SQL QueRIE Recommendations: a query fragment-based approach

Javad Akbarnejad
Computer Engineering Dept.
San Jose State Univ.
San Jose, CA

Magdalini Eirinaki
Computer Engineering Dept.
San Jose State Univ.
San Jose, CA

Suju Koshy
Computer Engineering Dept.
San Jose State Univ.
San Jose, CA

Duc On
Computer Engineering Dept.
San Jose State Univ.
San Jose, CA

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

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 interface (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 transformed 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

1. 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 cannot 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. All those issues make the problem of interactive database exploration very different from its web counterpart.

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 this 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

[1]`http://genome.ucsc.edu/`
[2]`http://cas.sdss.org/`

queries for the active user. We show through experimentation that the proposed method generates meaningful recommendations on real-life traces from the SkyServer 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 details concerning the queries' preprocessing; Section 6 includes some experimental results that evaluate several parameters of our framework and Section 7 concludes the paper with our plans for future work.

2. RELATED WORK

Even though the problem of generating personalized recommendations has been broadly addressed in the Web context [10], only a handful of related works exist in the database context. Some work has been done in the area of personalized recommendations for keyword or free-form query interfaces [11]. In this scenario, a user queries a database using keywords or free-form text and the personalization system recommends items of interest. Our approach is different from this scenario because it aims to assist users who query relational databases using either ad-hoc or form-based queries. Also, our framework recommends queries instead of "items" from the database. Finally, QueRIE does not require from the users to explicitly declare their preferences beforehand in order to generate recommendations.

A multidimensional query recommendation system is proposed in [3, 5, 4]. In this work the authors address the related problem of generating recommendations for data warehouses and OLAP systems. In this work, the authors propose a framework for generating Online Analytical Processing (OLAP) query recommendations for the users of a data warehouse. Although this work has some similarities to ours (for example, the challenges that need to be addressed because of the database context), the techniques and the algorithms employed in the multidimensional scenario (for example, the similarity metrics and the ranking algorithms) are very different to the ones we propose.

The necessity of a query recommendation framework is emphasized in [6], where the authors outline the architecture of a collaborative query management system targeted at large-scale, shared-data environments. As part of this architecture, they suggest that data-mining techniques can be applied to the query logs in order to generate query suggestions. The authors present a general outline of a framework for query recommendations pointing out that this is a challenging process. However, they do not provide any technical details on how such a recommendation system could be implemented.

Two very recent works propose frameworks for query recommendations using the information recorded in the query logs [13, 14]. In [13], the authors propose a query recommender system that represents the past queries using the most frequently appearing tuple values. Then, after predicting which new tuples might be of interest to the end user, they reconstruct the query that retrieves them. Contrary to our work, this approach is tuple-based. Moreover, the proposed scheme works better with relations that have discrete attribute values, contrary to scientific databases, where most attributes are numeric. The authors also propose a global ranking of the queries, based on the statistics of the database and not the query logs. Both approaches are evaluated in a preliminary empirical study, yet no discussion on scalability issues is provided. In [14], the authors propose a framework that recommends join queries. They use the data recorded in the query logs and reconstruct queries, however they assume that the end user should provide the system with some tables to be used as input and other tables to be used as output, along with the respective selection conditions. This approach clearly differs from ours in that they do not take the current user's session into consideration, neither they perform recommendations in the traditional "personalized" form (i.e. finding similarities among users or items).

In our previous work [2, 9] we defined the QueRIE conceptual framework and proposed a user-based approach that focused on the tuples touched by each query in a user's session. The system finds similarities among the current and past users based on this tuples' representation of the user sessions. A session summary predicting tuples of interest is constructed and used to identify queries recorded in the query logs that touch the same tuples. Since the proposed instantiation is based on user-based collaborative filtering, an approximation technique for accelerating the real-time calculations was also proposed. Contrary to our previous work, in this work we follow the item-based collaborative filtering approach that allows most of the calculations to be performed offline, thus enhancing the real-time performance of the system. Moreover, the queries are represented by their fragments and not the tuples they retrieve. In this way, the proposed instantiation focuses on identifying similar queries in terms of structural similarity thus capturing the semantics of the database exploration. The prototype of the QueRIE framework, incorporating both instantiations, will be presented in [1].

3. PRELIMINARIES

Users typically explore a relational database through a sequence of SQL queries. The goal of the exploration is to discover interesting information or verify a particular hypothesis. The queries are formulated based on this goal and reflect the user's overall information need. As a consequence, the queries posted by a user during one "visit" (commonly called *session*) to the database are typically correlated, in that the user formulates the next query in the sequence after having inspected the results of previous queries.

Given a user i, let $\mathcal{Q}_i$ denote the set of SQL queries that the user has posed. We model this subset of the database covered by the queries of each user as a *session summary*. This summary captures the parts of the database accessed by the user and incorporates a metric of importance for each part. Contrary to Web recommender systems, where the users are represented by the items they visit/rate/purchase, in the context of relational databases, several ways to model the session summaries exist. For instance, a crude summary may contain the names of the relations that appear in the queries of the user, and the importance of each relation can be measured as the number of queries that reference it. On the other extreme, a detailed summary may contain the actual results inspected by the user, along with an explicit rating of each result tuple. Assuming that the choice of the summary is fixed for all users, we use S_i to denote the summary for user i.

To generate recommendations, the framework computes a "predicted" summary S_0^{pred}. This summary captures the predicted degree of interest of the active user S_0 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. The predicted summary is defined as follows:

$$S_0^{\text{pred}} = f(\alpha * S_0, (1-\alpha) * \{S_1, \ldots, S_h\}). \qquad (1)$$

In other words, the predicted summary depends on both the active user S_0 and the summaries $S_1, \ldots, S_h$ of past users.

Contrary to Web recommender systems that rely exclusively upon the summaries of past users, we introduce a "mixing factor" $\alpha \in [0, 1]$ that determines the importance of the active user's queries as opposed to these of the past users in the computation of the predicted summary. In this way, we are able to predict queries that "expand" queries previously submitted by the user, in terms of adding slightly different clauses, parameters, or restructuring the query. Intuitively, we expect the active user to behave in a similar way, by posing queries that cover adjacent or overlapping parts of the database, in order to locate the information they are seeking. We should note, however, that the framework will also predict completely different queries as well, depending on the information recorded in the query logs.

Using S_0^{pred}, the framework constructs queries that cover the subset of the database with the highest predicted importance. In turn, these queries are presented to the user as recommendations. This step differs from the respective one in Web recommender systems since, in this case, the predicted summary is not a straightforward representation of queries. On the contrary, we need to devise algorithms that, given the predicted summary, can synthesize meaningful queries that will form the recommendation set.

Overall, our framework consists of three components: (a) Session summaries: the construction of a session summary for each user based on her past queries, (b) Recommendation seed computation: the computation of a predicted summary S_0^{pred} that serves as the seed, and (c) Generation of query recommendations: the generation of queries based on S_0^{pred}. An interesting point is that components (a) and (c) form a closed loop, going from queries to session summaries and back. Again, this design choice follows the fact that all user interaction with a relational database occurs through declarative queries.

In what follows, we investigate a query fragments-based approach to modeling the queries, and consequently the users.

4. FRAGMENT-BASED RECOMMENDATIONS

In order to generate recommendations, we follow a methodology similar to the item-based collaborative filtering. This approach is based on the pair-wise similarity among the items involved in the recorded user sessions. Items that co-appear in many sessions are considered similar to each other and these similarities are used in order to generate recommendations for an active session. Contrary to user-based collaborative filtering, this technique allows the calculation of all similarities offline, thus accelerating the real-time calculations and enabling fast recommendations' generation.

In this paper, we represent each user session by the query fragments (attributes, tables, joins and predicates) identified in the respective queries. The objective is to identify fragments that co-appear in several queries posed by different users, and use them in the recommendation process. Thus, QueRIE first calculates offline the pair-wise similarities of all query fragments recorded in the query logs. These similarities are subsequently used to predict, in real time, the "ranking" (i.e. importance) of each fragment with regards to the current user session. In turn, the highest ranked query fragments are selected and used to retrieve queries that include them, which are used as recommendations. The proposed algorithm is presented in more detail in what follows.

4.1 Session summaries.

The session summary vector S_i for a user i consists of all the query fragments ϕ of the user's past queries. Let $\mathcal{Q}_i$ represent the set of queries posed by user i during a session and F represent the set of all distinct query fragments recorded in the query logs. We assume that the vector S_Q represents a single query $Q \in \mathcal{Q}_i$. For a given fragment $\phi \in F$, we define $S_Q[\phi]$ as a binary variable that represents the presence or absence of ϕ in a query Q. Then $S_i[\phi]$ represents the importance of ϕ in session S_i.

We propose two different weighting schemes for computing the fragment weights in S_i:

Binary scheme.

$$S_i = \bigvee_{Q \in \mathcal{Q}_i} S_Q. \tag{2}$$

In this scheme all participating fragments receive the same importance weight, regardless of whether they appear in many queries in the session or only one.

Weighted scheme.

$$S_i = \sum_{Q \in \mathcal{Q}_i} S_Q. \tag{3}$$

In this approach fragments that appear more than once in a user session will receive higher weight than others.

4.2 Recommendation seed computation.

Using the session summaries of the past users and a vector similarity metric, we construct the $(|F| \times |F|)$ *fragment-fragment matrix* that contains all similarities $sim(\rho, \phi)$, $\rho, \phi \in F$. Intuitively, and according to the item-based collaborative filtering approach, the more the sessions that include both fragments, the more similar these fragments are. The similarity metric employed depends on the weighting scheme that was chosen in the previous step, thus we employ Jaccard's coefficient and cosine similarity for the binary and weighted schemes respectively. We should note that all pairwise similarities are calculated and stored off-line. This results in a very efficient execution of the algorithm in terms of computational time.

The recommendation seed, modeled by S_0^{pred}, represents the estimated importance of each query fragment with regard to the active user's behavior S_0. Similarly to the item-to-item collaborative filtering approach of web recommender systems, we employ the fragment-to-fragment similarities that are computed in the previous step:

$$S_0^{\text{pred}}[\phi] = \frac{\sum_{\rho \in R} S_0[\rho] * sim(\rho, \phi)}{\sum_{\rho \in R} sim(\rho, \phi)}, \tag{4}$$

where R represents the set of top-k similar query fragments ($k \leq |F|$). Please note that, even if we follow the binary approach, S_0^{pred} is not a binary vector.

As shown in Equation 1, we introduce a "mixing factor" $\alpha \in [0, 1]$ that allows us to include or exclude the fragments of the active user session in the recommendation process. α is a parameter of the QueRIE framework. When $\alpha = 0$ we follow the classic item-based collaborative filtering approach, whereas when $\alpha = 1$ we follow a content-based approach, in that only the fragments included in the active user's queries are taken into consideration. More discussion on the effect of α is included in Section 3.

4.3 Generation of query recommendations.

The recommendation set will include queries that have been previously recorded in the query logs. In this way, we ensure that the queries are understandable and executable, since they were authored by humans. This decision allows for faster and more intuitive recommendations, as compared to the option of synthesizing queries on the fly.

Once the predicted summary S_0^{pred} has been computed, the top-n fragments F_n (i.e. the fragments that have received the higher weight) are selected. Then all past queries Q, $Q \in \bigcup_i \mathcal{Q}_i$ receive a rank QR with respect to the top-n fragments:

$$QR(Q) = \frac{|F_Q \cap F_n|}{|F_Q|} * \frac{|F_Q \cap F_n|}{n}, \quad (5)$$

where F_Q represents the fragments of query Q. In other words, the queries are ranked based on a normalized metric measuring the number of common query fragments of each query Q to the top-n list. Finally, the top-m ranked queries are used as the recommendation set.

5. 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.

5.1 Query 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 "$\leq$" 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.

5.2 Query 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 subqueries. 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 attributes used for filtering. For example, $s.x \geq 0.2$ and $s.x \geq 0.8$ 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, HAVING, 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

6. EXPERIMENTAL EVALUATION

The framework proposed in this paper has been implemented in a prototype that will be demonstrated in [1]. In this section we present preliminary experimental results using real user traces. More specifically, we evaluate several parameters of the framework, namely the value of items used for the generation of recommendations n, the effect of the mixing factor α, and the employed weighted schemes.

6.1 Data Set.

We evaluated our framework using traces of the Sky Server database[3]. The traces contain queries posed to the database between the years 2006 and 2008. The query logs are anonymous, thus we used the methods described in [12] to clean and separate the query logs in sessions. For this reason, each session is considered as a different user. The characteristics of the data set and the queries are summarized in Table 2. Two real user sessions including a total of eight queries is included in [1].

6.2 Methodology.

In order to measure the prediction accuracy of QueRIE, we use the holdout set methodology [8]. The data is divided into two disjoint sets, the training set and the test set. The pair-wise fragment similarity is computed against the training set. Each user session in the test set is divided in two parts. One part is treated as the active user's queries, while the second part is treated as unseen (i.e. future) queries. Subsequently, using the active user's queries from the test set and the pre-calculated fragment-based similarities, QueRIE generates a set of query recommendations. We compare the recommended queries with the unseen queries from the test set and calculate the precision, recall and F-score for each session. This is performed by calculating these measures for each pair of queries, as shown in Equations 6, 7 and 8 and keeping the maximum value, assuming that the end user will also select only one out of the m recommended queries each time.

$$Precision = \frac{|F_r \cap F_u|}{|F_r|} \quad (6)$$

$$Recall = \frac{|F_r \cap F_u|}{|F_u|} \quad (7)$$

[3]We used the BestDR6 version.

Table 3: Default parameter values

Top-k	5
Top-n	5
Top-m	5
α	0.5
# weighting scheme	weighted (cosine)
Training set	160 sessions
Test set	20 sessions

$$F - Score = \frac{2 * Precision * Recall}{Precision + Recall} \quad (8)$$

In the formulas above, F_r and F_u represent the fragments of the recommended and unseen queries respectively. In the experiments that follow, we report the average precision and recall over the 160 sessions of the data set.

We performed several experiments evaluating the performance of the framework, and the effect of the various parameters of the algorithm. Due to space constraints, in this paper we present the most important findings in terms of the number of fragments n selected from S_0^{pred} to calculate the query rank QR, the mixing factor α, and the weighting scheme. Table 3 shows the default values kept constant for the remaining parameters in each case.

6.3 Experimental Results

Evaluation of the Top-n parameter. The recommended queries in QueRIE are identified, by first selecting the top-n fragments of the predicted summary S_0^{pred} and using them to rank all previous queries using the QR formula. Figures 1 and 2 show the average precision and F-score for various top-n values ($n \in \{3, 5, 10, 50\}$). We notice that the accuracy of the recommendations increases, as expected, with the value of n. However, for very large values of n, the accuracy decreases again. This is completely justifiable, since when n is a very large number, the notion of "most similar" fragments does no longer hold and barely similar items are included in the recommendation process. QueRIE achieves the higher precisions for $n = 10$ and $n = 5$ (0.8 and 0.75 respectively), whereas F-score is the same for both values (0.76 and 0.75 respectively). Given the small difference in terms of accuracy and the fact that the lower the number of fragments n, the faster the real-time calculations, we adopt $n = 5$ as the default value for the framework.

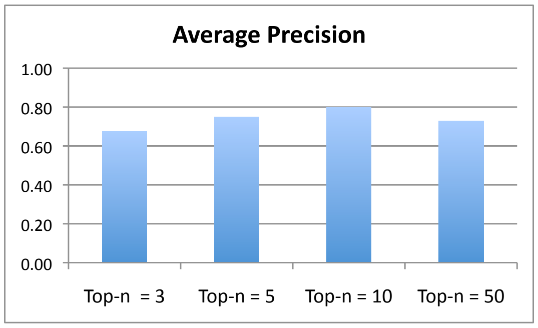

Figure 1: Average precision for various top-n values

Evaluation of the mixing factor α. In this instantiation of the QueRIE framework, the mixing factor α is introduced before the selection of the top-n fragments. Since including only the current user's session ($\alpha = 1$) is a content-based approach not employing the collective intelligence recorded in the query logs, we only evaluate the impact of including ($\alpha = 0.5$), or excluding ($\alpha = 0$) the fragments recorded in the active user's session. As expected intuitively, the accuracy of the recommendations is enhanced significantly when the active user's fragments are incorporated in the recommendation process. More specifically, precision and F-score are 0.76 and 0.74 respectively for $\alpha = 0.5$, whereas they drop to 0.48 and 0.43 when $\alpha = 0$, as shown in Figures 3 and 4. This verifies our initial claim that database recommender systems are very different in nature from their web counterparts. As pointed out in Section 3, one significant difference is that, in the case of SQL queries we want to expand or enhance the queries that were previously submitted by the user. The user benefits from this addition, since probably most users are interested in posting queries similar to the ones they have already posted during the same session.

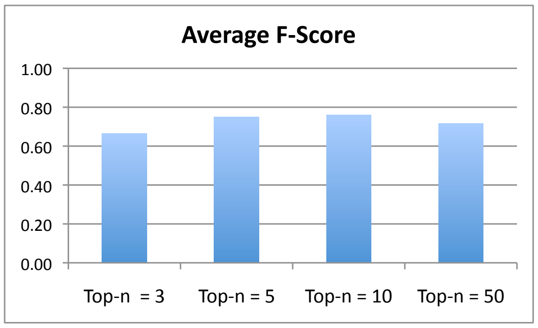

Figure 2: Average f-score for various top-n values

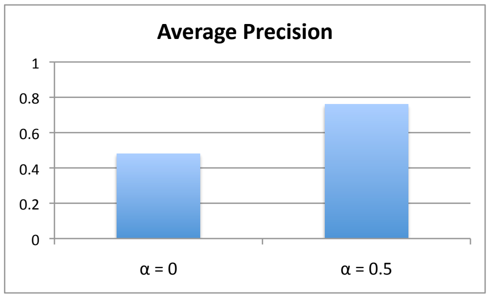

Figure 3: Average precision for different α values

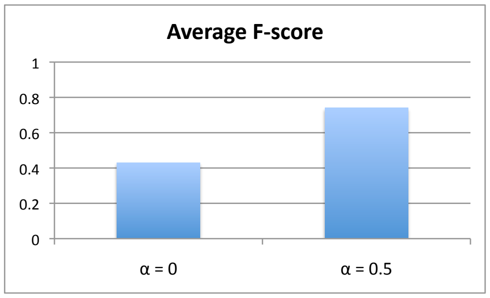

Figure 4: Average f-score for different α values

Evaluation of the weighting scheme. Depending on the weighting scheme selected, the representation of the query and session vectors, and consequently the metrics used to calculate the similarities between fragments, differs. In this instantiation we have introduced the *binary* and the *weighted* schemes and employ the Jaccard coefficient and the cosine similarity metric respectively. In this set of experiments, we evaluate the effect of the representation. Intuitively, the binary representation is much more simplistic and is expected to provide less accurate results, since valuable information with regards to the importance of each fragment in a session is missing. The results, shown in Figures 5, 6 verify this intuition, however we notice that the difference is very small, with a precision of 0.74 and 0.79 for the binary and weighted schemes respectively, and an F-score of 0.69 and 0.74 respectively. For the specific dataset both schemes performed similarly in terms of real-time performance. Thus we adopt the *weighted* scheme as the default value, since it resulted in slightly better prediction accuracy.

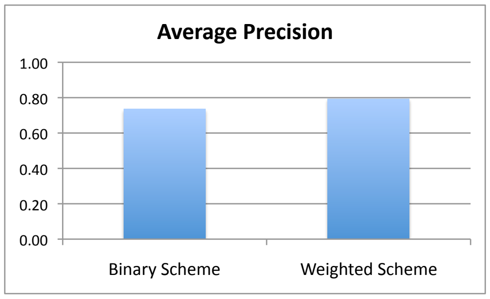

Figure 5: Average precision for different weighting schemes

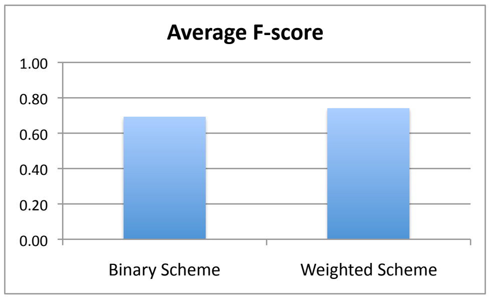

Figure 6: Average f-score for different weighting schemes

7. 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 SkyServer 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-$n \in \{5, 10\}$ (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.

8. ADDITIONAL AUTHORS

9. REFERENCES

[1] J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. S. V. Varman. SQL QueRIE Recommendations (demo paper). In *Proc. of the 36th International Conference on Very Large Data Bases (VLDB 2010)*, 2010.

[2] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Collaborative filtering for interactive database exploration. In *Proc. of the 21st International Conference on Scientific and Statistical Database Management (SSDBM '09)*, 2009.

[3] A. Giacometti, P. Marcel, and E. Negre. Recommending Multidimensional Queries. In *Proc. of the 11th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'09)*, 2009.

[4] A. Giacometti, P. Marcel, E. Negre, and A. Soulet. Query recommendations for olap discovery driven analysis. *Intl. Journal on Data Warehousing and Mining (IJDWM) (to appear)*.

[5] A. Giacometti, P. Marcel, E. Negre, and A. Soulet. Query recommendations for olap discovery driven analysis. In *Proc. of the ACM 12th International Workshop on Data Warehousing and OLAP (DOLAP'09)*, 2009.

[6] N. Khoussainova, M. Balazinska, W. Gatterbauer, Y. Kwon, and D. Suciu. A case for a collaborative query management system. In *Proc. of the 4th Biennal Conference on Innovative Data Systems Research (CIDR 2009)*, 2009.

[7] N. Koudas, C. Li, A. K. H. Tung, and R. Vernica. Relaxing join and selection queries. In *Proc. of the 33nd international conference on Very large data bases (VLDB '06)*, pages 199–210, 2006.

[8] B. Liu. *Web Data Mining: Exploring Hyperlinks, Contents and Usage Data*. Springer, 2nd edition, 2007.

[9] S. Mittal, J. S. V. Varman, G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. QueRIE: A Recommender System supporting Interactive Database Exploration. In *2009 edition of the IEEE International Conference on Data Mining series (ICDM'09) - to appear in ICDM 2010 proceedings because of editor's error*, 2009.

[10] B. Mobasher. *The Adaptive Web: Methods and Strategies of Web Personalization*, volume 4321 of *LNCS*, chapter Data Mining for Personalization, pages 90–135. Springer, Berlin-Heidelberg, 2007.

[11] A. Simitsis, G. Koutrika, and Y. Ioannidis. Precis: From unstructured keywords as queries to structured databases as answers. *VLDB Journal*, 17(1):117–149, 2008.

[12] V. Singh, J. Gray, A. Thakar, A. S. Szalay, J. Raddick, B. Boroski, S. Lebedeva, and B. Yanny. Skyserver traffic report - the first five years. *Microsoft Research, Technical Report MSR TR-2006-190*, 2006.

[13] K. Stefanidis, M. Drosou, and E. Pitoura. "You May Also Like" Results in Relational Databases. In *3rd International Workshop on Personalized Access, Profile Management, and Context Awareness in Databases (PersDB 2009)*, 2009.

[14] X. Yang, C. M. Procopiuc, and D. Srivastava. Recommending join queries via query log analysis. In *25th International Conference on Data Engineering (ICDE 2009)*, pages 964–975, 2009.