QuRIE: Collaborative Database Exploration

[1]Prof.Shinde J.P, [2]Sanap Vishal,[3]Nikhil Bagati
[1]Assit.Professor,[2]Student,[3]Student
[1]Information Technology,
[1]P.R.E.C Loni, Maharashtra, India

***Abstract* - Relational database systems are becoming increasingly popular in the scientific community to support the interactive exploration of large volumes of data. In this scenario, users employ a query inter-face (typically, a web-based client) to issue a series of SQL queries that aim to analyze the data and mine it for interesting information. First-time users, however, may not have the necessary knowledge to know where to start their exploration. Other times, users may simply overlook queries that retrieve important information. In this work we describe a framework to assist non-expert users by providing personalized query recommendations. The querying behavior of the active user is represented by a set of query fragments, which are then used to identify similar query fragments in the recorded sessions of other users. The identified fragments are then trans-formed to interesting queries that are recommended to the active user. An experimental evaluation using real user traces shows that the generated recommendations can achieve high accuracy.**

***Keywords* - Recommender Systems, Collaborative Filtering, Relational Databases, Interactive Exploration.**

I. INTRODUCTION

Relational database systems are becoming increasingly popular in the scientific community in order to provide access to large volumes of scientific data. Examples include the Genome browser[1] that hosts a genomic database, and SkyServer[2] that stores large volumes of astronomical measurements. Scientific databases are usually accessed through a web-based interface that allows users to submit SQL queries and retrieve the results. Even though users have the ability to issue complex queries over large data sets, the task of knowledge discovery remains a big challenge. Users may not know which parts of the database hold useful information, may overlook queries that retrieve relevant data, or might not have the required expertise to formulate such queries. Moreover, because of the continuously increasing size of the database, an extensive exploration of the whole database is usually very time-consuming. These factors clearly hinder data exploration and limit the benefits of using a relational database system.To address the important problem of assisting users when exploring a database, we designed the QueRIE framework (Query Recommendations for Interactive data Exploration). QueRIE assists users by generating dynamic, personalized query recommendations in ad-hoc or form-based query environments. The idea is to provide the user with a set of SQL queries that are expected to be relevant to their information needs. The user will be able to directly submit or further refine these queries, instead of having to compose new ones.

QueRIE is built on a simple premise that is inspired by Web recommender systems: 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. Collaborative filtering is a well known, mature technique for realizing this idea that we can borrow from Web recommender systems, but its application to database queries presents several challenges. First, SQL is a declarative language, and hence syntactically different queries may retrieve the same data. 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 can-not rely directly on the SQL queries. A second important challenge is how to assign importance to the data retrieved by a user's queries, since we cannot assume an explicit rating system as in the case of the Web. Finally, the recommendations to the users have to be in the form of SQL queries, since recommending specific data items may not be very intuitive. Thus, we need to "close the loop" by first decomposing the user queries into lower-level elements in order to compute similarities and make predictions, and then map the recommended elements back to meaningful and intuitive SQL queries that users can understand or refine.

In our previous work [2, 9], we presented the QueRIE architecture, framework, and the application of user-based collaborative filtering using witness tuples to represent user queries. In these papers, we propose an item-based approach that uses query fragments to represent the user queries. The recorded fragments are used to identify similar query fragments in the previously recorded sessions, which are in turn "assembled" in potentially interesting queries for the active user. We show through experimentation that the proposed method generates meaningful recommendations on real-life traces from the Sky Server database. The rest of the paper is organized as follows: in Section 2 we review related research performed in the area of query recommendations for relational databases; in Section 3 we provide a brief overview of the QueRIE conceptual framework; in Sections 4 and 5 we present the proposed fragment-based instantiation of the conceptual framework, along with some specific implementation de-tails concerning the queries' preprocessing; Section 6 includes some experimental results that evaluate several parameters of our frame-work and Section 7 concludes the paper with our plans for future work.

II.RELATED WORK

2.1 Hive - A petabyte scale data warehouse using HADOOP *(A. Thusoo et al.)*

This paper says that HADOOP is a popular open-source mapreduce implementation which is being used in companies like Yahoo, Facebook etc. to store and process extremely large data sets on commodity hardware. However, the map-reduce programming model is very low level and requires developers to write custom programs which are hard to maintain and reuse.Hive, an open-source data warehousing solution built on top of HADOOP. Hive supports queries expressed in a SQL-like declarative language - HiveQL, which are compiled into mapreduce jobs that are executed using HADOOP.

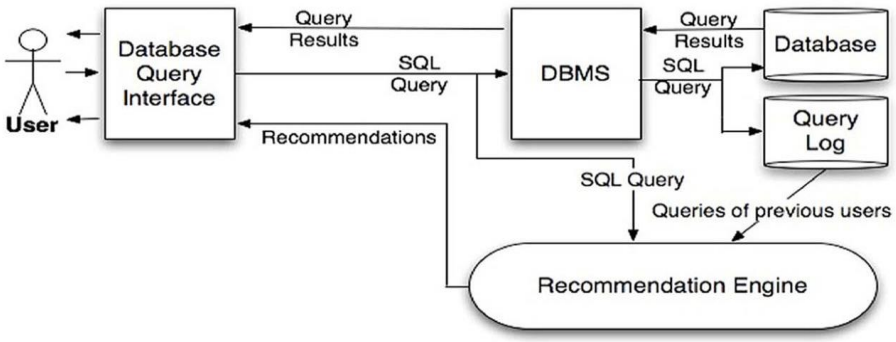
2.2 QueRIE: A recommender system supporting interactive database exploration *(S. Mittal, J. S. V.Varman)*
This paper mentioned that the demonstration presents QueRIE,a recommender system that supports interactive database exploration. This system aims at assisting non-expert users of scientific databases 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.

2.3 Amazon.com recommendations: Item-to-item collaborative filtering (G. Linden, B. Smith, and J.York*)*
This paper wrote that recommendation algorithms are used to personalize the online store for each customer. The store radically changes based on customer interests, showing programming titles to a software engineer and baby toys to a new mother. There are three common approaches to solving the recommendation problem: traditional collaborative filtering, cluster models, and search-based methods. Here, it compares these methods with our algorithm, which is called item-to-item collaborative filtering. The algorithm produces recommendations in real-time, scales to massive data sets, and generates high quality recommendations.

2.4 Recommending Multidimensional Queries (A.Giacometti, P. Marcel, and E. Negre)
In this work, the authors propose a framework for generating OLAP query recommendations for the users of a datawarehouse. The techniques and the algorithms employed in the multidimensional scenario (for example, the similarity metrics and the ranking algorithms) are very different to the one that proposed. Recommendations can be computed on the fly efficiently and that our system can be tuned to obtain objectively good recommendations.

2.5 QueRIE: Collaborative Database Exploration (Magdalini Eirinaki, Suju Abraham, Neoklis Polyzotis, Naushin Shaikh)
This work describes an instantiation of the QueRIE framework,where the active user's session is represented by a set of query fragments. The recorded fragments are used to identify similar query fragments in the previously recorded sessions, which arein turn assembled in potentially interesting queries for the active user. We show through experimentation that theproposed method generates meaningful recommendations on real-life traces from the SkyServer database and propose a scalable design that enables the incremental update of similarities, making real-time computations on large amounts of data feasible.

III .QUERIE Architecture
The framework for generating query recommendations is given in Figure 1. The input query is accepted from the user.The user's query is forwarded to both the database system and the recommendation engine. The DBMS processes the input query and return the result to the database query interface. At the same time, the query is stored in the query log. The recommendation engine finds matching patterns in the system's query log by using previous user's querying behavior. And generate a set of query recommendation that is returned to the user through the query interface. Three approaches are used to generate the query recommendations: (1) suggested query by dictionary mapping (2) suggested query by tuple based query recommendations (3) suggested query by fragment history.



IV. Methodology
The active users query is decomposed into basic elements by the Querie framework paper Querie framework that captures the essence of the query's logic. These elements are used to compute similarities between users, as well as signature of the user's querying behavior. Recommendations are generated by mining queries from the system log that match well with the Signature. The proposed system consists of the following three modules:

- Database Monitoring
- Query Analyzer
- Query Recommendation

When a user logs in the system the user have to submit an SQL query to the query interface. The SQL query is sent to the database as well as the recommendation engine. Also the query is fragmented and stored in the system log. The query is processed by the query analyzer and returns the result. At the same time recommendation engine will generate the recommendation that match well with the query.

4.1 Database Monitoring

Each time when a user logs in the system he/she have to submit an SQL query through the database query interface.SQL query which is provided by the user is sent to syntax checker to check SQL syntax then the SQL query is fragmented and then transmitted to both the DBMS and recommendation engine. The input query is fragmented and stored in the system query log. This will make the comparison easier when finding the highly similar patterns from the system query log. Query fragmentation also makes storing the query in the database in an easier way.

4.2 Query Analyzer

The DBMS requested to the database and return the results to the user and then query is fragmented, creating an implicit query profile which consists of fragments. The given query is decomposed into fragments with respect to the keywords such as select, from, where, group by, having, order by. Names are given for the fragmented queries. The fragmented query attributes are stored in the fragment table with respect to the fragment name.

V.QUERY PREPROCESSING

In order to create the fragment-based query and session vectors, we needed to preprocess the queries included in the query logs and decompose them. This process consists of two steps, namely query generalization and query parsing. In the first step, the queries are generalized based on a set of rules, in order to be analyzed and matched more efficiently. Then, they are parsed and converted into a template, in preparation for comparison analysis.

5.1Query Relaxation

Because of the plethora of slightly dissimilar queries existing in the query logs, we decided to relax them in order to increase their cardinality, and thus the probability of finding similarities between different user sessions. Our intuition is that if two users query the same table and attributes, using slightly different filtering conditions, the algorithm should consider them as similar.As part of this relaxing process, we follow a simplified version of the framework proposed in [7]. In essence, all the WHERE clauses are relaxed by converting the numerical data and string literals to generic string representations. For example, all strings are replaced by STR, all hexadecimal numbers by HEXNUM and all decimals by NUM. A similar generalization is also followed for lists or ranges of numbers and strings. The mathematical and set comparators are also replaced by string equivalents, for example "=" is replaced by EQU and " " by COMPARE . In the current implementation of QueRIE we do not treat different numeric intervals as separate, however this is orthogonal to the framework and part of our future work plans.

5.2Query Parsing

Once the queries are generalized, they are converted into fragments. The current implementation of QueRIE only supports SPJ (SELECT, PROJECT, JOIN) queries, whereas if a query includes sub-queries, these are dropped. However, this is an implementation detail orthogonal to the overall framework, which can be easily extended to support sub queries. Each of the SPJ fragments are separated using regular expressions. The Start and End designated keywords used to identify fragments are shown in Table 1.Each distinct fragment is assigned a numerical identifier, used in the query and session vector representation. For each new fragment not previously recorded in the query log, QueRIE generates a new identifier. Such updates occur in real-time, as the current user posts a query including new fragments. In the case of the WHERE clause, only the joins and the filter conditions are stored. Because of the generalization, the fragments in the WHERE clause are not differentiated based on their actual values, rather based on the at-tributes used for filtering. For example, s:x 0:2 and s:x 0:8 will be represented by the same fragments. In addition we do not differentiate (i.e. handle differently) between joins and filters, as we anticipate the similarity calculation would generate proper results regardless of the type of WHERE condition.

Table 1: Parsing keywords

Fragment name	Start keyword	End keyword
Attribute string	SELECT	FROM
Relation string	FROM	WHERE, GROUP BY, ORDER BY, end of query
Where string	WHERE	GROUP BY, ORDER BY, end of query
Group By string	GROUP BY	ORDER BY, HAV-ING, end of query
Having string	HAVING	ORDER BY, end of query

Table 2: Data Set Statistics

# Sessions	180
# Distinct queries	1400
# Distinct query fragments	755
# Non-zero pair-wise fragment similarities	30436

CONCLUSIONS

In this paper we presented a fragment-based instantiation of 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. We also performed an experimental evaluation of various parameters of the framework using real traces from the Sky Server database. Overall, we showed that the precision of the recommendations is close to 80% when the active user's session is included in the prediction process, we employ the weighted scheme, and top-n 2 f5; 10g (with all other parameters set to default). This shows that QueRIE is very effective in generating useful recommendations to the end users of relational database systems. In terms of performance, QueRIE's fragment-based recommendation engine is able to generate real-time recommendations in quite fast (an average of 25 sec for each session in the test set).We are currently working on evaluating the remaining parameters of the problem. We also plan to compare the fragment-based instantiation with the tuple-based one, proposed in our previous work

REFERENCES

[1] Magdalini Eirinaki, Suju Abraham, Neoklis Polyzotis and Naushin Shaikh "QueRIE: Collaborative database exploration," in *Proc. IEEE Transactions*,vol.26, no.7,july 2014.
[2] A. Thusoo *et al.*, "Hive - A petabyte scale data warehouse using hadoop," in *Proc. IEEE 26th ICDE*, Long Beach, CA, USA, Mar. 2010, pp. 996–1005.
[3] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis, "Collaborative filtering for interactive database exploration," in *Proc. 21st Int. Conf. SSDBM*, New Orleans, LA, USA, 2009, pp. 3–18.
[4] S. Mittal, J. S. V. Varman, G. Chatzopoulou, M. Eirinaki, and N. Polyzotis, "QueRIE: A recommender system supporting interactive database exploration," in *Proc.IEEE ICDM*, Sydney, NSW, Australia, 2010.
[5] J. Akbarnejad *et al.*, "SQL QueRIE recommendations,"*PVLDB*, vol. 3, no. 2, pp. 1597–1600, 2010.
[6] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proc. 28th STOC*, NewYork, NY, USA, 1996.
[7] E. Cohen, "Size-estimation framework with applications to transitivenclosure and reachability," *J. Comput. Syst. Sci.*, vol. 55, no. 3, pp. 441–453, 1997.Pocket Telephone, Inc.
[8] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering,"*IEEE Internet Comput.*, vol. 7, no. 1, pp. 76–80, Jan./Feb.