SYTYCD - Team KoJAk

This repository holds the submission to So You Think You Can Demo for Team KoJAk, K=Ken Tune, J=Jochen Joerg and A=Adam Fowler. Please note that Adam Fowler started on 31 Aug 2012 and so qualifies as a Newbie.

Scenario

A Bank has just bought an Insurance Company. They want to repidly find how to save money. To answer the question of who their most expensive customers are they need to combine data in to Relational Databases - one from each company, each with different schema, and information embedded within Insurance Claims documents. Using MarkLogic 7's document database and triple store they find they can easily find this answer in just a few minutes.

Elevator pitch

MarkLogic helps organisations pull together information held in their relational databases, SharePoint repositories, and on the web, with minimal up front design work. This helps them respond quicker to new information, to exploit new information sources, and repurpose existing information assets. This greatly reduces integration time from months to hours, reducing costs and making organisations more effective and profitable in their core business. To prove we can deliver we have a quick five minute demonstration that does precisely this.

T-shirt sizes

One US Large (Adam), One Medium (Ken) and One Small (Jochen) please.

Demo video

This (very cheesy video) can be watched on YouTube at: http://youtu.be/4cUqMzsu0F4

How are ML 7 features used?

Triples allow storage of any facts. Tables, rows and columns are just facts about entities. So we represent data as triples with flexibility to spare. It is much easier now to ‘join’ the data – including the fact that the rdf representation of RDMBS data encodes foreign and primary key relationships making ‘natural’ joins natural in our environment. Inferencing to add facts does not alter the original data or relationships. The mapping we've used is the W3C RDB2RDF Direct Mapping, an open standard released as a recommendation in Sep 2012.

We’ve never had a way of generically pulling data out of an RDBMS and now we do. That on its own is pretty impressive in our view. The ability to join to other RDBMS schema, plus documents and other semantic resources is pretty mind blowing. Add to that the fact that Adam’s MLJS widgets make it a snap. (Adam: Thanks Ken!)

Note that the inferencing step could not make use of Sparql CONSTRUCT syntax because this generations bnodes (subject's without an IRI). This means that it is not possible to say 'Give me all the facts for Subject Y' with no prior knowledge of the subject. (I.e. some uniquely identifying property). This means the 'Entity Facts' widget could not load any Joint Customer's facts. Thus the inferencing step uses Sparql followed by a graph-insert, with a step in XQuery for generating unique Joint Customer names. Note that this is likely to be required in real life for such a situation anyway, as the combined company will want a unique reference creating for the Joint Customer record.

Also note that the REST endpoints (and specifically the W3C Sparql protocol and graph store API) do NOT provide the ability to pass a content search to restrict a sparql query. Hence the combining of semantic and content searches was done in JavaScript. There is no difference practically in doing this in JavaScript that using sem:sparql($mysparql, $myquery) in XQuery - except no REST Extension is required by doing this in JavaScript. Hence doing this in JavaScript for time's sake. This would be a good candidate for a REST endpoint though.

Demo Script

This is located in the repository in Word format at: https://github.com/mustard57/SYTYCD/blob/master/presentation/KojakDemoScript-AF.docx

Code Repository

We have a GitHub repository for all the code, reployable with Roxy, available at: https://github.com/mustard57/SYTYCD

Sales Background

There are many prospects that have a requirement for relational data migration and adhoc business user querying. Sparql and MarkLogic content search are a good fit for this requirement. This demonstration shows how you can quickly get answers by using an Enterprise NoSQL database. For a list of customers and total $ amount please contact Adam Fowler.

Data and software statement

No third party data or paid for software was used in the demonstration.

Re-usability

The vast majority of this demonstration can be re-used. The MLJS MarkLogic REST API toolkit and widgets were used to build the UI. MLSAM was used for integration to a relational database (MySQL). Below is a complete list of what is reusable:-

- MLJS Widgets - many exist, those shown include the rdb2rdf import wizard, sparql search bar, sparql results, entity facts, content search bar, content search results, and google Kratu (table data visualisation)
- Roxy was used as a web framework in hybrid mode so we could connect MLJS to the V7 REST API
- RDB2RDF Rest Extension for importing relational DB tables in to the triple store
- QConsole inferencing sample scripts and test search are available in a workspace in this GitHub repository
- Ken Tune created a database generation script to generate unique and related data for the two different database schemas. This will be very useful for future demonstrations where we are not permitted access to customer data.

The only code not completely re-usable was the JavaScript to configure how the widgets were initialised on the screen, and which entities and relationships were shown (as they are ontology specific), and the docsemlink widget which extracted the document URIs from the final matching doc list (created through using a combined semantic and content search) and generating SPARQL to fetch summary information on customers for use by the 'CEO'. In total only 370 lines of code are not reusable, but are usable as blog entries and demos. 12130 lines of code are re-usable. [1]

This means only 2.96 % of the code is NOT re-usable as-is. Happy days.

[1] Total combines the sql.xqy, rdb2rdf.xqy, rdb2rdf-lib.xqy, mljs.js, widgets.js, widgets.css, widget-triples.js, widget-kratu.js, widget-search.js, widget-rdb2rdf.js. This excludes their underlying dependencies.

Note that MLJS is fully documented on GitHub.

Extending this demonstration

In the future potential avenues of extension include:-

- Given more time we could create convincing documents that could be part of the demo.
 - Extend the data generation script to include sample document generation
 - Include a larger sample of documents linked to customers
 - Joining to external semantic resources.
- An intelligent sql like editor for queries to restrict what data to import. Do-able – we just didn't have the time. (Like the Sparql search widget, but for SQL)
- Use of 'not' queries – find me all customers with mortgages who have not got insurance. Find me all customers with insurance who have not got mortgages. (Adding a cross selling example rather than only cost reduction)
- Adding in binary data – a use case we considered was 'joining' non-compatible warehouse inventory systems – we could add in pictures of pallets of paving slabs or timber or cornflake boxes for an impressive demo.
- Adding in XML mixed with relational – no-one else can do tabular + XML – we would ideally showcase that.
- Fuzzy search – our matching query in the demo was for exact matches (people with same

name / birthday etc) but data is often not as clean as that. A demo involving fuzzy matching (allowing for casing / typos / whitespace differences) would be even more realistic.
- Expand to other verticals' use cases e.g. mocked up healthcare data (GP records and Hospital records). Requires time to create the scenario and data. KYC data – unifying data repositories.
- Other UI additions
 - HighCharts or network node widgets over triple search results
- Add a query that is mainly a content query, but with some aspects of triples (we did the opposite, concentrating on the triples but including basic document search. As we're all familiar with content search this seemed logical)
- Create a custom REST extension that allowed a combined structured query to restrict a sparql query, and a further sparql query to fetch related information based on the results of the first combined query. This currently is not possible in a single call in either the REST API or in sem:sparql, and is viewed by Adam Fowler as a restriction that needs addressing (after going through this demo exercise)
- Add support for generating an ontology description automatically to the rdb2rdf rest extension as an output of the import operation
- Amalgamate data from document range indexes, sparql result triples, and live data from an external RDBMS (i.e. no import step needed) in to a combined customer view information page (dashboard/explore)
- Take the start of the ontology we developed for document relationships and extend that for common MarkLogic modelling needs with semantic data (mentioned-in, derived-from, and other relationships/facts about the documents - reviewer, originator, project etc.)

Open Standards

In addition to the usual MarkLogic open standards, this demo made use of Sparql, REST, The W3C's RDB2RDF direct mapping for converting relational schema to triples, ANSI SQL and the Information Schema standards for accessing SQL table structure and relationship information.

Open Source software used

960.css and bootstrap.css were used to control the layout. Roxy was used as the configuration and deployment mechanism for MarkLogic. Google Kratu was used as a tabular visualisation of triple data (sparql results). MLJS was used for abstracting the REST calls and to provide visualisation widgets. This is licensed, like Roxy, under the Apache 2 license. MySQL was used

as the relational database. Tomcat was used to host the MLSAM instance for database communication. MLSAM's sql.xqy extension was used to communicate to MLSAM. ECMAScript 3 (JavaScript) and CSS 2 and 3 were used for the UI.

No proprietary software or platform specific hacks were used for this demonstration, maximising re-usability.

Running the Demo

Note that the demo is accessible on this server from a web browser: http://kojak.demo.marklogic.com:8040/

Setting up the demo

Instructions for this can be found at: https://github.com/mustard57/SYTYCD/blob/master/README.md