Query Optimization in Semantic Web Repositories

Ralf Heese
AG Corporate Semantic Web
Freie Universität Berlin

On the basis of the Berlin SPARQL Benchmark researchers measured the query execution time of a set of SPARQL queries and their corresponding SQL queries. The results show that the SQL queries run faster on the relational database system than the SPARQL queries on RDF repositories (by factor 5 to 20). Considering these results we think that there is most likely room for improving the query processing of the RDF repositories.

In the dissertation we investigate approaches to improve the query processing of RDF repositories, such that the query execution times of these systems reache approximately the one of relational database systems. The focus of the work lies on three areas: physical management of the RDF data, their indexing, and query optimization.

Physical Management Several RDF repositories rely on relational technologies to manage and to query RDF data. They use different approaches to store the data in relational databases, e.g., triple relation, property relations, and vertical partitioning. Using a relational database the query engine has to join triples to create solutions for a query. These join operations increase the query execution time – although the query evaluation is supported by indexes.

Besides the relational approaches there exist RDF repositories that implement a native data management. Similar to these, we account for the graph based model of RDF and develop and investigate a native RDF repository. The main idea of our approach is to organize the data such that all triples having the same subject are stored on the same physical database page. Assuming that the query engine can decompose a query into starlike parts our approach will reduce the number of accessed database pages. Thus, it could help to decrease the overall query execution time. Furthermore, all triples belonging to a given subject can easily be located and retrieved due to the data organization. This property could improve the evaluation of path like queries. In the future we will also research other clustering strategies, e.g., exploit domain knowledge and store semantically related triples on the same page.

Indexing Current RDF repositories usually create indexes on one or more components of an RDF triple. In the dissertation we propose an approach that is based on the materialization of basic graph pattern. In the following we refer to this materialization as index and to the basic graph pattern as index pattern. The main idea of our approach is to create (manually) several indexes on the

RDF data. The indexes are used to solutions for some parts of a query and thus, reduce the number of joins. Since the index patterns may overlap, solutions of a greater part of a query may be computed efficiently, e.g., by joining indexes instead of triples. The query processing works as follows: The query engine selects a set of indexes from the set of all defined indexes which are eligible for the query at hand, e.g., the index patterns are contained in the query pattern. Based on a cost function the query engine chooses a set of eligible indexes that are likely to reduce the query execution time. Finally, a solution of the part covered by an index is extended to a solution of the complete query by accessing the RDF data.

Query Optimization In the third part of the dissertation we research the optimization of processing SPARQL queries. In this context we focus on the *query rewriting* phase. As the basis for the query rewriting we adapted the query graph model used in the relational database management system Starburst to represent SPARQL queries. A SPARQL query is translated to a query graph during parsing and is enriched with information related to query processing. On top of the SPARQL query graph model we defined transformation rules which transform a query into a semantically equivalent one. The goal of these rules is to construct a query execution plan which can be evaluated more efficient than processing the original query, e.g., by changing the evaluation order of basic pattern or by using indexes.

Finally, we integrate the three described approaches into an RDF repository. The query optimizer transforms the query such that the query execution engine can exploit indexes and the characteristics of the native storage system, e.g., the triple pattern are grouped according to their subject or to match an index pattern. It could also exploit the properties of the native storage to create a query execution plan that efficiently extends partial solution (e.g., retrieved from an index) to a solution of the complete query pattern.