,O(,C) columns

    Example

    OSPC index means that the index table contains triples sorted according to object, then according to subject, then predicate and then context. This index is suitable for searching data given an object (i.e. matching the BGP ?x ?y :a), or object+subject (e.g. matching the BGP ?x :p :a).

Materialization

- Some RDF stores use **materialization** to speed-up queries. This means that on each update the set of inferences is recomputed and stored.

  RDFS Materialization Example

  Listing 1 : Data for Insertion

  ```
  @prefix : <http://example.org/>
  :B rdfs:subClassOf :A .
  :C rdfs:subClassOf :B .
  ```

  Listing 2 : Stored data

  ```
  @prefix : <http://example.org/>
  :B rdfs:subClassOf :A .
  :C rdfs:subClassOf :B .
  :C rdfs:subClassOf :A .
  ```

OpenRDF Sesame

OpenRDF Sesame Features
  is an RDF triple store providing wide range of

Repository Types

- in memory

- filesystem

- relational database

- federated

- SPARQL endpoint

Access Types

- Java API

- CLI, Workbench

- SPARQL HTTP protocol

Inferencing

- No

- RDFS (materialized)

- Direct Type (materialized)

- Custom rules

Query Languages

- SPARQL

- SeRQL

OpenRDF Sesame

- sesame uses SPOC and POSC indexes by default,

- lacking user management support,

- ☺ simple and well-known system capable of handling big data,

- ☹ poor administration tools,

- more in tutorials . . .

OWLIM

- is an OWL repository built on Sesame

- implemented using rules (forward/backward chaining), so it is **incomplete** w.r.t. OWL, but provides *most* of the inferences,

- uses materialization

- many optimizations and extensions, e.g. spatial queries:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX omgeo: <http://www.ontotext.com/owlim/geo#>
PREFIX : <http://onto.mondis.cz/resource/npu/>

SELECT DISTINCT ?pagis ?pagis_lng ?pagis_lat ?idReg ?pagis_idobPg
WHERE {
    ?pagis geo:lat ?pagis_lat .
    ?pagis geo:long ?pagis_lng .
    ?pagis :hasIdReg ?idReg .
    ?pagis :hasIdobPg ?pagis_idobPg .
    ?pagis omgeo:within ( 10 10 200 200 )
}
```

Ontology Management in RDF Stores

Methods of Communication with RDF Stores

- custom APIs – Jena, Sesame,

```
Model m = ModelFactory.getModel("http://example.org/personal");
Resource i = m.getResource("http://example.org/person1");
i.addProperty(ResourceFactory.getProperty("http://example.org/hasName"),"John");
m.close();
```

- SPARQL Update through a SPARQL endpoint,

```
PREFIX : <http://example.org/>
INSERT { GRAPH :personal { :person1 :hasName "John" } }
```

- SPARQL Graph Store HTTP Protocol.

```
POST /gs?graph=http%3A%2F%2Fexample.com%2Fpersonal
Host: anyhost.com
Content-Type: text/turtle
@prefix : <http://example.com/>.
:person1 :hasName "John".
```

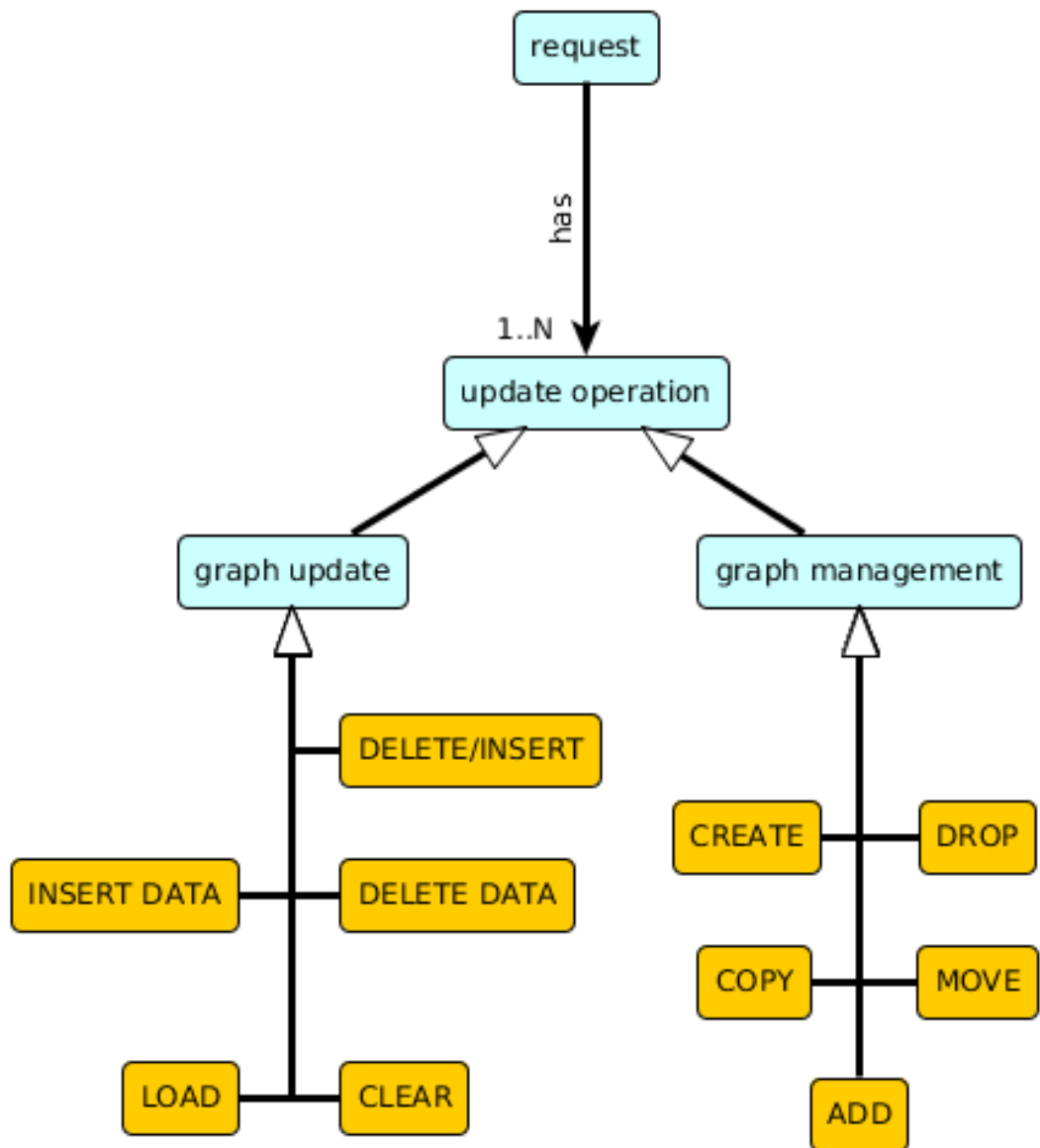## 0.1.2 SPARQL Update

Types of update operations

Each request consists of one or more operations:

graph update operations are **DELETE DATA**,
INSERT DATA,
DELETE/INSERT,
LOAD,
CLEAR

graph management operations are **CREATE**,
ADD,
COPY,
MOVE,
DROP

Graph Update Operations

INSERT DATA

Syntax and semantics

syntax INSERT DATA QD

semantics inserts ground data to a GS, blank nodes are considered disjoint with GS.

Example query RU1:

```
PREFIX : <http://example.org/>
INSERT DATA {
 :john a :Employee .
 GRAPH :personal {
   :john :hasGender 'male'.
 }
 GRAPH :corporate {
   :john :hasFunction _:d.
   _:d :hasName 'developer'.
 }
}
```

After two runs of RU1 w.r.t an empty GS

```
@prefix : <http://example.org/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
:john rdf:type :Employee.
:personal {
    :john :hasGender 'male'.}
:corporate {
    :john :hasFunction _:b.
    _:b :hasName 'developer.
    :john :hasFunction _:c.
    _:c :hasName 'developer. }
```

DELETE DATA

Syntax and semantics

syntax DELETE DATA *QD*

semantics deletes ground data from a GS, *b-nodes are forbidden in QD.*

Example query RU2:

```
PREFIX : <http://example.org/>
DELETE DATA {
 :john a :Employee .
 GRAPH :personal {
   :john :hasGender 'male'.
 }
}
```

RU2 returns an empty GS after a single run w.r.t the following graph:

```
@prefix : <http://example.org/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
:john rdf:type :Employee.
:personal {
    :john :hasGender 'male'.}
```

`DELETE/INSERT`

Syntax and semantics

syntax (**WITH** *iriRef*)?
      (*DelCls InsCls?* |*InsCls*)
      (**USING** (**NAMED**)? *iriRef*)*
      WHERE *GGP*

semantics deletes ground data from a GS, *b-nodes are forbidden in QD.*

`WITH` defines a graph that the query operates on (matching *GGP*, deleting in *DelCls*, inserting in *InsCls*)

```
PREFIX : <http://example.org/>
WITH :personal
INSERT { ?x a :ManWithIntoPersonal . }
WHERE { ?x :hasGender 'male'. }
```

`USING (NAMED)?` defines a graph that the **`WHERE`** part operates on.
      USING overrides **WITH**, and **GRAPH** overrides both.

```
PREFIX : <http://example.org/>
INSERT { GRAPH :newpersonal
{ ?x a :ManUsingIntoAnother . } }
USING NAMED :personal
WHERE { ?x :hasGender 'male'. }
```

`LOAD`

Syntax and semantics

syntax **LOAD** *iriRefS* (**INTO GRAPH** *iriRefT*)

semantics loads RDF document from the specified IRI into the specified graph.

`INTO GRAPH` *iriRefT* loads the data into the graph *iriRefT*.

No data are deleted from the graph *iriRefS*. If **`INTO GRAPH`** clause is missing, the data are inserted into the default graph.

```
PREFIX : <http://example.org/>
LOAD <http://www.w3.org/TR/owl-guide/wine.rdf>
INTO GRAPH :wine
```

CLEAR

Syntax and semantics

syntax[2] **CLEAR** (**GRAPH** *iriRef* |**DEFAULT** |**NAMED** |**ALL**)

semantics clears all data from the specified graph(s).

GRAPH *iriRef* clears the graph specified by the *iriRef*

DEFAULT clears *only* the default graph

NAMED clears *all* named graphs

ALL clears *all* graphs

```
PREFIX : <http://example.org/>
CLEAR GRAPH :newpersonal
```

Graph Management Operations

CREATE

Syntax and semantics

syntax CREATE GRAPH *iriRef*

semantics creates an empty named graph identified by *iriRef*

- for GSs (e.g. Sesame) that create/drop graphs "on demand" (e.g. during **INSERT**, **CLEAR**, etc.) the **CREATE** operation does nothing

```
PREFIX : <http://example.org/>
CREATE GRAPH :yetnewpersonal
```

DROP

Syntax and semantics

syntax DROP (**GRAPH** *iriRef* |**DEFAULT** |**NAMED** |**ALL**)

semantics drops (deletes) the specified graphs.

- for GSs (e.g. Sesame) that create/drop graphs "on demand", **DROP** does the same thing as **CLEAR**. In other cases, it additionally deletes the graph.

```
DROP ALL
```

COPY/MOVE/ADD

Syntax and semantics

syntax (**COPY**|**MOVE**|**ADD**) (**GRAPH** *iriRefS* |**DEFAULT**) **TO** (**GRAPH** *iriRefT* |**DEFAULT**))

semantics copies/moves/adds data from one named/default graph to another named/default graph.

- **MOVE** deletes the data in the source graph, **COPY/ADD** leaves them untouched.

- **COPY,MOVE** deletes the data in the target graph, **ADD** leaves them untouched.

```
PREFIX : <http://example.org/>
COPY GRAPH :personal
TO GRAPH :yetnewpersonal2
```

Remarks

- **ADD**, **COPY**, resp. **MOVE**, can be simulated by **INSERT**, **DROP/INSERT**, resp. **DROP/INSERT/DROP** combination.

- all graph management operations have their **SILENT** form, similarly to **LOAD** and **CLEAR**.

0.1.3 SPARQL Graph Store HTTP Protocol

HTTP crash course

- Hypertext Transfer Protocol, currently HTTP 1.1, RFC 2616

- operations – **GET**, **POST**, **PUT**, **DELETE**, **HEAD**, **PATCH**

- return codes – 2xx (success), 3xx (redirection), 4xx (protocol error), 5xx (server error)

Graph Identification

direct means using graph IRI as request URI

```
GET /g1 HTTP/1.1
Host: ex.cz
Accept: text/turtle; charset=utf-8
```

indirect means using graph IRI as a request parameter whenever direct identification is not possible (Why ?).

```
GET /graph-store?graph=http%3A//ex.com/g1 HTTP/1.1
Host: example.cz
Accept: text/turtle; charset=utf-8
```

- **graph=http%3A...** and **default** are used to indirectly identify named/default graphs respectively, similarly to a SPARQL **GRAPH** clause.

HTTP operations

Description

GET retrieves an RDF graph corresponding to the referred graph (like SPARQL **CONSTRUCT**,

PUT stores the RDF payload as the referred graph in GS (like SPARQL **DROP/INSERT**),

DELETE removes the graph content of the referred graph in GS (like SPARQL **DROP**),

POST inserts the RDF payload content to the referred graph in GS (like SPARQL **INSERT**),

HEAD same as **GET**, but without returning the actual RDF content, e.g. for testing validity of dereferencable IRIs,

(PATCH) optionally embedding a SPARQL 1.1 Update request to modified the referred graph.

HTTP details

- **Accept** header for **GET** specifies the mime type for requested RDF

- **Content-type** header for **PUT,POST** specifies the mime type of the enclosed RDF payload

- *typical* RDF mime types

RDF/XML	*Turtle*	*N3*	*TriG*
application/rdf+xml	text/turtle	text/rdf+n3	application/x-trig

- *typical* HTTP error codes:

  400 Bad Request – failing to parse RDF payload according to the given Content-type.

  404 Not Found – the requested content does not exist

  405 Method Not Allowed – unsupported HTTP verb/malformed request syntax

  406 Not Acceptable – in case Accept header is invalid

  415 Unsupported Media Type – content type is not understood

- content type multipart/form-data for **POST** requests can be used to *RDF-merge* more RDF documents into a graph in GS.

Examples

```
GET /gs?graph=http%3A%2F%2Fex.com%2Fc
Host: example.com
Accept: text/turtle
```

```
PREFIX : <http://ex.com/>
CONSTRUCT {?s ?p ?o}
WHERE { GRAPH :c {?s ?p ?o} }
```

```
PUT /gs?graph=http%3A%2F%2Fex.com%2Fc
Host: example.com
Content-Type: text/turtle
@prefix : <http://ex.com/>.
:j :hasName "John"@en.
```

```
PREFIX : <http://ex.com/>
DROP SILENT GRAPH :c
INSERT { GRAPH :c
{:j :hasName "John"@en}}
```

```
POST /gs?graph=http%3A%2F%2Fex.com%2Fc
Host: example.com
Content-Type: text/turtle
@prefix : <http://ex.com/>.
:j :hasName "John"@en.
```

```
PREFIX : <http://ex.com/>
INSERT { GRAPH :c
{:j :hasName "John"@en}}
```

```
DELETE /gs?graph=http%3A%2F%2Fex.com%2Fc
Host: example.com
```

```
PREFIX : <http://ex.com/>
DROP GRAPH :c
```

OpenRDF Sesame HTTP Protocol

```
<SESAME_URL>
/protocol : protocol version (GET)
/repositories : overview of available repositories (GET)
/<REP_ID> : query eval. and admin. tasks on a repo (GET/POST/DELETE)
  /statements : repository statements (GET/POST/PUT/DELETE)
  /contexts : context overview (GET)
  /size : #statements in repository (GET)
  /rdf-graphs : named graphs (NGs) overview (GET)
    /service : GS ops on indirectly ref. NGs (GET/PUT/POST/DELETE)
    /<NAME> : GS ops on directly ref. NGs (GET/PUT/POST/DELETE)
      /namespaces : overview of namespace definitions (GET/DELETE)
      /<PREFIX> : namespace-prefix definition (GET/PUT/DELETE)
```

0.1.4 Application Access to Ontologies

RDF access – status

- Most libraries are in Java (open-source), but many others appear as well in other languages, incl. python, .NET, or Ruby.

- open-world (ontologies) vs. closed-world (application data model)

Low-level APIs

OWLAPI (`http://owlapi.sourceforge.net`) – a de-facto standard API for accessing/parsing OWL 2 ontologies,

Jena (`http://jena.apache.org`) – complex RDF/SPARQL API; one of the most used ones

Sesame (`http://www.openrdf.org`)– RDF API for programmatic access to the Sesame RDF triple store.

... and other

Listing 3 : Example Jena code

```
Model m = ModelFactory.getModel("http://example.org/personal");
Resource i = m.getResource("http://example.org/person1");
i.addProperty(ResourceFactory.getProperty("http://example.org/hasName"),"John");
m.close();
```

High-level APIs

... are typically based on ORM

AliBaba (`http://www.openrdf.org`)– API for programmatic access to RDF datasets through Sesame

JAOB (`http://wiki.yoshtec.com/jaob`) – API for programmatic access to OWL ontologies

JOPA (`http://sourceforge.net/projects/jopa`) – API for programmatic access to OWL2-DL ontologies, with integrity constraint checking

Listing 4 : Example JOPA code

```
Person person1 = em.find("http://example.org/person1");
person1.setHasName("John"); \\ plus ORM for Person class
```

References

[1] Steve Harris, Nick Lamb, and Nigel Shadbolt. "N.: 4store: The Design and Implementation of a Clustered RDF Store". In: *In: Scalable Semantic Web Knowledge Base Systems - SSWS2009*. 2009, pp. 94–109.