2009

Query Recommendations for Interactive Database Exploration

Gloria Chatzopoulou, *University of California, Riverside*
Magdalini Eirinaki, *San Jose State University*
Neoklis Polyzotis, *University of California, Santa Cruz*

Query Recommendations for Interactive Database Exploration

Gloria Chatzopoulou[1] *, Magdalini Eirinaki[2], and Neoklis Polyzotis[3]

[1] Computer Science Dept., University of California Riverside, USA
chatzopd@cs.ucr.edu
[2] Computer Engineering Dept., San Jose State University, USA
magdalini.eirinaki@sjsu.edu
[3] Computer Science Dept., University of California Santa Cruz, USA
alkis@cs.ucsc.edu

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. To assist users in this context, we draw inspiration from Web recommender systems and propose the use of personalized query recommendations. The idea is to track the querying behavior of each user, identify which parts of the database may be of interest for the corresponding data analysis task, and recommend queries that retrieve relevant data. We discuss the main challenges in this novel application of recommendation systems, and outline a possible solution based on collaborative filtering. Preliminary experimental results on real user traces demonstrate that our framework can generate effective query recommendations.

1 Introduction

Relational database systems are becoming increasingly popular in the scientific community to manage large volumes of experimental data. Examples include the Genome browser[4] that provides access to a genomic database, and Sky Server[5] that stores large volumes of astronomical measurements. The main advantage of a relational database system is that it supports the efficient execution of complex queries, thus enabling users to interactively explore the data and retrieve interesting information. It should be noted that the aforementioned systems employ web-based query interfaces in order to be accessible to a broad user base.

* This work was performed while the author was affiliated with UC Santa Cruz
[4] http://genome.ucsc.edu/
[5] http://cas.sdss.org/

Even though a database system offers the means to run complex queries over large data sets, the discovery of useful information remains a big challenge. 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. This issue clearly hinders data exploration, and thus reduces the benefits of using a database system.

To address this important problem, we draw inspiration from the successful application of recommender systems in the exploration of Web data. In particular, we focus on approaches based on user-based collaborative filtering. The premise is simple: If a user A has similar querying behavior to user B, then they are likely interested in the same data. Hence, the queries of user B can serve as a guide for user A.

The transfer of this paradigm entails several challenging problems. In web collaborative filtering systems, a user-item matrix approach is used to generate recommendations. More specifically, each user is represented as an item vector, where the values of the vector elements correspond to the user's preferences for each item (such as movie ratings, purchased products, read articles, etc.) The similarities between users in this representation can be easily computed using vector similarity metrics. Given the most similar users and their preferences, the collaborative filtering system can subsequently predict what items will interest each user, and generate item recommendations.

The aforementioned methodology cannot be directly applied to the context of a relational database for several reasons. First, we observe that in the case of a database the "items" of interest are database records, and the users access these items indirectly by posing SQL queries. Thus, even though the users' behavior is identified by the set of queries they send to the database, their interest lies on the database tuples they retrieve. Given that SQL is a declarative language, however, the same data can be retrieved in more than one way. This complicates the evaluation of similarity among users based on their queries alone, since it is no longer obvious whether they are interested in the same "items".

This raises a second important issue that needs some consideration. The similarity between users can be expressed as the similarity between the fragments of their queries or, alternatively, the data that they retrieve. This is not as straightforward, since a query fragment or a tuple might have different levels of importance in different user sessions. Thus, we must be able to create implicit user profiles that model those levels of importance, in order to effectively compare the users.

Finally, contrary to the user-based collaborative filtering approach, the recommendation 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 SQL queries in order to recommend them. Moreover, these SQL queries must be meaningful and intuitive, so that users can parse them and understand their

intent and usefulness. All those issues make the problem of interactive database exploration very different from its web counterpart.

In this paper, we present our work in the development of a query recommender system for relational databases. We first discuss an abstract framework that conceptualizes the problem and defines specific components that must be instantiated in order to develop a solution. Based on this framework, we develop a solution that transfers the paradigm of collaborative filtering in the context of relational queries. The recommended solution can be implemented using existing technology, and is thus attractive for a real-world deployment. Finally, we present an experimental study on real user traces from the Sky Server database. Our results indicate that our first-cut solution can provide effective query recommendations, and thus demonstrate the potential of our approach in practice.

The remainder of the paper is structured as follows. We review the related work in Section 2 and cover some preliminaries in Section 3. Section 4 discusses the conceptual framework and its instantiation. The experimental results are presented in Section 5. Section 6 concludes the paper and outlines directions for future work.

2 Related Work

So far, the work that has been done in the area of personalized databases has focused to keyword-based query recommendation systems [1]. In this scenario, a user can interact with a relational database through a web interface that allows him/her to submit keywords and retrieve relevant content. The personalization process is based on the user's keyword queries, those of previous users, as well as an explicit user profile that records the user's preferences with regards to the content of the database. Clearly, our approach is different from this scenario in several ways. First, the proposed framework is meant to assist users who pose complex SQL queries to relational databases. Moreover, the system does not require from its users to create an explicit profile. This gives a higher level of flexibility to the system, since the same user might have different information needs during different explorations of the database.

Our inspiration draws from the successful application of user-based collaborative filtering techniques, proposed in the Web context [2, 3, 4, 5, 6, 7, 8, 9, 10]. As previously mentioned, this approach cannot be directly applied to the relational database context. The inherent nature of interactive database exploration raises certain implications that cannot be addressed by the straightforward collaborative filtering approach. In this work, we are based on its premises, but extend them in order to apply them in the database environment.

The challenges of applying data mining techniques to the database query logs are also addressed in [11]. In this work, the authors outline the architecture of a Collaborative Query Management System targeted at large-scale, shared-data environments. As part of this architecture, they independently suggest that data mining techniques, such as clustering or association rules, can be applied to the query logs in order to provide the users with query suggestions. We should stress,

however, that contrary to our work, the authors do not provide any technical details on how such a recommendation system could be implemented.

The work presented in this paper is part of the QueRIE (Query Recommendations for Interactive database Exploration) project. In this project, we investigate the application of personalization techniques in interactive database exploration, particularly to assist the user in discovering interesting subsets of the database with minimal effort.

3 Preliminaries

Our work considers the interactive exploration of a relational database using SQL queries. In what follows, we summarize some basic notions in this context that will be used in the remainder of the paper.

3.1 Database and Querying Model

We consider a relational database comprising N relations denoted as $R_1, \ldots, R_N$. We use Q to denote a Select-Project-Join (SPJ) query over the database, and $ans(Q)$ for its result set. We focus on the class of SPJ queries because they are common in interactive database exploration, particularly among the group of novice users which is the focus of our work.

We say that a tuple τ of some relation R_n, $1 \leq n \leq N$, is a witness for a query Q if τ contributes to least one result in $ans(Q)$. We use R_n^Q to denote the set of witnesses for Q from relation R_n. (For notational convenience, we assume that $R_n^Q = \emptyset$ if relation R_n is not mentioned in Q.) Overall, the subsets $R_1^Q, \ldots, R_N^Q$ contain the tuples that are used to generate the results of Q. In that respect, we say that $R_1^Q, \ldots, R_N^Q$ is the subset of the database touched by Q.

3.2 Interactive Data Exploration

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.

We identify users with unique integer identifiers. Given a user i, let $\mathcal{Q}_i$ denote the set of SQL queries that the user has posed. In accordance with the previous definitions, we assume that the SQL queries belong to the class of SPJ queries. We define R_n^i, $1 \leq n \leq N$ as the union of R_n^Q for $Q \in \mathcal{Q}_i$, i.e., the set of tuples of relation R_n that the user's queries have touched. Hence, $R_1^i, \ldots, R_N^i$ represent the subset of the database that has been accessed by user i. A summary of the notation used throughout this paper is included in Table 1.

Table 1. Notation Summary

R_n	Relation n
R_n^Q	Set of witnesses for query Q from R_j
R_n^i	Set of tuples of R_n that queries of user i have retrieved
S_i	Session summary of user i
S_0^{pred}	Extended session summary for current user

4 Personalized Query Recommendations

The problem of personalized query recommendations can be formulated as follows: Given a user that is currently exploring the database, recommend queries that might be of interest to him/her. To generate such recommendations, the system will rely on information gathered from the querying behavior of past users, as well as the queries posed by the current user so far.

The information flow of the QueRIE framework is shown in Figure1. The active user's queries are forwarded 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. The Recommendation Engine combines the current user's input with information gathered from the database interactions of past users, as recorded in the Query Log, and generates a set of query recommendations that are returned to the user.

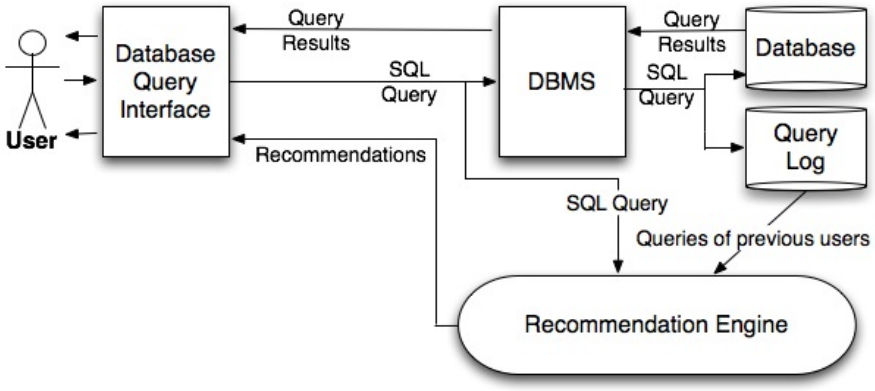

Fig. 1. QueRIE Architecture

In what follows, we identify the current user with the id 0, and note that $\mathcal{Q}_0$ contains the queries that the user has posed thus far. We use $\{1, \ldots, h\}$ to denote the set of past users based on which recommendations are generated.

The following sections describe a solution to this interesting problem of generating personalized query recommendations. We begin by discussing a concep-

tual framework that can encompass different approaches, and then propose an instantiation based on collaborative filtering.

4.1 Conceptual Framework

Clearly, the queries of each user touch a subset of the database that is relevant for the analysis the user wants to perform. We assume that this subset is modeled 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. 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, our framework generates a "predicted" summary S_0^{pred}. This 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. As an example, if the summary S_0 contains the names of the relations that the user has referenced so far, then S_0^{pred} may contain more relations that might be of interest, along with the respective degree of "interestingness" for each part.

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.

Overall, our framework consists of three components: (a) the construction of a session summary for each user i based on the queries in $\mathcal{Q}_i$, (b) the computation of S_0^{pred} based on the active user S_0 and the summaries $S_1, \ldots, S_h$ of past users, and (c) 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 summaries and back. This is a conscious design choice following the fact that all user interaction with a relational database occurs through declarative queries.

4.2 A Witness-based Collaborative Filtering Approach

We now discuss an instantiation of the previously described framework. In the following sections, we discuss the model and construction of session summaries using witnesses, the computation of the extended summary S_0^{pred}, and the recommendation algorithm.

Session Summaries. The session summary S_i is represented as a weighted vector that corresponds to the database tuples. We assume that the total number of tuples in the database, and as a consequence the length of the vector, is T.

The weight $S_i[\tau]$ represents the importance of a given tuple $\tau \in T$ in session S_i. In what follows, we describe the computation of tuple weights in S_i.

We assume that the vector S_Q represents a single query $Q \in \mathcal{Q}_i$. The value of each element $S_Q[\tau]$ signifies the importance of the tuple as the witness for Q. We propose two different weighting schemes for computing the tuple weights in S_i:

Binary weighting scheme.

$$S_Q[\tau] = \begin{cases} 1 & \text{if } \tau \text{ is a witness;} \\ 0 & \text{if } \tau \text{ is not a witness.} \end{cases} \tag{1}$$

This is the most straightforward approach. There are two options: either a tuple is a witness in Q, or not. All participating tuples receive the same importance weight.

Result weighting scheme.

$$S_Q[\tau] = \begin{cases} 1/|ans(Q)| & \text{if } \tau \text{ is a witness;} \\ 0 & \text{if } \tau \text{ is not a witness.} \end{cases} \tag{2}$$

Here $ans(Q)$ is the result-set of Q. The intuition is that the importance of τ is diminished if Q returns many results, as this is an indication that the query is "unfocused". On the other hand, a small $ans(Q)$ implies that the query is very specific, and thus the witnesses have high importance.

Given the vectors S_Q for $Q \in \mathcal{Q}_i$, we define the session summary of user i as:

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

Using the session summaries of the past users, we can construct the $(h \times T)$ *session-tuple matrix* which, as in the case of the user-item matrix in web recommender systems, will be used as input to our recommendation algorithm.

Computing S_0^{pred}. Similarly to session summaries S_i, the predicted summary S_0^{pred} is a vector of tuple weights. Each weight signifies the predicted importance of the corresponding tuple for the active user. In order to compute those weights, we adopt the method of a linear summation that has been successfully employed in user-based collaborative filtering. More specifically, we assume the existence of a function $sim(S_i, S_j)$ that measures the similarity between two session summaries and takes values in $[0, 1]$. The similarity function $sim(S_i, S_j)$ can be realized with any vector-based metric. In this work, we employ the cosine similarity:

$$sim(S_i, S_j) = \frac{S_i S_j}{\|S_i\|_2 \|S_j\|_2}. \tag{4}$$

This implies that two users are similar if their queries imply similar weights for the database tuples.

The predicted summary is defined as a function of the current user's summary S_0 and the normalized weighted sum of the existing summaries:

$$S_0^{\text{pred}} = \alpha * S_0 + (1 - \alpha) * \frac{\sum_{1 \leq i \leq h} sim(S_0, S_i) \cdot S_i}{\sum_{1 \leq i \leq h} sim(S_0, S_i)} \tag{5}$$

The value of the "mixing" factor $\alpha \in [0, 1]$ determines which users' traces will be taken into consideration when computing the predicted summary. If $\alpha = 0$, then we follow the user-based collaborative filtering approach and take into account only the past users' traces. On the other hand, when $\alpha = 1$, only the active user's session summary is taken into account when generating recommendations, resulting in what is known in the recommendation systems as content-based filtering. Finally, any value in between allows us to bias the predicted vector and assign more "importance" to either side, or equal "importance" to both the current and the previous users (when $\alpha = 0.5$). This bias can be useful for two reasons. First, we do not want to exclude from the recommendation set any queries that touch the tuples already covered by the user. Even though there might exist some overlap, such queries may provide a different, more intuitive presentation (e.g., a different PROJECT clause), making it easier for the user to search for the information she is looking for. Second, by including the covered tuples in S_0^{pred}, we are able to predict queries that combine the seen tuples with unseen tuples from other relations. In other words, we are able to predict queries that "expand" on the results already observed by the user. Intuitively, we expect from 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. These two observations derive from the nature of database queries and are in some sense inherent in the problem of personalized query recommendations.

Generating Query Recommendations. Having computed S_0^{pred}, the algorithm recommends queries that retrieve tuples of high predicted weights. One possibility would be to automatically synthesize queries out of the predicted tuples in S_0^{pred}, but this approach has an inherent technical difficulty. Another drawback of this approach is that the resulting queries may not be intuitive and easily understandable. This is important in the context of query recommendations, as users must be able to interpret the recommended queries before deciding to use them.

To avoid the aforementioned issues, we choose to generate recommendations using the queries of past users. Such recommendations are expected to be easily understandable, since they have been formulated by a human user. More concretely, we maintain a sample of the queries posed by previous users. In the context of predicting queries for the active user, we assign to each query Q in the sample an "importance" with respect to S_0^{pred}. This importance is computed as the similarity between the query vector S_Q and S_0^{pred}, and is defined as follows:

$$rank(Q, S_0^{\text{pred}}) = sim(S_Q, S_0^{\text{pred}}). \tag{6}$$

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.

5 Experimental evaluation

We completed a prototype implementation of the framework described in the previous section. We are also in the process of developing a visual query interface that employs our framework to provide on-demand recommendations to users who navigate the database. In this section we present preliminary experimental results of using our system with real database traces, as well as examples of queries and the related recommendations generated for this data set.

5.1 Data Set.

We evaluated our framework using traces of the Sky Server database[6]. The traces contain queries posed to the database between the years 2006 and 2008. We used the methods described in [12] to clean and separate the query logs in sessions. The characteristics of the data set and the queries are summarized in Table 2.

Table 2. Data Set Statistics

Database size	2.6TB
#Sessions	720
#Queries	6713
#Distinct queries	4037
#Distinct witnesses	13,602,430
Avg. number of queries per session	9.3
Min. number of queries per session	3

5.2 Methodology.

We employ 10-fold cross validation to evaluate the proposed framework. More concretely, we partition the set of user sessions in 10 equally sized subsets, and in each run we use 9 subsets as the training set and we generate recommendations for the sessions in the remaining subset. For each test user session of size L, we build the session summary S_0 using $L-1$ queries and we thus generate recommendations for the L-th query of the user. In order to generate the top-n recommendations we use the queries in the current training set.

We experimented with different values for n. In this paper we report the results for $n = 3$ and $n = 5$. A larger recommendation set might end up being overwhelming for the end user, who is usually interested in selecting only a few recommended queries.

[6] We used version BestDR6.

The effectiveness of each recommended query is measured against the L-th query of the session, using the following precision and recall metrics:

$$precision = \frac{\mid \tau_{Q_L} \cap \tau_{Q_R} \mid}{\mid \tau_{Q_R} \mid} \tag{7}$$

$$recall = \frac{\mid \tau_{Q_L} \cap \tau_{Q_R} \mid}{\mid \tau_{Q_L} \mid} \tag{8}$$

where τ_{Q_L} represents the witnesses of the L-th query and τ_{Q_R} represents the witnesses of the recommended query. The precision metric shows the percentage of "interesting" tuples to the user with respect to all the recommended tuples. The recall metric captures the hit ratio of each recommended query with respect to the last query of the user.

Following the practice of previous studies in recommender systems [13], we report for each user session the maximum recall over all the recommended queries, and compute the precision for the query that achieved maximum recall. We also report the average precision and recall for one set of recommendations. Unless otherwise noted, we set $\alpha = 0.5$.

5.3 Results.

We conducted several experiments to evaluate different aspects of our system. Overall, the results show the feasibility of query recommendations as a guide for interactive data exploration.

In the first experiment, we evaluate the effectiveness of recommendations for the two different tuple-weighting schemes described in Section 4.2, namely the *Binary* and the *Result* methods. Figures 2 and 3 show the inverse cumulative frequency distribution (inverse CFD) of the recorded precision and recall for the test sessions. (Recall that all sessions are used as test sessions, using the 10-fold cross validation methodology described earlier.) A point (x, y) in this graph signifies that x% of user sessions had precision/recall $\geq y$. For instance, as shown in Figure 3, the *Binary* method achieves a perfect recall (i.e., the recommendations cover all the tuples that the user covers with his/her last query) for more than half of the test sessions. We also observe that several test sessions have a precision and recall of 0, i.e., the recommendations did not succeed in predicting the intentions of the user. On closer inspection, these test sessions are very dissimilar to training sessions, and thus the framework fails to compute useful weights for S_0^{pred}.

Overall, we observe that both methods achieve similar precision. This means that both methods' recommended queries cover the same percentage of interesting tuples for the user. The *Binary* method, however, achieves much better results than the *Result* one in terms of recall. As previously mentioned, recall represents the number of recommended tuples that were of interest to the user with respect to the user's last query, and is a better predictor in terms of usefulness of the recommended query. This finding implies that the result size of the

query may not be a good indicator of the focus of users. Thus, in the experiments that follow, we report the results of the *Binary* weighting scheme only.

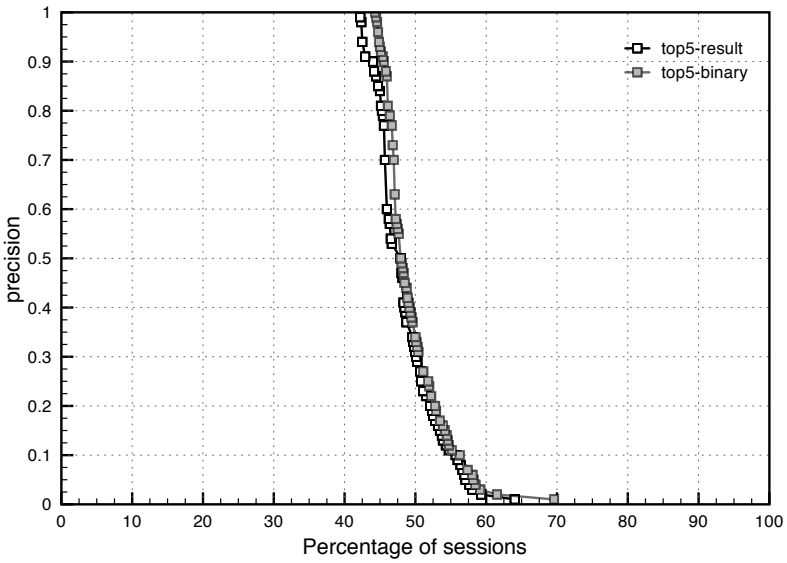

Fig. 2. Inverse CFD of precision for top-5 recommendations

In the next set of experiments, we compare the recommendations with regards to the size of the recommendation set. More specifically, in Figures 4 and 5 we compare the top-3 and top-5 recommendation sets in terms of precision and recall respectively. Both recommendation sets achieve good results, being 100% accurate in almost half test sessions. The top-5 recommendation set seems to be performing better than the top-3 one, both in terms of precision and recall. This can be justified by the fact that we report the maximum recall over all the recommended queries and compute the precision for the query that achieved maximum recall. This query might not always be included in the top-3 ones, but is very often included in the top-5 recommendations. Notably, in about 55% of the test sessions, the maximum recall was 1, meaning that the recommended query covered all the tuples that were retrieved by the user's original one.

Figure 6 shows the average recall and precision of all top-5 recommended queries. In this case we achieve high precision and recall for almost 1/3 of the test sessions. The lower average precision and recall for the remaining sessions means that some recommended queries might not be as accurate in covering the interesting subsets of the database, dragging the overall average down. In real-life applications, however, it is likely that the active user will be able to select the few recommendations closest to his/her interests. This motivates the use of the maximum recall metric, which is used in the experiments of Figures 4 and 5.

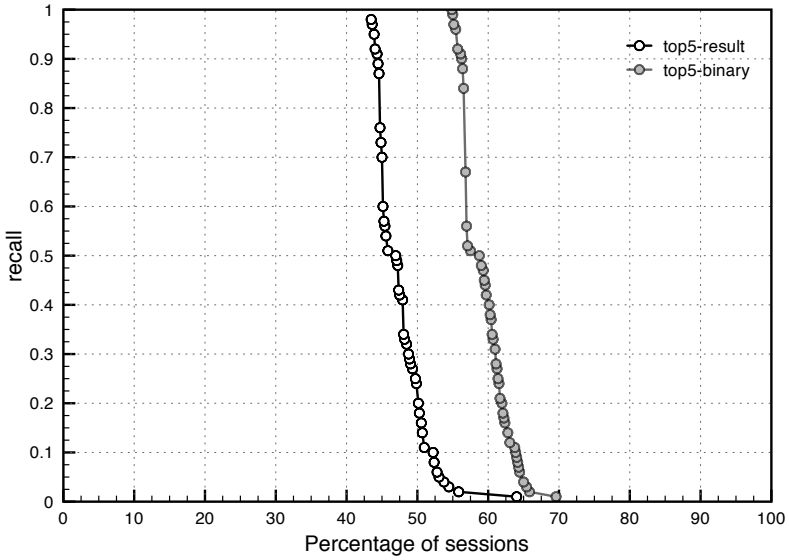

Fig. 3. Inverse CFD of recall for top-5 recommendations

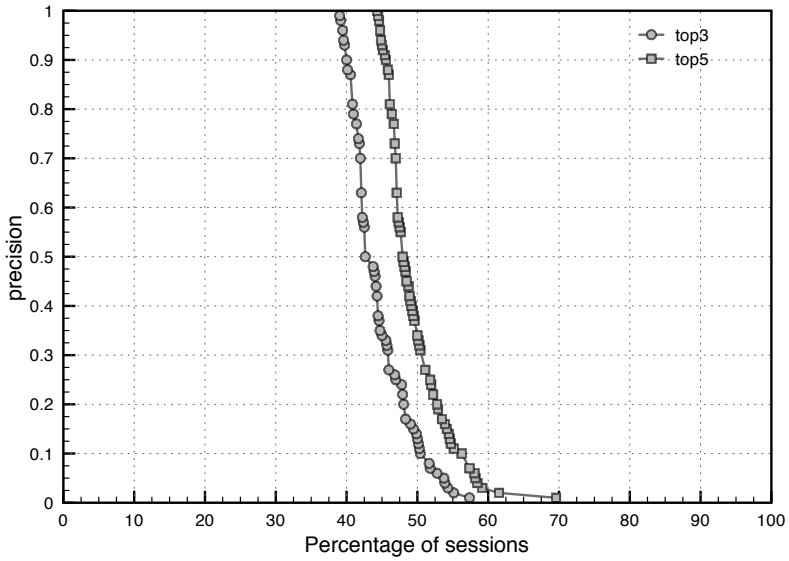

Fig. 4. Precision of top-3 and top-5 recommendations

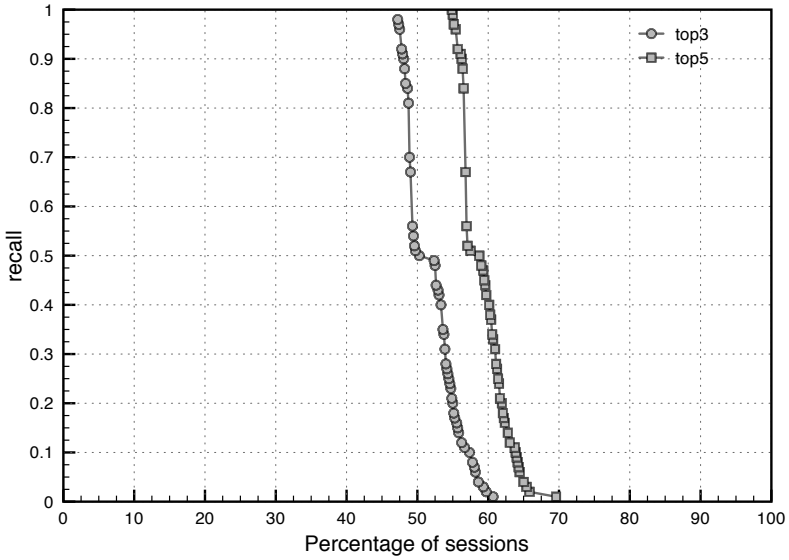

Fig. 5. Recall of top-3 and top-5 recommendations

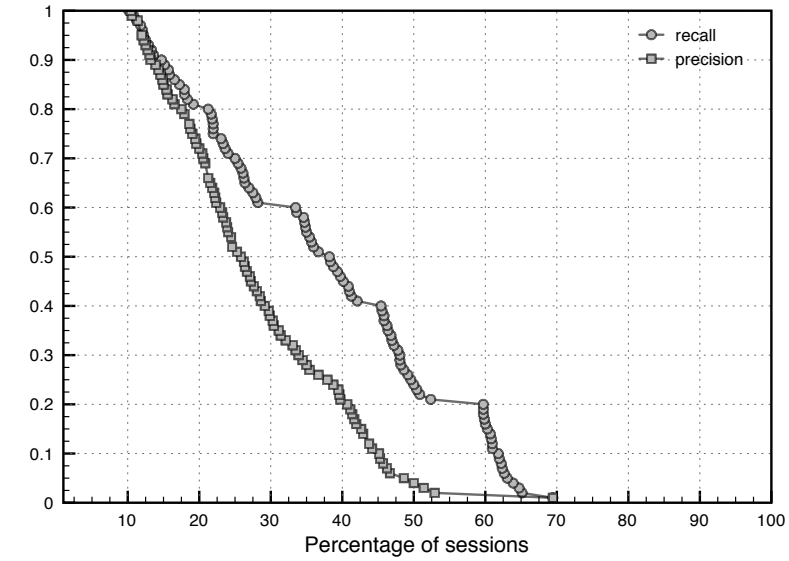

Fig. 6. Average precision and recall of top-5 recommendations

Next, we evaluate the effect of the mixing factor α (Equation 5). More specifically, we evaluate the precision and recall of the top-5 recommendations for the pure user-based collaborative filtering approach ($\alpha = 0$), the content-based filtering approach ($\alpha = 1$), as well as the case when both inputs are given equal importance ($\alpha = 0.5$). As shown in Figures 7 and 8, the pure collaborative filtering approach ($\alpha = 0$) yields worst results with respect to the other two approaches, in terms of both the precision and the recall of the recommendations. The comparison of the other two approaches ($\alpha = 0.5$ and $\alpha = 1$) shows that the combination of both sources ($\alpha = 0.5$) yields slightly better results in terms of recall. We should point out, however, that the results shown here are tightly connected to the specific data set and workload. In practice, we expect that α will be calibrated prior to deploying the recommendation algorithm, based on the characteristics of the database and a representative user workload.

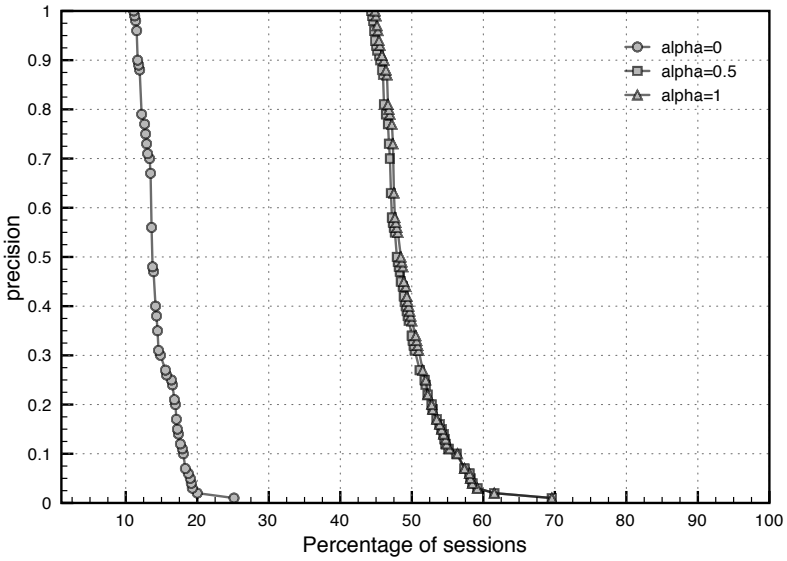

Fig. 7. Precision of top-5 recommendations for different α values

Finally, we present some examples of recommended queries for sessions in which the recommendations were 100% successful in terms of maximum recall and precision. Table 3 shows a description of the session's characteristics and the recommended query. The table lists recommendations for three user sessions, where the users had a very different progression in terms of the submitted queries. Our system was able to recommend a query that returned exactly the same results as the actual last query of the user, without the two queries being necessarily identical. This evidence demonstrates the usefulness of our approach in assisting users to interactively explore a relational database.

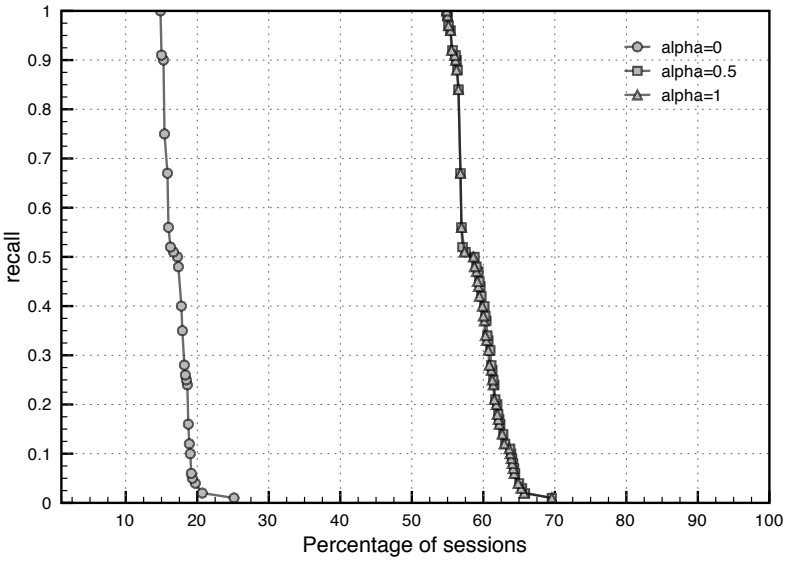

Fig. 8. Recall of top-5 recommendations for different α values

6 Conclusions

In this paper, we present a query recommendation framework supporting the interactive exploration of relational databases and an instantiation of this framework based on user-based collaborative filtering. The experimental evaluation demonstrates the potential of the proposed approach.

We should stress that this is a first-cut solution to the very interesting problem of personalized query recommendations. 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. Another interesting direction is to apply item-based collaborative filtering instead of the user-based approach of the current framework. We also intend to explore other approaches for instantiating the proposed conceptual framework.

We are also in the process of developing a visual query interface for the QueRIE system and plan to evaluate its performance using real users. To ensure that the system generates real-time recommendations for the active users of a database, we need to devise smart methods to compress the session-tuple matrix and to speed up the computation of similarities. In this direction, we plan to leverage randomized sketching techniques as a compression method [14, 15, 16].

Table 3. Query recommendations examples

Session description	**Recommended query**
Each consecutive query was posted to a different table.	SELECT * FROM PhotoZ WHERE objId = 0x082802f0c19a003e;
The user kept refining the same query adding exactly one selection predicate in every consecutive query.	SELECT p.ra, p.dec, s.z, s.ew, s.ewErr FROM specLine s, PhotoObj p WHERE s.specObjId = p.specObjid AND s.specLineId = 1549;
The user posted queries to the same tables, but each query had several selection clauses in addition to the previous one.	SELECT top 10 L1.height Halpha_h, L2.height Hbeta_h, L3.height OIII_h, L4.height NII_h, L1.sigma Halpha_sig, L2.sigma Hbeta_sig, L3.sigma OIII_sig, L4.sigma NII_sig FROM Specline L1, Specline L2, Specline L3, Specline L4, SpecObj WHERE SpecObj.SpecObjID = L1.SpecObjID AND SpecObj.SpecObjID = L2.SpecObjID AND SpecObj.SpecObjID = L3.SpecObjID AND SpecObj.SpecObjID = L4.SpecObjID AND SpecObj.specClass = 3 AND L1.lineID = 6565 AND L2.lineID = 4863 AND and L3.lineID = 5008 AND L4.lineID = 6585;

References

[1] Koutrika, G., Ioannidis, Y.: Personalized queries under a generalized preference model. In: ICDE '05: Proceedings of the 21st International Conference on Data Engineering. (2005) 841–852

[2] Adomavicius, G., Kwon, Y.: New recommendation techniques for multicriteria rating systems. IEEE Intelligent Systems **22**(3) (2007) 48–55

[3] Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Trans. on Knowl. and Data Eng. **17**(6) (2005) 739–749

[4] Bell, R., Koren, Y., Volinsky, C.: Modeling relationships at multiple scales to improve accuracy of large recommender systems. In: KDD '07: Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. (2007) 95–104

[5] Deshpande, M., Karypis, G.: Item-based top-n recommendation algorithms. ACM Trans. Inf. Syst. **22**(1) (2004) 143–177

[6] Greco, G., Greco, S., Zumpano, E.: Collaborative filtering supporting web site navigation. AI Commun. **17**(3) (2004) 155–166

[7] Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. **22**(1) (2004) 5–53

[8] Lee, H.J., Kim, J.W., Park, S.J.: Understanding collaborative filtering parameters for personalized recommendations in e-commerce. Electronic Commerce Research **7**(3-4) (2007)

[9] Mohan, B.K., Keller, B.J., Ramakrishnan, N.: Scouts, promoters, and connectors: the roles of ratings in nearest neighbor collaborative filtering. In: EC '06: Proc. of 7th ACM Conference on Electronic Commerce. (2006) 250–259
[10] Park, S., Pennock, D.M.: Applying collaborative filtering techniques to movie search for better ranking and browsing. In: KDD '07: Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. (2007) 550–559
[11] Khoussainova, N., Balazinska, M., Gatterbauer, W., Kwon, Y., Suciu, D.: A case for a collaborative query management system. In: CIDR '09: Proceedings of the 4th biennal Conference on Innovative Data Systems. (2009)
[12] Singh, V., Gray, J., Thakar, A., Szalay, A.S., Raddick, J., Boroski, B., Lebedeva, S., Yanny, B.: Skyserver traffic report - the first five years. Microsoft Research, Technical Report MSR TR-2006-190 (2006)
[13] X. Jin, Y. Zhou, B.M.: Task-oriented web user modeling for recommendation. In: UM'05: Proc. of 10th International Conference on User Modeling. (2005) 109–118
[14] Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. (1996) 20–29
[15] Cormode, G., Garofalakis, M.: Sketching streams through the net: distributed approximate query tracking. In: VLDB '05: Proceedings of the 31st international conference on Very large data bases. (2005) 13–24
[16] Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing. (1998) 604–613