Exploring Large RDF Datasets using a Faceted Search

Juan Francisco Garcia Navarro, Victor Antonio Lopez Villamar
Jagannathan Srinivasan, Matthew Perry, Souripriya Das, Zhe Wu

Oracle Corporation

{juan.f.garcia, victor.a.lopez, jagannathan.srinivasan}@oracle.com
{ matthew.perry, souripriya.das, alan.wu}@oracle.com

Abstract. We propose a facet-based RDF data exploration mechanism that lets the user visualize large RDF datasets by successively refining a query. The novel aspects of our work are: i) the SPARQL query pattern is visualized as a *query graph*, ii) the successive refinements are visualized in a *query refinement graph*, and iii) the result triples are visualized as *a result RDF graph*. The scheme is *scalable* and it visualizes RDF graphs stored in Oracle Database 12c Spatial and Graph Option with Cytoscape, a graph visualization tool.

1 Introduction

For large RDF datasets, we propose a facet-based RDF data exploration mechanism that lets the user visualize data through a query refinement process. To describe our proposal, we use the GovTrack RDF data [1].The user starts with a conjunctive SPARQL triple pattern[1] (Fig. 1a), for example to get bill and voting information for sessions of the U.S Congress. From this SPARQL triple pattern, a *query graph* is formed as a directed-graph (Fig. 1b), where the subject and the objects (including variables, IRIs, or literals) are represented as nodes, and the predicates (both IRIs and variables) are represented as edges. Thus, the seven triple patterns would result in a directed graph with 6 nodes and 7 edges (the pre-recorded viewlet available at [6]).

The starting query provides the most general form (shown in Fig. 1a and 1b). We treat each variable in the query as a facet, which can have many possible values. Thus, subsequent refined queries are derived by replacing the variable in the SPARQL query with the value selected from the corresponding facet. The refined query can be executed on demand to get the resulting RDF triples. This process can be repeated until all variables are substituted. Furthermore, a *context menu* on each node (Fig. 1c) allows the user to list and choose among the possible values for the selected variable. This list also shows the number of solutions that would result after selecting a specific value thereby allowing the user to determine how much the result space will be reduced before performing the substitution.

[1] In general, our scheme is applicable to an arbitrarily complex SPARQL query since a query can be represented as a directed graph using its *abstract syntax tree*.

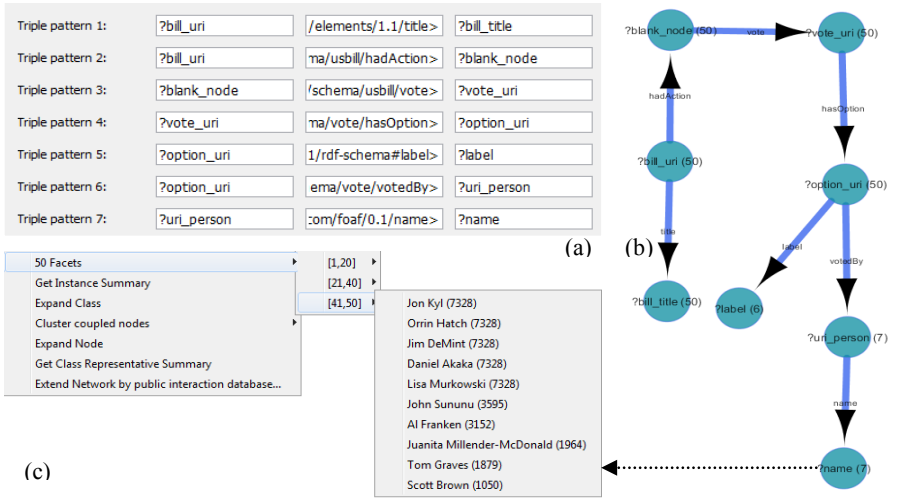

Fig. 1. a) The SPARQL query triple pattern b) the query graph, and c) a context menu

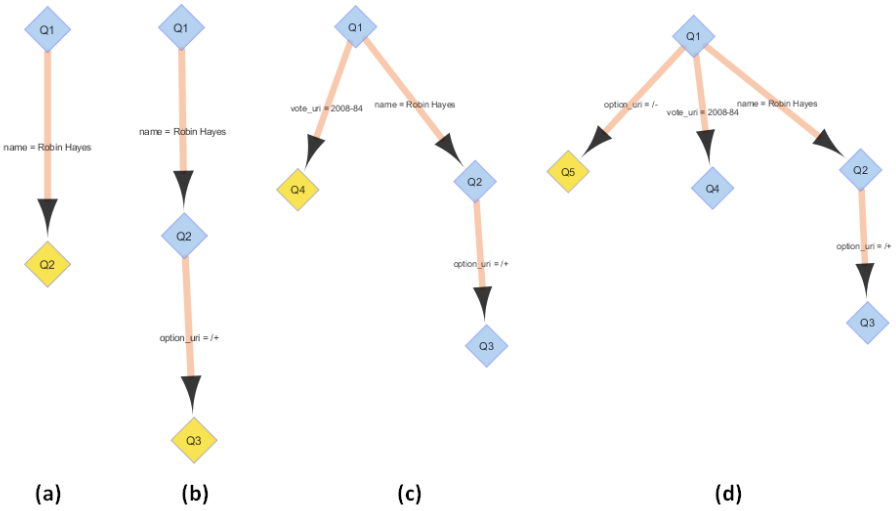

Fig. 2. Query Refinement graph (node: the original/refined query; edge: the substitution)

To capture the substitution steps, a *query refinement graph* is created. This graph is populated with nodes representing each of the refined queries that are being created. Figures 2a-2d show the different branches of the query graph, where each of the nodes represent a specific query with the facet values already replaced. Note that in addition to refining the last resulting query, which would represent a leaf node, the refinement can as well proceed from the root or any intermediate node.

Figure 2a shows the creation of the new node Q2, resulting from substituting the variable `?name` with the value `"Robin Hayes"`. Figure 2b shows the new query Q3 resulting from replacing variable `?option_uri` in query Q2. The user can further explore from the root node (Q1) to create another branch (node Q4 in Figure 2c) by replacing a different variable. Similarly, the user can explore from the root node (Q1) and create a new branch (node Q5 shown in Figure 2d).

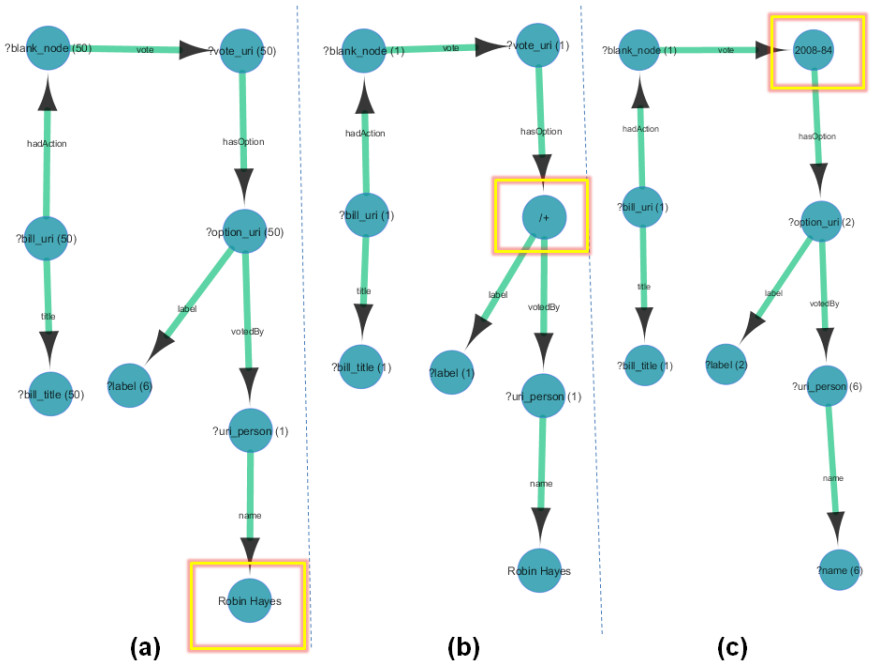

Fig. 3. The sequence of modified query graphs on substitutions

Once a substitution is made, and a new node is added into the query-refinement graph, the query- graph (shown in Fig. 1b) is also updated. The node counts are updated with the modified possible values on the nodes, according to the replaced variable. This sequence is illustrated in Figures 3a, 3b, and 3c, which correspond to their counterpart steps of the sequence shown on Figures 2a, 2b, and 2c.

At this point, we have applied the concept of facets to refine a SPARQL query successively. The result is a set of sub-queries, identified in Figures 2a-2d that can be independently executed giving a smaller graph that users may find easier to visualize and analyze, compared to the starting query. Fig. 4 shows the result of executing query Q5, depicted on Figure 2d, resulting in a graph having 270 nodes and 269 edges, with a total of 132 sub-graphs that match as result of the SPARQL query. Note that queries are executed as SPARQL CONSTRUCT WHERE queries to return a graph rather than bindings. Analogously, each of the nodes depicted in Figures 2a, 2b and 2c and labeled as Q2, Q3, and Q4, which represent sub-queries derived from the original query, can also be executed.

This way of successive refinement helps users explore the query solution space in an incremental manner. At each step, the facet counts for remaining variables give an indication of the solution space, prior to the actual visualization of any of the queries.

The above scheme is used to visualize RDF graphs stored in Oracle Database 12c Spatial and Graph Option [2] with Cytoscape [3], a graph visualization tool. The idea of hierarchical faceted navigation has been presented as early as 2002 in [4]. In [5], a combination of graph visualization and facet based filtering is used. However, our scheme is *scalable*. It requires materialization of only the first result set in a compact integer id-based format, on which facet counts are computed using full-scans or with bitmap index scans. We leverage Oracle's Parallel DML and query, and In-Memory capabilities to achieve interactive response times (Fig 4). For extremely large datasets (over billions of triples), sampling is used to limit the initial materialized result size.

Sample Performance on 8-core AMD FX-8350 CPU, 32 GB of RAM, Two 10000 rpm SATA disks, Oracle Enterprise Linux 5				
#vars (of 9) substituted	**Materialize Results**	**Facet Computation**		
		No Index	**Bitmap Index**	**In-Memory**
0	35s (18M rows)	87s	3.73s	***3.90s***
1	N/A (11M rows)	85s	38.00s	***2.13s***
2	N/A (0.2M rows)	85s	0.59s	***0.90s***

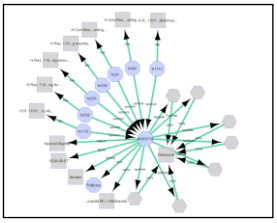

Fig. 4. Sample Performance Results and the Result Graph of a refined query.

2 Conclusions

We described a facet-based approach to effectively explore large RDF datasets. Our proposal includes a visual graph-based representation in order to make the querying process easier. The solution described includes creation of a query graph to represent the original SPARQL query and query refinement graph to keep track of what has been explored. The query refinement graph presents a hierarchy of sub-queries that are derived from the original query. Furthermore, each node in the query refinement graph, which represents a sub-query, can be executed to generate a smaller result graph that is easier for the user to visualize, analyze and thus understand.

3 References

1. Tauberer, J.,GovTrack, http://www.xml.com/lpt/a/1643
2. Oracle DB 12c Spatial & Graph , http://www.oracle.com/us/products/database/038407.htm
3. Cytoscape: An Open Source Platform for Network Data, http://www.cytoscape.org/
4. Hearst, M. A., Elliott, A., English, J., Sinha, R. R.,Swearingen, K., Yee. K.: Finding the flow in web site search. Commun. ACM 45(9): 42-49 (2002) .
5. Heim, P., Ertl T., and Ziegler, J., Facet Graphs: Complex Semantic Querying Made Easy, ESWC 2010.
6. Exploring Large RDF Datasets using a Faceted search Viewlet, In http://download.oracle.com/otndocs/tech/semantic_web/viewlets/iswc2015_faceted_search.zip