QueRIE: A Query Recommender System Supporting Interactive Database Exploration

Conference Paper · December 2010

DOI: 10.1109/ICDMW.2010.43 · Source: DBLP

5 authors, including:

Sarika Mittal
University of Petroleum & Energy Studies
6 PUBLICATIONS **82** CITATIONS

SEE PROFILE

Magdalini Eirinaki
San Jose State University
72 PUBLICATIONS **2,641** CITATIONS

SEE PROFILE

QueRIE: A Query Recommender System supporting Interactive Database Exploration

Sarika Mittal, Jothi Swarubini Vindhiya Varman, Magdalini Eirinaki
Computer Engineering Dept.
San Jose State Univ.

Gloria Chatzopoulou*
Computer Science Dept.
Univ. of California, Santa Cruz

Neoklis Polyzotis
Computer Science Dept.
Univ. of California, Santa Cruz

Abstract

This demonstration presents QueRIE, a recommender system that supports interactive database exploration. This system aims at assisting non-expert users of scientific databases by generating personalized query recommendations. Drawing inspiration from Web recommender systems, QueRIE tracks the querying behavior of each user and identifies potentially "interesting" parts of the database related to the corresponding data analysis task by locating those database parts that were accessed by similar users in the past. It then generates and recommends the queries that cover those parts to the user.

Keywords: recommender systems, collaborative filtering, relational databases, interactive exploration

1 Introduction

Relational database systems are becoming increasingly popular in the scientific community. For example the Genome browser[1] provides access to a genomic database, and SkyServer[2] stores large volumes of astronomical measurements. Those databases usually employ a web-based interface that allows a broad user base to submit SQL queries and retrieve the results. Even though such database systems offer the means to run complex queries over large data sets, the discovery of useful information remains a big challenge. As an example, users who are not familiar with the database may overlook queries that retrieve interesting data, or they may not know what parts of the database provide useful information. Moreover, most of the users of such databases do not have the required SQL background that would allow them to run complex queries and retrieve otherwise interesting results. Not being able to fully exploit the capabilities of such systems hinders data exploration, and thus reduces the benefits of using a database system.

To address this important problem of assisting users when exploring a database, we designed the QueRIE (Query Recommendations for Interactive data Exploration) system. Our inspiration draws from the successful application of recommender systems in the exploration of Web data. The premise on which the system is built is simple: If a user A has similar querying behavior to user B, then they are likely interested in retrieving the same data. Hence, the queries of user B can serve as a guide for user A.

Transferring the collaborative filtering paradigm to the database context presents several challenges. First, SQL is a declarative language, thus the same data can be retrieved in more than one way. This complicates the evaluation of similarity among users, since contrary to the web paradigm where the similarity between two users can be expressed as the similarity between the items they visit/rate/purchase, we cannot rely directly on the SQL queries. A second important issue is how to create implicit user profiles that measure the level of importance of the same data for different users. Finally, contrary to the user-based collaborative filtering approach, the recommendations to the users have to be in the form of SQL queries, since those actually describe what the retrieved data represent. Thus, we need to "close the loop" by first decomposing the user queries into lower-level components in order to compute similarities and make predictions, and then re-construct them back to meaningful and intuitive SQL queries in order to recommend them.

Contrary to keyword-based query recommendation systems [5], QueRIE is meant to assist users who need to pose complex SQL queries to large relational databases. Moreover, QueRIE does not require from the users to explicitly

*This work was performed while the author was affiliated with UC Santa Cruz

[1] http://genome.ucsc.edu/

[2] http://cas.sdss.org/

declare their preferences beforehand in order to generate recommendations. A multidimensional query recommendation system is proposed in [3]. In this work the authors address the related problem of generating recommendations for data warehouses. The challenges of applying data mining techniques to the database query logs are addressed in [4]. In this work, the authors outline the architecture of a query management system and propose that data mining techniques can be used in order to provide the users with query suggestions. Contrary to our work the authors do not provide any technical details on how such a recommendation system could be implemented.

In this demonstration we will showcase the QueRIE system. The demonstrated system will enable users to query the SkyServer database, obtaining real-time personalized query recommendations that are generated using user traces collected from the SkyServer query log. Apart from the graphical user interface of the system we will also demonstrate the underlying functionality of the system. A video demonstrating the system can be downloaded from: `http://www.engr.sjsu.edu/meirinaki/ICDM_Demo.m4v`.

2 System Overview

The information flow of the QueRIE personalization system is shown in Figure 1. The active user's queries are forwarded through the *database query interface* to both the DBMS and the *recommendation engine*. The DBMS processes each query and returns a set of results. At the same time, the query is stored in the query log. This query log is processed offline in order to create the predictive model. Each time a user accesses the system, the recommendation engine combines her input with the predictive model and generates a set of query recommendations. In what follows, we provide a very brief overview of the QueRIE framework and the underlying algorithm, since the demonstration will follow the same information flow. A detailed technical description of the proposed framework can be found at [1].

The queries of each user touch a subset of the database that is relevant to the analysis the user wants to perform. We assume that this subset is modeled as a *session summary* S_i for user i. We use $\{1, \ldots, h\}$ to denote the set of past users based on which recommendations are generated and 0 to identify the current user. To generate recommendations, our framework extends the summary S_0 of the active user to a "predicted" summary S_0^{pred}. This extended summary captures the predicted degree of interest of the active user with respect to all the parts of the database, including those that the user has not explored yet, and thus serves as the seed for the generation of recommendations.

To summarize, our framework consists of three components: (a) the construction of a session summary S_i for each user i, (b) the computation of a "predicted" summary S_0^{pred} for the active user, based on the active user's and the past users' summaries, and (c) the generation of queries based on S_0^{pred}. Those queries will be presented to the user as recommendations.

Session Summaries. We define the session summary S_i as a vector of tuple weights that covers all the database tuples. The weight of each vector element represents the importance of the respective tuple in the exploration performed by user i. For this purpose we employ two different weighting schemes which are detailed in the accompanying paper [1]. Using the session summaries of the past users, we can define the conceptual *session-tuple matrix* that, as in the case of the user-item matrix in web recommender systems, will be used as input in our collaborative filtering process.

Computing the "Predicted" Summary. Similarly to session summaries, the extended summary S_0^{pred} is a vector of tuple weights. In order to compute this summary, we assume the existence of a function $sim(S_i, S_j)$ that measures the similarity between two summaries and takes values in $[0, 1]$. Using this function, we compute the extended summary as a weighted sum of the existing summaries:

$$S_0^{\text{pred}} = \sum_{0 \leq i \leq h} (sim(S_0, S_i) \times S_i)$$

The similarity function sim can be realized with any vector-based metric, such as the cosine similarity measure.

Generating Recommendations. The final step is to generate queries that cover the interesting tuples in S_0^{pred}. Since our overall objective is to provide the users with intuitive, easily understandable recommendations, we do not try to construct synthetic queries using S_0^{pred}. Instead, we use the queries of past users. This results in a recommendation set that includes understandable (human-edited) queries. We assign to each past query Q an importance with respect to S_0^{pred} , computed as $rank(Q, S_0^{\text{pred}}) = sim(S_Q, S_0^{\text{pred}})$. Hence, a query has high rank if it covers the important tuples in S_0^{pred}. The top ranked queries are then returned as the recommendation.

Accelerating the online computations. To ensure that the aforementioned approach generates real-time recommendations for the active users of a database, we need to compress the session-tuple matrix and to speed up the computation of similarities. For this reason, we employ the MinHash probabilistic clustering technique that maps each session summary S_i to a "signature" $h(S_i)$ [2]. The Jaccard similarity between vectors is thus reduced to the similarity of their signatures: $JaccardSim(S_i, S_0) = sim(h(S_i), h(S_0))$.

3 Demonstration Scenario

As shown in Figure1, QueRIE consists of two main building blocks, namely the database query interface and

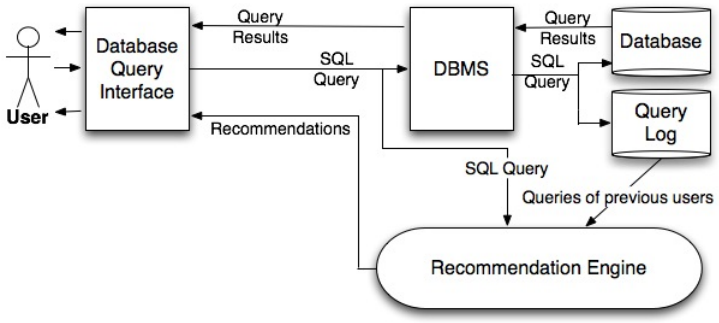

Figure 1. System Architecture

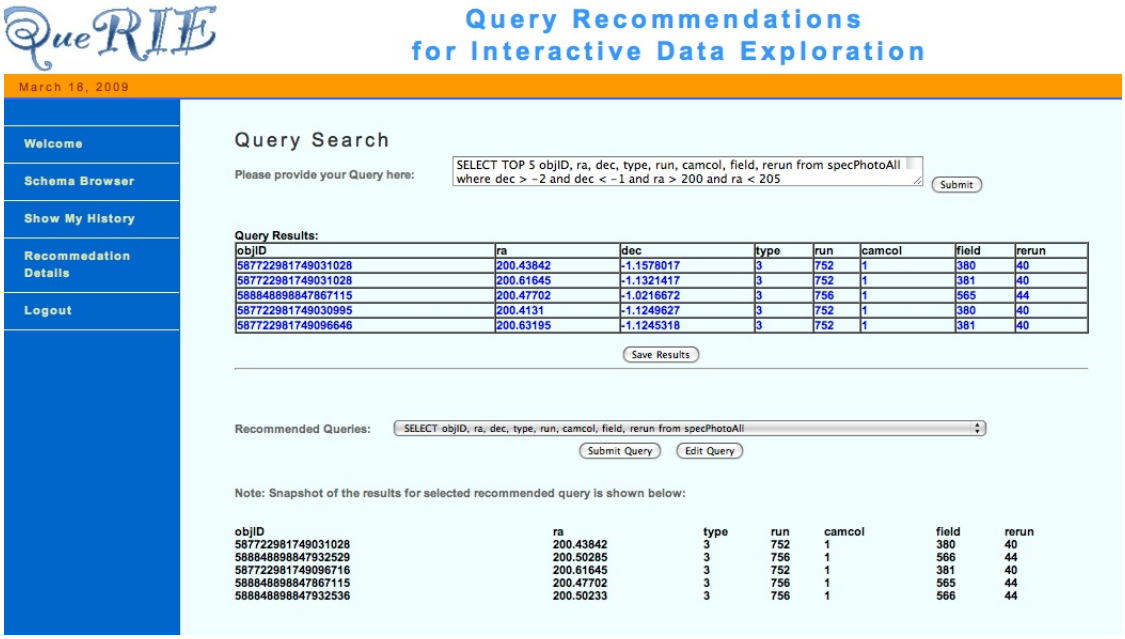

Figure 2. QueRIE interface after a query has been submitted

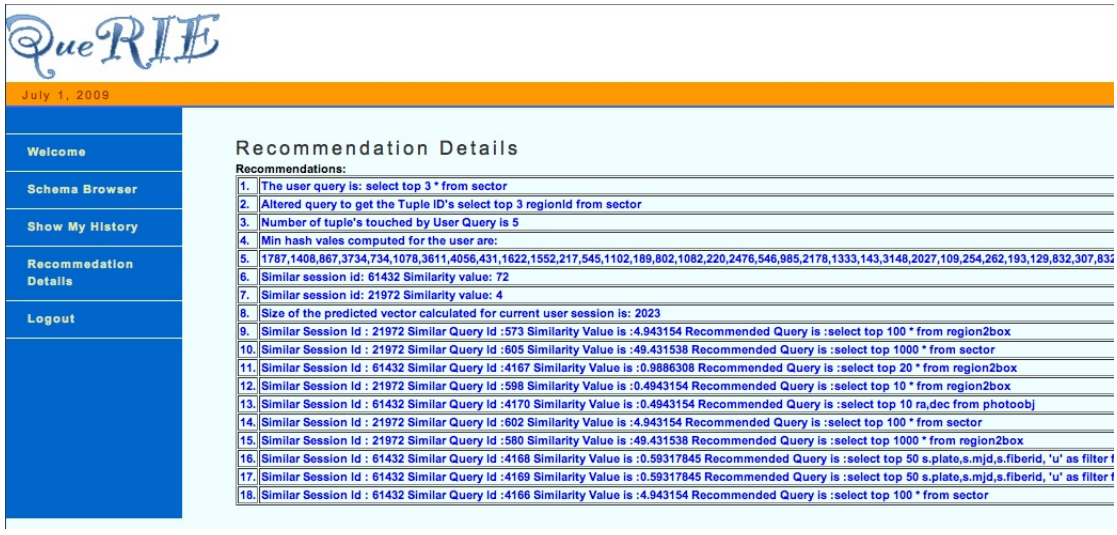

Figure 3. QueRIE back-end console

the recommendation engine, and uses two information repositories, namely the database itself, as well as its query logs. The database query interface module is built using HTML, JSP and JavaScript. The recommendation engine module is built using C++. The two modules interact through the JNI framework. The demonstrated system interacts with the SkyServer database and uses real user traces for the query logs of past users.

The demonstration shows the functionality of the system and the internal operations of the recommendation engine. In that way, we are able to demonstrate the usefulness of such a system from the user's perspective, and at the same time allow someone to understand and evaluate how the various data inputs are manipulated in order to generate the final recommendations.

Once a user logs in the system, she is able to write and submit an SQL query to SkyServer. QueRIE sends the request to the database, and presents the user with the results. At the same time, the system records the user's queries, creating an implicit user profile. This user profile is used as input to the recommendation engine, along with the predictive model to generate real-time query recommendations for the active user. The system generates different query recommendations for different users, depending on which parts of the database they are accessing.

The recommended queries are presented to the user in a drop-down list, as shown in Figure 2. For each recommended query, the user is able to examine a sample of the results that will be retrieved, in order to decide whether it addresses his information needs, prior to actually submitting it to the DBMS.

At all times, the active user is able to: (a) formulate a query from scratch, (b) select a recommended query and submit it as it is, or (c) select a recommended query and edit it before submitting it to the database. All the aforementioned options result in an SQL query being submitted to the SkyServer database, which is handled by the system as described before. Moreover, the interface allows the user to browse the database schema in order to find more information on the tables and the respective attributes and formulate the queries. The user is also able to review and resubmit queries that were posed during her recent history. Those options are available through the "Schema Browser" and "Show My History" menu tabs respectively.

Apart from demonstrating the user interface of our system, we also incorporate a "back-end" console that enables users to observe and understand how the underlying algorithms work, as well as evaluate the usefulness of the recommendations. More specifically, for each query submitted to the system, the console shows details related to the representation of the session summary as a MinHash signature, the session ids of similar sessions, as well as the most similar queries included in them, along with the respective similarity values. A snapshot of this interface is shown in 3.

In order to evaluate the usefulness of the system and the accuracy of the recommendations, we follow the "recommended vs. actual queries" approach. More specifically, we will have some "demo" users, for which at least one session of $k > 3$ queries is stored in the query logs. After submitting the first $k - n$ queries to the system, the users will be able to see both QueRIE's recommendations and the actual n last queries of the user side by side, and compare them.

4 Conclusions

We have described the functionalities we demonstrate for QueRIE, a recommender system that assists users when interacting with large database systems. QueRIE enables users to query a relational database, while generating real-time personalized query recommendations for them. Currently QueRIE interacts with the SkyServer database but it is evident that the system can be easily adapted to interact with any relational database given that its query logs are available for analysis. QueRIE does not require an explicit user profile or keyword-based queries. On the contrary, it "closes the loop" by accepting SQL queries as input, decomposing them in order to identify interesting database areas for each user, and re-transforms them in SQL queries that are presented as recommendations.

We should stress that this is a first-cut solution to the very interesting problem of interactive database exploration. There are many open issues that need to be addressed. For instance, an interesting problem is that of identifying "similar" queries in terms of their structure and not the tuples they retrieve. Two queries might be semantically similar but retrieve different results due to some filtering conditions. Such queries need to be considered in the recommendation process. We are currently working on extending our framework to cover such query similarities.

References

[1] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Collaborative filtering for interactive database exploration. In *SSDBM '09*, 2009.

[2] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55:441–453, 1997.

[3] A. Giacometti, P. Marcel, and E. Negre. Recommending Multidimensional Queries. In *DaWaK'09*, September 2009.

[4] N. Khoussainova, M. Balazinska, W. Gatterbauer, Y. Kwon, and D. Suciu. A case for a collaborative query management system. In *CIDR '09*, 2009.

[5] G. Koutrika and Y. Ioannidis. Personalized queries under a generalized preference model. In *ICDE '05*, 2005.