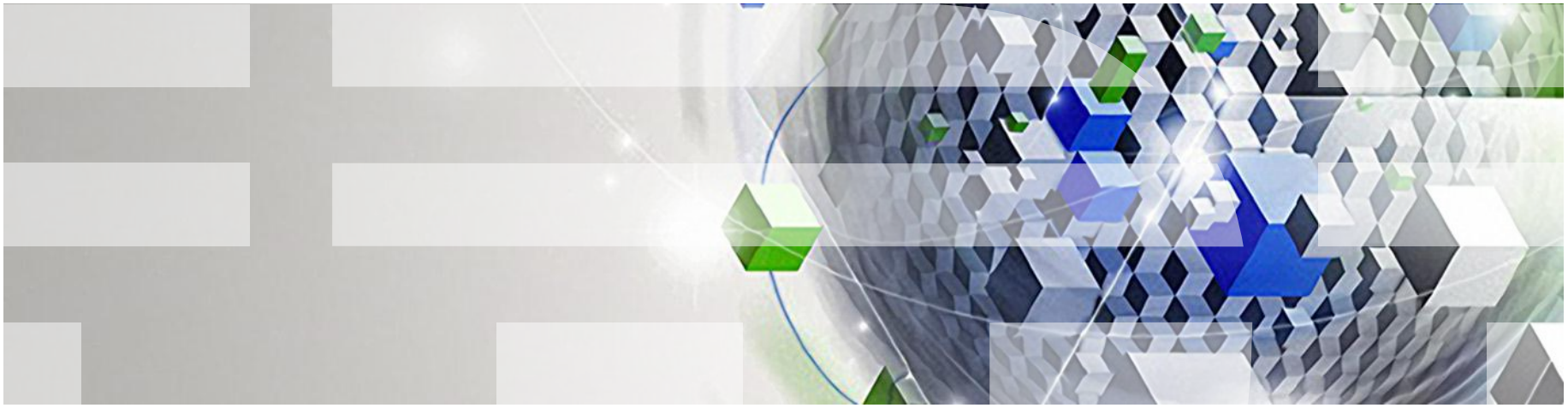
Information Management software

DB2 NoSQL Graph Store
What, Why & Overview

Mario Briggs, March 28 2012



Agenda

- Graph Stores
- What is RDF
- Why RDF in DB2
- Details of RDF Support in DB2

Graph Stores

- Different Graph Store products exist in the market today.
  - Some provide custom API's & Query Languages
  - Many support the W3C's RDF standard. (e.g. Jena, OpenSesame, Virtuoso, AllegroGraph etc)
- In DB2 we support the RDF standard.

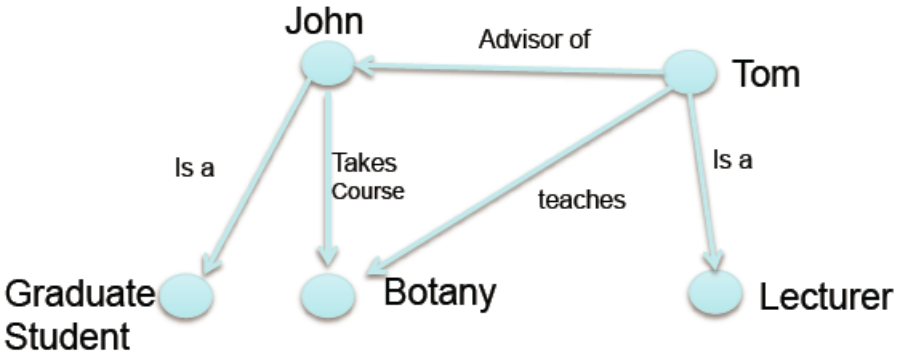
What is RDF

- RDF provides a general method to decompose any information into pieces called triples.
  - Each triple is of the form ‘Subject’ , ‘Predicate’, ‘Object’.
  - Subject and Object is the names for 2 things in the world. Predicate, the relationship between them.

Subject	Predicate	Object
Tom	is a	Lecturer
Tom	teaches	Botany

  - Subject, Predicate, Object are given as URI’s (stand-ins for things in the real world)
  - Object can additionally be raw text

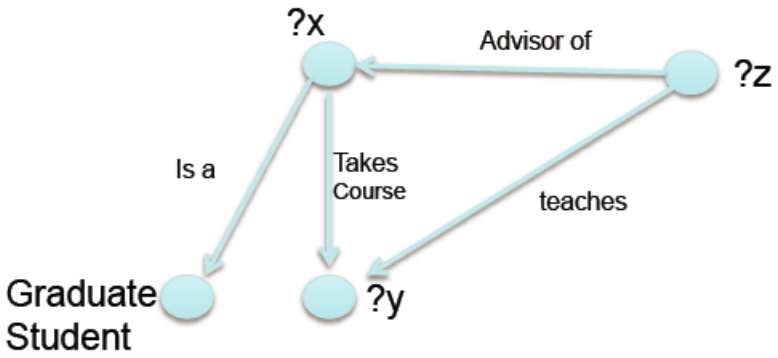
In technical terms a labeled directed graph, where each edge is a triple.



What is SPARQL

- SPARQL : A subgraph pattern matching query language.

  Find me all Graduate students who take a course taught by their advisor



SPARQL:

```
"SELECT ?gradStudent, ?course, ?advisor WHERE  {
            ?gradStudent <isA> <GraduateStudent>
            ?gradStudent <takesCourse> <?course>
            ?advisor     <AdvisorOf>    <?gradStudent>
            ?advisor     <teaches>      <?course>
    }
```

Result :

?gradStudent – John
?course – Botany
?advisor - Tom

What is RDF : Further reading

http://www.rdfabout.com/intro/

Why RDF in DB2

- Seen many client projects needing to
  - Integrate multiple independent schemas. Each evolves separately
  - Bridge the structured and unstructured worlds
  - Queries that integrate across data are complex. Searches need to follow links between the data elements across schemas to find relationships.

  While we built customized solutions for these in the past, well accepted standards have emerged from w3c in this area

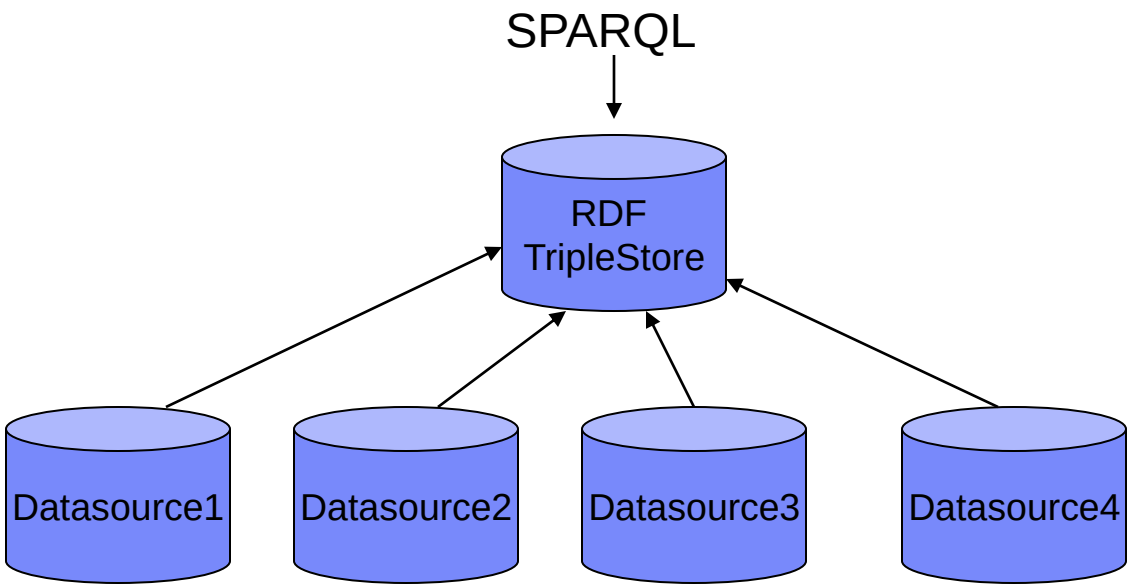
- Some projects did use RDF with open-source triple-stores
  - Face scalability issues with regards to transactions, concurrency and isolation.

Ride on top of DB2' existing enterprise capabilities instead of reinventing the wheel.
  - ACID, Security, Backup/recovery, compression, load balancing & parallel execution.

Application Scenarios

- Loosely integrating multiple data silos using common terms.
  Software life-cycle tools collaboration - OSLC http://open-services.net/
  NASA's Expertise Location Service - http://tinyurl.com/dc3a4n
  Pfizer's Idea Project - http://tinyurl.com/8ax8vsn

- Linking structured and unstructured data - http://tinyurl.com/7f7pdmj
  integrate internal product data with external websites data



Agenda

- Graph Stores
- What is RDF
- Why RDF in DB2
- Details of RDF Support in DB2
  - Objective, Usage Scenarios
  - SPARQL Support
  - Access Control / Security options
  - Default Stores vs Optimized Stores

DB2 RDF Objective

Ability for applications to store and query RDF data in DB2

How does the User consume ?

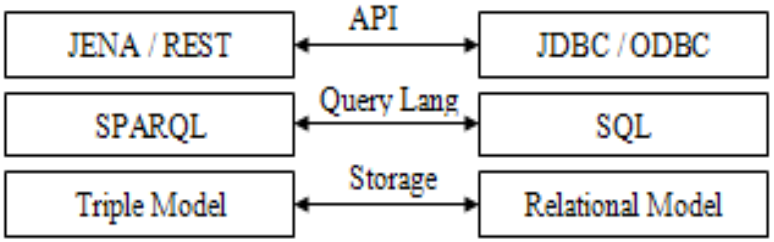
- In DB2 LUW 10.1 we support
  - Java API's for RDF application consumers.
  - HTTP based SPARQL query.

- DB2 RDF support is implemented in rdfstore.jar file that ships with all DB2 Clients

  - Additional jar file dependencies that are shipped with DB2 (wala.jar, antlr3.3.jar)
    These are located in sqllib/rdf/lib

  - Additional jar files dependencies that need to be downloaded by the user (JENA and ARQ jars)

- Place these jars on RDF application's classpath



Relational vs RDF - analogy

Scenario 1 : Creating a RDF Store / RDF Dataset in DB2

- All that is needed is a Connection to the DB2 database in which to create the RDF Store/Dataset.
  - Assign a unique name to the Store.

commandline e.g.

createrdfstore STORENAME -db DB2_DBNAME -user DB2User -password DB2Pass [-host DB2Server] [-port DB2Port] [-schema DB2Schema]

Java API e.g.

com.ibm.rdf.store.StoreManager.*createStore(java.sql.Connection, String db2Schema, String storeName)*

What does a DB2 RDF Store look like at the backend

- **A set of user tables within a database schema that stores the RDF data of the RDF dataset.**
  - Direct Primary : stores the RDF triples and the graph they belong to indexed by subject. Predicates and objects for a subject in a graph are stored in pair columns in this table. A particular predicate can occur in any one of three columns in this table.
  - Direct Secondary : stores the RDF objects that share the same subject and predicate within a RDF graph.
  - Reverse Primary : stores the RDF triples and the graph they belong to indexed by object. Predicates and subjects for a object in a graph are stored in pair columns in this table. A given predicate can occur in any one of three columns in this table
  - Reverse Secondary : stores the RDF subjects that share the same object and predicate within a RDF graph.
  - longStrings : stores RDF Literals and IRI's whose value exceeds a determined character length

Core intuition: **Optimized for graph pattern matching which involves a lot of 'star queries'** (known to be very common in RDF datasets).

What does a DB2 RDF Store look like at the backend

Direct Primary

Subject	Graph		predicate	value1		predicate	value4		predicate10	value10		predicate4	value40
s1			pA	vA					pB	vB		pC	lid:abc
s2						pF	vF		pB	lid:cdf			

Direct Secondary

Graph	listId	Value
	lid:abc	v1
	lid:abc	v2
	lid:cdf	v3
	lid:cdf	v4

Reverse Primary

Object	Graph		predicate	sub1		predicate	sub4		predicate10	sub10		predicate4	sub40
vA			pA	s1									
vF						pF	s2						

Reverse Secondary

Graph	listId	Value

Scenario 2 : Creating a RDF Store in DB2 from existing RDF Data

- Export the existing RDF data into a nquad formatted file.
- Use the BulkLoader provided – 'CreateRdfStoreAndLoader' command

commandline e.g.

java createrdfstoreandloader STORENAME -db DB2_DBNAME … -rdfdata NQUAD_Datafile –storeloadfile STORE_LOADFILE.SQL

The BulkLoader creates empty tables in the database and a set of DB2 LOAD files that can be used to load the existing RDF data in bulk to these tables. Run the STORE_LOADFILE.SQL at a DB2 command prompt to bulk load the RDF data into the tables.

Scenario 3 : Programatically inserting/updating data in the RDF Store

- We support the JENA API's for this.

```
//connect to the RDF Store
Store store = StoreManager.connectStore(conn, "foo");

//get the default model in the store
Model m = RdfStoreFactory.connectDefaultModel(store, conn);

// begin an explicit TXN
m.begin();

// Add the Data
String personURI = "http://somewhere/JohnSmith";
String fullName = "John Smith";
Resource johnSmith = m.createResource(personURI);
johnSmith.addProperty(VCARD.FN, fullName);

// Explicitly commit the TXN
m.commit();
```

* Higlighted classes are DB2 RDF Store specific Factory classes.

Programatically inserting/updating data in the RDF Store - Contd

- Adding a whole graph e.g.

```
// create an in-memory representation of the graph
Model memModel = ModelFactory.createDefaultModel();
for (int i = 0; i < 10; i++) {
    Node s = Node.createURI("sub:" + (i));
    Node p = Node.createLiteral("pre" + i);
    Node v = Node.createURI("Val:" + (i));
    Triple t = Triple.create(s, p, v);

    memModel.add(memModel.asStatement(t));
}

// get the Model/Graph to in DB2 Store in which to add to
Model db2model = RdfStoreFactory.connectNamedModel(store, conn,"Graph1");

// persist the memory graph to DB2
db2model.add(memModel);
```

Scenario 4 : Querying the RDF Store using SPARQL

- Again we support this through the JENA ARQ API's

```
//connect to the RDF Store
Store store = StoreManager.connectStore(conn, "foo");

// get the Dataset object
Dataset ds = RdfStoreFactory.connectDataset(store, conn);

// create the QueryExecution object
QueryExecution qe = RdfStoreQueryExecutionFactory.create(
                                "select ?who where { ?who ?p 'John Smith' }",
                                ds);

  // execute query and process resultset
  ResultSet rs = qe.execSelect();
  while (rs.hasNext()) {
      QuerySolution qs = rs.next();
      RDFNode sub = qs.get("who");
      …
  }
  qe.close();
```

Scenario 4 : Querying the RDF Store using SPARQL

- For HTTP based SPARQL query access we support using JOSEKI
  - setup JOSEKI in your Web/App Server
  - Configure JOSEKI to access DB2 RDF

SPARQLer Query Results

bookOrganization	totalPrice
<http://books.example/org1>	"21" ^^<http://www.w3.org/2001/XMLSchema#double>
<http://books.example/org2>	"7" ^^<http://www.w3.org/2001/XMLSchema#double>

Level of SPARQL supported by DB2

- **In DB2 LUW 10.1 we support**
  - All of SPARQL 1.0
  - The following from SPARQL 1.1

    STRSTARTS, STRENDS, IF,COALESCE, GROUP BY, HAVING, SUM, COUNT, AVG, MIN, MAX, SELECT Expressions , LET assignment and SubQueries

Agenda

- Objective
- Usage Scenarios
- SPARQL Support
- Access Control / Security options
- Default Stores vs Optimized Stores

Access Control / Security options

- **Coarse grained Access Control**
  - **Use** Table level permissions of DB2
- **Fine Grained Access Control**
  - We support more fine grained access control at an RDF Graph level.
  - Application up-ahead specifies which pre-defined RDF predicates are used for access control of RDF graphs.
  - 2 Flexible options available for enforcement

Enforced by the DBMS engine via DB2 FGAC. (Use existing DB supported features for defining and specifying the access control required). Use for more robust cases.

Enforced by the RdfStore Java layer. Additional metadata passed via the QueryExecution object for each SPARQL query executed/entire session. Limited support of operators and operands.

Agenda

- Objective
- Usage Scenarios
- SPARQL Support
- Access Control / Security options
- Default Stores vs Optimized Stores

Default Stores VS Optimized Stores

- Default Store
  - Use this when you have no existing RDF Data.
  - Default # of columns in primary tables. Picked on analysis of sample RDF datasets.
  - Uses hashing to determine which column a predicate gets stored in. If column is already occupied uses 3 hash functions to find a empty location. If none found, insert a new row.
- Optimized Store
  - Use this when you already have existing RDF Data.
  - We intelligently assign columns to predicates based on predicate co-occurrence for subject/object in the RDF data. E.g., age, and social security number co-occur as predicates of Person, and headquarters and revenue co-occur as predicates of Company, but age and revenue never occur together in any entity.
  - Exploits predicate co-occurrence and reduces number of rows in table, making joins more efficient.
  - Indexing predicates is simpler since a predicate can be confined to 1 column.
  - Multiple predicates can be assigned to a single column allowing a single DB2 index to index multiple predicates.
  - More positive effects on performance.

Default Stores VS Optimized Stores :

- How to choose between them
  - Default Store
    Use this when you have no existing RDF data on which to calculate predicate co-occurrence.
  - Optimized Store
    Use this when you have existing RDF data on which we can calculate predicate co-occurrence. (E.g. existing RDF application migrating from other stores, after System QA cycle)

- How to create them
  - Default Store
    Default options of ‘createRdfStore’ API creates a default store.
  - Optimized Store
    - BulkLoader (CreateRdfStoreAndLoader) automatically creates a Optimized Store.
    - Start with a default Store and post System QA cycle, use ‘GenPredicateMapping’ API to calculate predicate co-occurrence in the Store. Feed this output to ‘createRdfStore’ API to create Store for production.
    - Start with default Store and after sufficient representative data has been inserted, use ‘checkreorg’ and ‘reorg’ commands to reorganize to an optimized store

Thank You