July, 2014

QueRIE: Collaborative Database Exploration

Magdalini Eirinaki, *San Jose State University*
Suju Abraham, *Lucille Packard's Children Hospital*
Neoklis Polyzotis, *University of California, Santa Cruz*
Naushin Shaikh, *Data Domain, EMC*

QueRIE: Collaborative Database Exploration

Magdalini Eirinaki, Suju Abraham, Neoklis Polyzotis, Naushin Shaikh

Abstract—Interactive database exploration is a key task in information mining. However, users who lack SQL expertise or familiarity with the database schema face great difficulties in performing this task. To aid these users, we developed the QueRIE system for personalized query recommendations. QueRIE continuously monitors the user's querying behavior and finds matching patterns in the system's query log, in an attempt to identify previous users with similar information needs. Subsequently, QueRIE uses these "similar" users and their queries to recommend queries that the current user may find interesting. In this work we describe 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 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 SkyServer database and propose a scalable design that enables the incremental update of similarities, making real-time computations on large amounts of data feasible. Finally, we compare this fragment-based instantiation with our previously proposed tuple-based instantiation discussing the advantages and disadvantages of each approach.

Index Terms—[H.2.8d] Data Mining; [H.2.8h] Interactive data exploration and discovery; [H.2.8k] Personalization;

1 Introduction

Database systems provide the critical infrastructure to access and analyze large volumes of data in a variety of applications. Prominent examples include large-scale data warehouses that support business-intelligence tools, systems for ad-hoc analytics over big data, and services for scientific-data exploration, such as the Genome browser[1] or SkyServer[2], which allow scientists to query large databases of scientific data over a web-enabled interface.

Despite the availability of querying tools over large databases, users often have difficulties in understanding the underlying schema and formulating queries. For instance, the study on Hive[3], the data warehouse platform used in Facebook, mentions the following [1]: "A result of heavy usage has also lead to a lot of tables generated in the warehouse and this has in turn tremendously increased the need for data discovery tools, especially for new users." Similar issues appear in other domains, particularly in the domain of scientific data management (e.g., the Genome Browser and SkyServer services mentioned earlier), where users are not necessarily experts and the underlying schemas can be complicated. As a result, even when users have the ability to issue complex queries over large data sets, the task of knowledge discovery remains a big challenge: users may not be familiar with the database schema, may overlook queries that retrieve relevant data, or might not have the required expertise to formulate such queries. Moreover, due to the continuously growing size of the data, an exhaustive exploration of such databases is practically infeasible.

To address the important problem of assisting users in the interactive exploration of a large database, we designed the QueRIE[4] system. QueRIE assists users of ad-hoc or form-based query environments by presenting them with personalized query recommendations. The recommended queries are relevant to the user's information needs and can be submitted directly or be further refined. In other words, the user can use them as "templates" for query formulation instead of having to compose new ones.

QueRIE is built on a simple premise that is inspired by Web recommender systems: If users A and B have posed similar queries, then the other queries of B may be of interest to user A and vice versa. In other words, we can recommend the queries of user B in order to help user A in their exploration of the database. In particular, we propose to implement this idea through Collaborative Filtering, a well known, mature technique that has been used in Web recommender systems. However, the transfer

- *M. Eirinaki is with the Computer Engineering Department, San Jose State University, San Jose, CA, USA.*
- *S. Abraham was with the Computer Engineering Department, San Jose State University, San Jose, CA, USA when this project was developed. She is currently affiliated with Lucille Packard's Children Hospital, Palo Alto, CA, USA.*
- *N. Polyzotis is with the Computer Science Department, University of California, Santa Cruz, CA, USA.*
- *N. Shaikh was with the Computer Engineering Department, San Jose State University, San Jose, CA, USA when this project was developed. She is currently affiliated with Data Domain, EMC, Santa Clara, CA, USA.*

[1] http://genome.ucsc.edu
[2] http://cas.sdss.org
[3] http://hive.apache.org
[4] standing for Query Recommendations for Interactive data Exploration

of this approach to the database context introduces several technical challenges. First, SQL is a declarative language, and hence syntactically different queries may reflect the same information need. Consider for example, the two queries Q1: SELECT A FROM R WHERE B = 10; and Q2: SELECT R.A FROM R JOIN S ON (R.C = S.C) WHERE R.B = 10;. If relations R and S have a key/foreign key relationship on attribute C, then both queries retrieve the same results. This complicates greatly the computation 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 simply compare SQL queries - we essentially have to solve the notoriously difficult query-equivalence problem. A second important challenge raises from the absence of an explicit rating system for the queries posed by the user - how do we know which queries are important in the computation of user similarity? Finally, the recommended queries need to be intuitive so that the user can understand and refine if necessary. Too "synthetic" queries might result being even more confusing for the user.

QueRIE addresses these challenges by employing a closed-loop approach. Specifically, the QueRIE framework decomposes each query into basic elements that capture the essence of the query's logic. These elements are used to compute similarities between users, as well as a signature of the user's querying behavior (and, to some extent of the user's information needs). Recommendations are generated by mining queries from the system log that match well with the signature. Hence, the user is presented with queries that match her querying behavior, and are likely to be more intuitive than purely synthetic ones.

In our previous work we outlined the QueRIE framework, and the application of user-based collaborative filtering using witness tuples to represent user queries [2], [3]. In this paper, we provide a comprehensive presentation of QueRIE, including an overview of our previous work (tuple-based instantiation), and presenting the details of a different instantiation (demonstrated in [4]) which includes 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 propose a scalable design that enables the incremental update of similarities, making real-time computations on large amounts of data feasible. We then show, through experimentation, that the proposed method generates meaningful and more accurate recommendations on real-life traces from the SkyServer database when compared to the scalable extension of the tuple-based instantiation.

The remaining of this paper is organized as follows: In Sections 2 and 3 we provide a brief overview of the abstract framework of the QueRIE system, and the tuple-based instantiation respectively. We then present the fragment-based instantiation in Section 4 and discuss some implementation details in Section 5. The experimental evaluation of the fragment-based approach, along with a comparison to the tuple-based approach is presented in Section 6. We discuss the related work in Section 7 and conclude the paper with our plans for future work in Section 8.

2 The QueRIE framework

This section discusses an abstract framework for generating query recommendations. The abstract framework is essentially a workflow, as depicted in Figure 1. 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.

We consider a setting where users 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. For example, a real user session, belonging to the SkyServer query logs, is shown here:

Query 1:	*SELECT count(*) FROM region WHERE type like 'tiprimary'*
Query 2:	*SELECT count(distinct id) FROM region WHERE type like 'tiprimary'*
Query 3:	*SELECT id, count(*) FROM region WHERE type like 'tiprimary' GROUP BY id*
Query 4:	*SELECT id, count(*) FROM region WHERE type like 'tiprimary' GROUP BY id HAVING count(*)> 1*

This query pattern clearly corresponds to an interactive exploration of the database: the user starts by counting the number of tuples satisfying a predicate, then counts the distinct objects corresponding to these tuples, and eventually retrieves the objects that occur more than once. The pattern also indicates that the user is not familiar with the schema, e.g., the fact that attribute *id* may have the same value in several tuples. The user could save a lot of time if the system could recommend the appropriate query (Query 4 in the example above) right after their first attempt (Query 1). This is possible if a similar session already exists

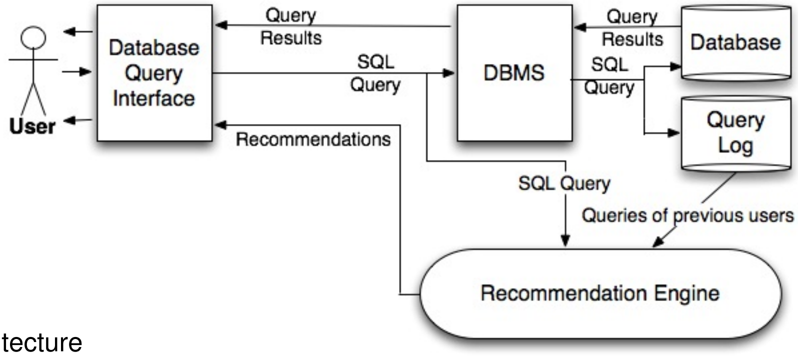


Fig. 1. QueRIE Architecture

in the query logs and thus can be used to generate recommendations for the active user. It is interesting to note here that we do not exclude from the recommendation set queries that have overlapping results with the ones the user has already posted since the system may miss good recommendations, as shown in the example above.

To simplify our presentation, we assume that each user has a single session with the database. This assumption can be lifted in a straightforward manner at the expense of more complicated notation. Given a user i, let Q_i denote the set of SQL queries that the user has posed so far in a single session. We introduce the notion of a *session summary* to summarize the characteristics of the queries posed in the session. 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. The different possibilities represent trade-offs between detail and conciseness, and as we will see later, also affect the quality of the generated queries. In what follows, we use S_i to represent the session summary for user i. User $i = 0$ will always represent the current user (for whom recommendations are generated), whereas $i = 1, \ldots, n$ represents past users of the system. In a slight abuse of notation, we use S_i to represent both the session summary and user i.

To generate recommendations for current user S_0, the framework first computes a "predicted" summary S^{pred}. This summary captures the predicted degree of interest of S_0 with respect to different query characteristics, including those that already appear in his/her queries, as well as new ones that have not been used yet. The summary S^{pred} is then used as the "seed" for the generation of recommendations. This two-step process is detailed below.

The predicted summary is defined as:

$$S^{\mathrm{pred}} = f(\alpha, S_0, S_1, ..., S_n). \tag{1}$$

f is a function that combines information from both the active user's summary S_0 and the summaries $S_1, \ldots, S_n$ of past users. The "mixing factor" $\alpha \in [0, 1]$ determines the importance of the S_0 (the current user's session) with respect to $S_1, \ldots, S_n$ (the sessions of other users). When $\alpha = 1$, S^{pred} takes into account only the queries in S_0, whereas $\alpha = 0$ has the opposite effect and only the queries of other users affect the recommendations. Neither of these extremes might be a good setting for all possible cases. Thus, we introduce α as a parameter of the system that can be tuned depending on the type of the database and the users' querying behaviors. It is interesting to contrast our approach to Web recommender systems, where the equivalent of S^{pred} is computed using $\alpha = 0$, i.e., based solely on information from other users (consider, for example, the case of recommending movies to users), or content-based recommender systems where $\alpha = 1$ (consider, for example, the case of generating playlists that may include songs the user likes as well as similar ones in terms of genre, vocals, instrumentation etc.).

Using the summary S^{pred}, the framework generates 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 can be performed in several ways. One approach would be to synthesize queries using the characteristics present in S^{pred}. However, this approach is not optimal for several reasons. For instance, it is important to recommend meaningful and intuitive queries. Thus the queries should have non-empty result sets, a property that needs to be verified before each recommendation. Moreover, unless there is some semantic knowledge about the data and/or the schema, the automatic synthesis of several characteristics to form an intuitive query is very difficult. To address these issues, we follow a different approach: we re-use queries that are included in the Query Log of the

TABLE 1
Notation Summary

m	number of users
Q_i	set of queries in session S_i
S_i	Session summary for user i / User i
$S_i[\tau]$	Importance of tuple τ in session S_i (tuple-based instantiation)
$S_i[\phi]$	Importance of fragment ϕ in session S_i (fragment-based instantiation)
S_0	Session summary for current user
S^{pred}	Predicted summary for current user
α	Mixing factor
S_Q	Single query vector of query Q
k	Number of top-ranked query fragments
n	Size of recommendations

DBMS. Of course, the selected queries are a good match to the characteristics described in S^{pred}, and this introduces a new challenge for our system. Such queries are expected to be intuitive, and easy to understand, since they correspond to queries formulated by other (human) users. Furthermore, we can verify easily that these queries return non-empty results, by examining the metadata in the system's query log.

Overall, our framework consists of the following components: (a) a model for session summaries, (b) a method to compute the session summaries $S_0, \ldots, S_n$, (c) a method to compute S^{pred}, and (d) a method to select queries based on S^{pred}. An interesting point is that the framework forms a closed loop, going from SQL queries to session summaries and back. Again, this design choice follows the fact that all user interaction with a relational database occurs through a declarative query language.

In what follows, we describe two instantiations of the abstract recommendations framework, namely a *tuple-based* recommendation engine and a *fragment-based* recommendation engine. We discuss the advantages and disadvantages of each approach in terms of efficiency and quality of recommendations. We also address the scalability problem of such systems, presenting some design enhancements that enable incremental updates and real-time computation for large data sets. A summary of the notation used throughout this paper is included in Table 1.

3 Tuple-based Query Recommendations

In this instantiation of the QueRIE framework, the session summary S_i is represented as a weighted vector, where every coordinate corresponds to a distinct database tuple. 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, and is non-zero only if τ is a witness for at least one query in the session. The intuition is that S_i captures the tuples in the base tables that are touched by the queries in the user's session. Hence, sessions that contain equivalent queries will map to the same summary.

We assume that the vector S_Q represents a single query Q. The value of each element $S_Q[\tau]$ signifies the importance of the tuple τ as the witness for Q. We consider two different weighting schemes for setting $S_i[\tau]$, a binary scheme and a result-based scheme:

Binary scheme.

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

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-based scheme.

$$S_Q[\tau] = \begin{cases} 1/|ans(Q)| & \text{if } \tau \text{ is a witness;} \\ 0 & \text{if } \tau \text{ is not a witness.} \end{cases} \tag{3}$$

Here $ans(Q)$ is the result-set of Q. The idea behind this scheme is similar to the IDF concept from information retrieval: 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 each query Q posed by user i, we define the session summary S_i as:

$$S_i = \sum_{Q \in Q_i} S_Q. \tag{4}$$

Using the session summaries of the past users, we can construct the $(n \times T)$ 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.

We compute S^{pred} as follows:

$$S^{\text{pred}} = \alpha \cdot S_0 + (1 - \alpha) \cdot \sum_{i=1,\ldots,n} sim(S_i, S_0) \cdot S_i, \tag{5}$$

where $sim(S_i, S_0)$ is a similarity metric between the two vectors (e.g., cosine similarity). This approach is inspired by Web recommender systems, where the idea is to bias the recommendations based on users who exhibit similar behavior to the current user. The difference is that we use the mixing factor α to blend in the behavior of the current user. Overall, S^{pred} yields a weight per tuple that corresponds to the importance of the tuple to the user's exploration.

Having computed S^{pred}, the algorithm recommends queries that retrieve tuples of high predicted weights. Specifically, for each candidate query Q[5], we compute the similarity $sim(S_Q, S^{\text{pred}})$. The few candidate

[5] We maintain a uniform random sample of past queries as our candidate pool.

queries with the highest similarity are returned as recommendations to the user. (The number of returned queries is a parameter of the framework.)

Overall, the tuple-based approach captures the user's querying behavior at a very fine level of detail – the individual witnesses to the user's queries. Moreover, it handles readily the issue of equivalent declarative queries, since the underlying witness sets are exactly the same. The downside is the increased complexity, since, in principle, the session summaries grow linearly with the size of the database. What is more, the similarities between the current user's session and those of previous users need to be calculated every time the active user submits a new query.

Fortunately, it is possible to implement this method more efficiently by employing randomized sketching techniques (e.g., AMS sketches [5] or min-hash sketches [6]) to compress the summaries and compute the similarity metrics. In fact, in the QueRIE prototype [3], [4] we have employed the MinHash probabilistic clustering technique that maps each session summary S_i to a "signature" $h(S_i)$ [6]. The Jaccard similarity between vectors is thus reduced to the similarity of their signatures: $JaccardSim(S_i, S_0) = sim(h(S_i), h(S_0))$. However, as shown in Section 6.3, this improvement in computational efficiency comes at the cost of loss of precision of the generated recommendations.

4 Fragment-based Query Recommendations

The fragment-based instantiation of the QueRIE framework works in a similar manner to the tuple-based one. The two main differences lie in the representation of the session summaries and the formulation of similarities. More specifically, the coordinates of the session summaries correspond to fragments of queries instead of witnesses. We identify as fragments the following syntactical features of the queries in the session: attribute references, tables references, join and selection predicates. At a high level, the idea behind this approach is to recommend queries whose syntactical features match the queries of the current user.

As discussed before, user-based collaborative filtering's main disadvantage is that it inherently requires real-time similarity calculations, as the active user's profile gets updated. This significantly slows the real-time generation of recommendations, making such a choice inappropriate for large-scale systems. On the other hand, item-based collaborative filtering performs all similarity calculations during the training process, and thus has much smaller overhead during the recommendations' generation phase. This is the reason why we decided to follow a methodology similar to the item-based collaborative filtering. Our objective is to identify fragments that co-appear in several queries posed by different users, and use them in the recommendation process. These fragments may, or may not include the ones in the user's active session S_0 depending on the value of the mixing factor α. 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 "rank" (i.e. importance) of each fragment with regards to the current user session. In turn, the highest ranked query fragments are the query characteristics used to mine the query logs and select the most relevant queries that are used as recommendations.

Formally, session summary S_i is a vector whose cell $S_i[\phi]$ contains a non-zero weight if the fragment ϕ appears in at least one query of the session. For a given fragment ϕ, we define a single query vector cell $S_Q[\phi]$ 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. Conceptually, the length of the vector is equal to the number of possible fragments, but we expect only few cells to have non-zero values. We consider two different weighting schemes, a binary scheme and a weighted scheme, both using the queries Q posed by user i:

Binary scheme.

$$S_i = \bigvee_{Q \in Q_i} S_Q. \tag{6}$$

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.

$$S_i = \underset{Q \in Q_i}{\ } S_Q. \tag{7}$$

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

The recommendation seed, modeled by S^{pred}, represents the estimated importance of each query fragment with regard to the active user's behavior. Similarly to the tuple-based instantiation, we again use the "mixing factor" α that allows us to include or exclude the fragments of the active user session in the recommendation process. However, instead of the costly session-tuple approach (similar to the user-item collaborative filtering), we employ a fragment-fragment approach (reminiscent of the item-item paradigm) to calculate the similarities. More specifically, we first define a fragment-similarity metric $sim(\rho, \phi)$ that evaluates the similarity of two fragments ρ and ϕ in terms of their corresponding weights in the session summaries $S_1, \ldots, S_n$. 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 (yet the framework can accommodate any metric). Using this metric, each coordinate $S^{\text{pred}}[\phi]$ is computed as follows:

$$S^{\text{pred}}[\phi] = \frac{\sum_{\rho \in R} S_0[\rho] * sim(\rho, \phi)}{\sum_{\rho \in R} sim(\rho, \phi)}, \qquad (8)$$

where R represents the set of top-k similar query fragments (k is a parameter of the framework). Intuitively, ϕ obtains a high weight if S_0 contains fragments that co-occur frequently with ϕ in the queries of past users.

The final step of generating recommendations is similar to that of the tuple-based approach: once the predicted summary S^{pred} has been computed, its similarity to each query summary S_Q is calculated, and the queries having the highest similarity to the active user's summary are returned as recommendations.

Another big advantage of item-to-item collaborative filtering is resilience to the cold start problem, since it is highly unlikely that a user's query will include only new fragments. In fact, this can happen only if none of the tables in the FROM clause have been referenced by any other query in the logs. Nonetheless, even in this rare case, QueRIE will be able to generate recommendations right after the second user query, as discussed in Section 5.2.

5 Discussion and implementation details

The fragment-based approach clearly captures information at a coarser level of detail, and hence it is expected to miss interesting correlations between users. For instance, two distinct selection predicates will be mapped to different fragments even if they are satisfied by the same tuples in the base tables. It is therefore expected that the basic tuple-based approach yields better results in terms of precision. This, however, comes with a cost; the tuple-based approach constructs large (and relatively dense) summaries and, most importantly, requires real-time calculations of the similarities between the session summary S_0 of the current user and these of past users. On the other hand, the big advantage of the fragment-based approach is that it can be implemented very efficiently; the space of fragments grows slowly allowing for a scalable system, the summaries are very sparse enabling faster similarity calculations and, most importantly, the fragment-to-fragment similarities can be computed offline and stored for very fast retrieval when recommendations need to be generated, leveraging all the advantages of item-to-item collaborative filtering [7]. A comparable response time is achieved when the tuple-based instantiation employs MinHash synopses. However, as shown in Table 7 this comes at the cost of prediction accuracy.

For the reasons above, we concentrated our efforts in optimizing the design of the fragment-based engine. In what follows, we discuss a few design decisions that enabled the implementation of a fast yet accurate query recommendation system.

TABLE 2
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

5.1 Query Preprocessing

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 relaxation process, we follow a simplified version of the framework proposed in [8]. 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. The trade-off of this relaxation process is increased recall vs. lower precision. We expect that the precision of the system would improve if a stricter representation of queries was followed. While this decision is orthogonal to the remainder of the framework, exploring its effect on the recommendation process is part of our future work plans.

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 2. For example, the fragments of Query 4 in Section 2 are: COUNT(*), REGION, REGION.TYPE PATMATCH, COUNT(*) COMPARE NUM.

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, but 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 ($s.x$ COMPARE NUM). 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.

5.2 Scalable similarity calculations

As discussed earlier, the fragment-based approach is much more efficient than the tuple-based one, since no online calculations are required. However, even the training of the model, that involves calculating all the fragment-to-fragment similarities, requires a significant amount of I/O and memory allocation, when the session summaries are stored in a database. To address this issue, we instead stored all the session summaries in a noSQL database. Such databases consist of hash tables that enable fast retrieval and calculations. The database includes the base hash table including the session summaries (fragments and counts indexed by the session id), and two derived ones: the inverted index of the base table (indexed by the fragment id), and the item similarity table (indexed by the fragment pair).

This design decision has clear advantages compared to storing the same information in a relational database. The key-value storage structure enables efficient calculation of the fragment-to-fragment similarities, without the I/O overhead or unnecessary scans of unrelated records. Moreover, this data structure is incrementally updated every time a new session summary needs to be added to the system: for each fragment in the session summary hash table, only the related records in the fragment hash table and the similarities of the related fragments are updated. If a fragment does not exist, the fragment and similarity hash tables are accordingly updated with new records. This real-time update of fragment similarities also addresses the cold start problem of recommender systems, as new fragments will be considered immediately for the subsequent recommendations (when the system is set to include the active user's fragments in the process).

This approach enables the real-time update of the query logs and fragment-to-fragment similarities, even on a stand-alone system. Of course, the above issue could also be addressed by parallelizing the process on a cloud. We leave this for future work.

5.3 The QueRIE system prototype

We implemented a prototype of our system that supports the two recommendation engines described previously. The prototype is implemented in Java and runs on top of a standard relational DBMS to store the query logs and a noSQL database to store the similarities (for the fragment-based engine)[6]. The database query interface module is built using HTML, JSP and JavaScript. The recommendation engine module is built using Java. The two modules interact through the JNI framework.

Once a user logs in the system, she is able to select one of the two recommendation engines. The user can author and submit a SQL query to SkyServer. QueRIE sends the request to the database, and presents the user with the results. At the same time, the system records the active user's queries, creating an implicit user profile. This user profile is used as input to the algorithm, along with the predictive model to generate real-time, personalized query recommendations. For each recommended query, the user is able to examine a sample of the results that will be retrieved, in order to decide whether it addresses her needs, prior to actually submitting it to the DBMS.

At all times, the active user is able to: (a) formulate a query from scratch, (b) select a recommended query and submit it as it is, or (c) select a recommended query and edit it before submitting it to the database. Moreover, the interface allows the user to browse the database schema, review and re-submit queries that were posed during her recent history, see more details on how the recommendations were generated, and change the various parameters of the framework. A snapshot of the QueRIE prototype is shown in Figure 2. The details of our system are described in [3], [4].

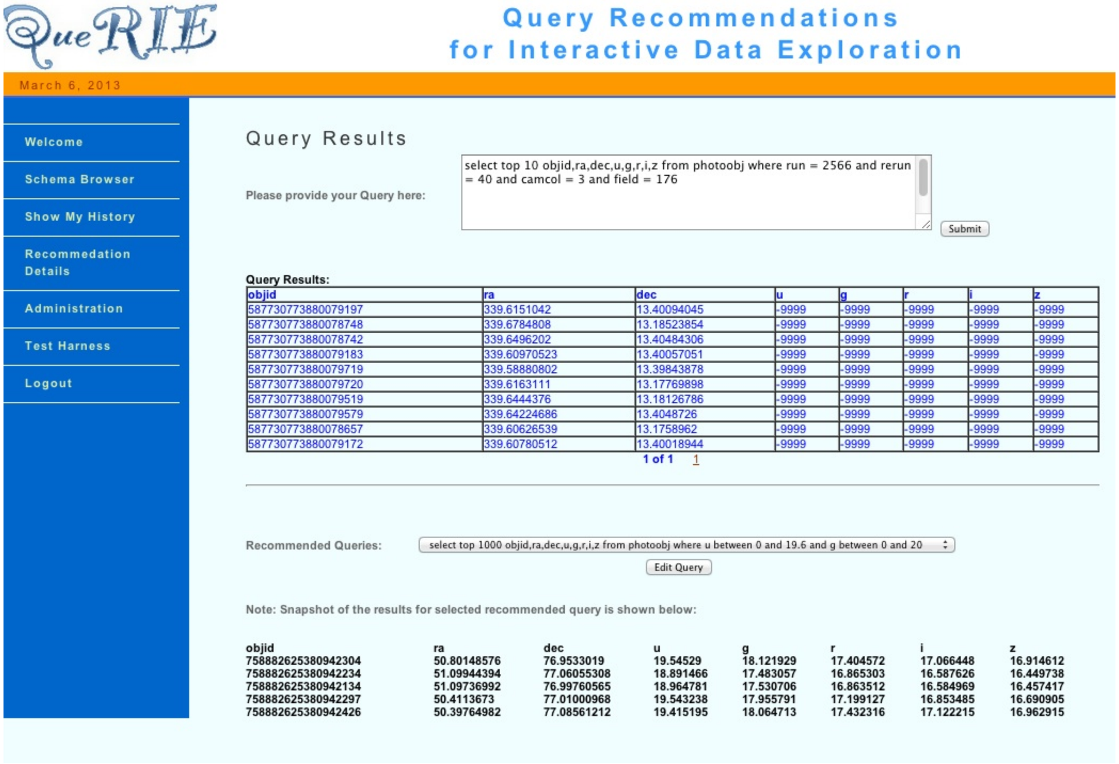
6 Experimental Evaluation

We evaluated our framework using traces of the Sky Server database[7]. The traces contain queries posed to the database between the years 2006 and 2008. We used the methods described in [9] to clean and separate the query logs in sessions[8]. The characteristics of the data set and the queries are summarized in Table 3. All the experiments were run on a 2.3 GHz Intel Core i5 processor with 2 cores and 4GB of RAM running OSX.

[6] The QueRIE prototype is using the MySQL and Voldemort databases respectively.

[7] We used version BestDR6.

[8] It should be noted that the proposed methods are not very reliable in classifying bots vs. mortals. Due to this fact, we expect that the reported results are slightly worse than what they would be if only human sessions were included in the training set. However, devising techniques for cleaning the SkyServer logs from bots is orthogonal to, and beyond the scope of this work.



Query Results

Please provide your Query here:

```
select top 10 objid,ra,dec,u,g,r,i,z from photoobj where run = 2566 and rerun
= 40 and camcol = 3 and field = 176
```

Query Results:

objid	ra	dec	u	g	r	i	z
587730773880079197	339.6151042	13.40094045	-9999	-9999	-9999	-9999	-9999
587730773880078748	339.6784808	13.18523854	-9999	-9999	-9999	-9999	-9999
587730773880078742	339.6496202	13.40484306	-9999	-9999	-9999	-9999	-9999
587730773880079183	339.60970523	13.40057051	-9999	-9999	-9999	-9999	-9999
587730773880079719	339.58880802	13.39843878	-9999	-9999	-9999	-9999	-9999
587730773880079720	339.6163111	13.17769898	-9999	-9999	-9999	-9999	-9999
587730773880079519	339.6444376	13.18126786	-9999	-9999	-9999	-9999	-9999
587730773880079579	339.64224686	13.4048726	-9999	-9999	-9999	-9999	-9999
587730773880078657	339.60626539	13.1758962	-9999	-9999	-9999	-9999	-9999
587730773880079172	339.60780512	13.40018944	-9999	-9999	-9999	-9999	-9999

1 of 1 1

Recommended Queries: select top 1000 objid,ra,dec,u,g,r,i,z from photoobj where u between 0 and 19.6 and g between 0 and 20

Note: Snapshot of the results for selected recommended query is shown below:

objid	ra	dec	u	g	r	i	z
758882625380942304	50.80148576	76.9533019	19.54529	18.121929	17.404572	17.066448	16.914612
758882625380942234	51.09944394	77.06055308	18.891466	17.483057	16.865303	16.587626	16.449738
758882625380942134	51.09736992	76.99760565	18.964781	17.530706	16.863512	16.584969	16.457417
758882625380942297	50.4113673	77.01000968	19.543238	17.955791	17.199127	16.853485	16.690905
758882625380942426	50.39764982	77.08561212	19.415195	18.064713	17.432316	17.122215	16.962915

Fig. 2. QueRIE interface after a query has been submitted

TABLE 3
SkyServer Query Logs - Data Set Statistics

Database size	2.6TB
# Sessions (training set)	412
# Sessions (test set)	45
# Queries	6713
# Distinct queries	4037
# Distinct witnesses	13,602,430
Avg # distinct query fragments	3212
Avg # non-zero pair-wise fragment similarities	60126
Avg. # queries per session	9.3
Min. # queries per session	4

TABLE 4
SkyServer Query Logs - Subset Statistics

# Sessions (training set)	140
# Sessions (test set)	20
# Distinct queries	1401
#Distinct witnesses	212,693
# Distinct query fragments	755
# Non-zero pair-wise fragment similarities	30436
Avg. # queries per session	8.8
Min. # queries per session	4

In what follows we first present an evaluation of the various parameters of the fragment-based approach. This small-scale experimental evaluation, using a subset of the above dataset (as described in Table 4) helped us choose the default values of the system's parameters. Using these, we then proceeded with more extensive experiments on the entire data set (described in Table 3). We present these results, along with an overview of the results of the tuple-based approach, in order to compare and discuss the trade-offs of the two instantiations. A detailed discussion of the results of the tuple-based instantiation can be found in [2]. We also present a comparison of the fragment-based approach with the MinHash extension of the tuple-based instantiation [3], briefly discussed in Section 3. Our results demonstrate that the fragment-based instantiation generates real-time recommendations with comparable accuracy to that of the baseline tuple-based instantiation (which is not fast enough to maintain an interactive user experience), and significantly outperforms its MinHash extension.

6.1 Evaluation of the system's parameters.

For this first set of experiments, we used a subset of the SkyServer data set. The characteristics of that data set and the queries are summarized in Table 4.

We performed several experiments evaluating the performance of the framework, and the effect of the various parameters of the algorithm. In this Section we present the most important findings in terms of the number of top-k fragments selected to calculate S^{pred}, and the weighting scheme. We omit the results for top-m and the mixing factor α since the results are in accordance to the ones we discuss in the following Section (where the entire dataset is used). Table 5 shows the default values kept constant for the remaining parameters in each case.

Methodology. In order to evaluate the various param-

TABLE 5
Default parameter values

Top-k fragments	5
Top-m recommendations	5
α	0.5
weighting scheme	weighted (cosine)

eters of QueRIE, we used the holdout set methodology [10]. 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 as shown in Equations 9 - 11. We keep the precision of the recommendation that had the maximum recall value, assuming that the end user will also select only one recommended query each time.

$$Precision = \frac{|F_r \cap F_u|}{|F_r|} \quad (9)$$

$$Recall = \frac{|F_r \cap F_u|}{|F_u|} \quad (10)$$

$$F - Score = \frac{2 * Precision * Recall}{Precision + Recall} \quad (11)$$

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 f-score over the 160 sessions of the data set.

Evaluation of the top-k parameter. QUERIE employs the top-k fragments of previous queries in order to generate recommendations, as discussed in Section 4. Figures 3 and 4 show the effect of the choice of k on the average precision and F-score for the recommendations. We notice that the accuracy of the recommendations increases, as expected, with the value of k. However, for very large values of k ($k > 10$), the accuracy starts decreasing again. This is completely justifiable, since when k 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 higher precisions for $k \in [5, 10]$ (0.75 and 0.8 for the two end points), whereas F-score is the same for both end points (0.75 and 0.76 respectively). Given the small difference in terms of accuracy and the fact that the lower the number of fragments k, the faster the real-time calculations, we adopt $k = 5$ as the default value for the framework.

Evaluation of the weighting scheme. The representation of the query and session vectors, and consequently the metrics used to calculate the similarities between fragments, differ depending on the weighting scheme. In our work 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 and 6 verify this intuition, with a precision (for max recall) of 0.74 and 0.84 for the binary and weighted schemes respectively, and an F-score of 0.72 and 0.82 respectively. For the specific dataset both schemes performed similarly in terms of execution time (needed 6 sec on average to generate recommendations). We adopt the *weighted* scheme as the default value, since it resulted in better prediction accuracy.

6.2 Prediction accuracy.

After determining the default values for k and the weighting scheme, as shown above, we proceeded to evaluate the performance of QueRIE in terms of accuracy for different values of the α parameter and the number of recommendations m. For this set of experiments, we employed the entire dataset, as shown in Table 3 and, when not stated otherwise, we set the parameters of the system to the ones shown in Table 5. We follow the same methodology that was used for evaluating the tuple-based approach (discussed in [2]) so that we can compare the two approaches.

Methodology. We employ 10-fold cross validation to evaluate the performance of the QueRIE 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. The effectiveness of each recommended query is measured against the L-th query of the test session, using the precision and recall metrics as defined in Equations 9 and 10. Following the practice of previous studies in recommender systems [11], 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 each set of recommendations.

Results. Figures 7 - 10 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 $\geq y$.

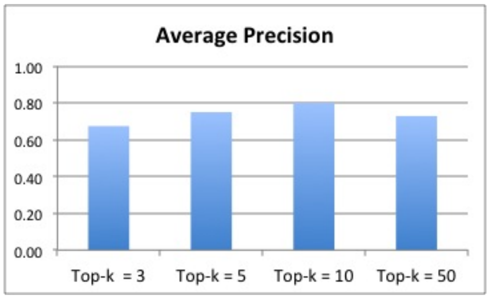


Fig. 3. Average precision for various top-k values

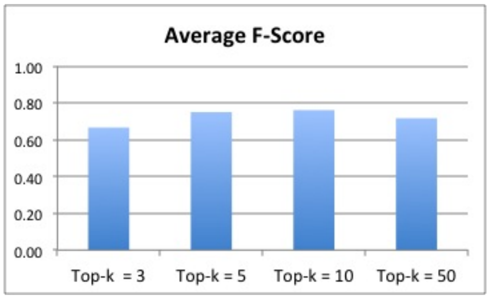


Fig. 4. Average f-score for various top-k values

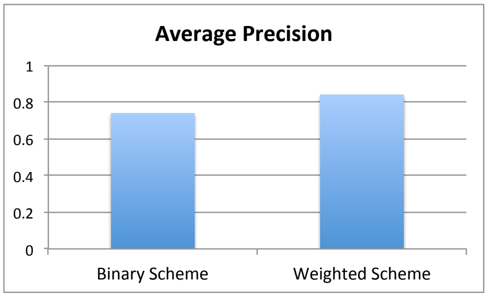


Fig. 5. Average precision for different weighting schemes

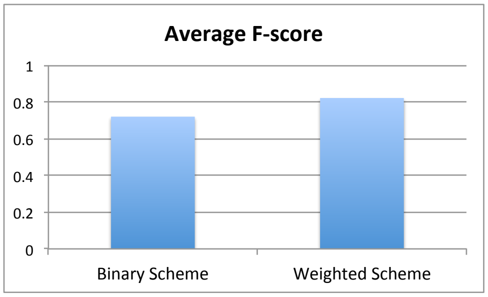


Fig. 6. Average f-score for different weighting schemes

We evaluate the results in terms of the mixing factor α, as well as the size of the recommendation set (top-m). Our first observation is that when the active user's session fragments are included ($\alpha = 0.5$), the precision and recall increase significantly independent of the size of the recommendation set. More specifically, we observe that, when $\alpha = 0.5$ the recommendations match exactly the next user query for more than 20% of the sessions, and the precision is acceptable (≥ 0.5) for more than half of the sessions. On the other hand, when we follow the pure collaborative filtering approach ($\alpha = 0$), precision and recall are overall lower for both sizes of the recommendation set. The findings verify our initial claim that database recommender systems are very different in nature from their web counterparts. As pointed out previously, 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 most users are interested in posting queries similar to the ones they have already posted during the same session.

Looking at the effect of the size m of the recommendation set, we cannot draw conclusions in favor of any of the two choices in the case of $\alpha = 0.5$. On the contrary, the system performs much better (in terms of precision for max recall) for $m = 5$ when $\alpha = 0$. We thus set the default parameters to our system as shown in Table 5.

Figures 11 and 12 show the average recall and precision among all top-3 and top-5 recommended queries respectively for $\alpha = 0.5$. In this case we achieve high precision and recall for more than 1/3 of the test sessions and acceptable precision for over 45% of the test sessions. The lower average precision and recall for the remaining sessions means that some recommended queries might not reflect the user's intentions at all, dragging the overall average down. Since the active user will be provided with a set of recommendations to choose from, we expect that she will instead opt for the ones closest to her interests.

6.3 Comparison and discussion

As a final step, we compare the fragment-based approach with the tuple-based one presented in [2] and its extension using MinHash synopses presented in [3]. As discussed previously, the tuple-based approach represents information at a very fine level of detail and thus yields much more accurate results. More specifically, as shown in Figure 13, the tuple-based approach achieves perfect precision for 45% of the sessions (as compared to 21% for the fragment-based approach, as shown in Figures 7 and 9). It is interesting, however, that overall more recommendations achieved acceptable precision (≥ 0.5) in the fragment-based approach (52% and 48% of sessions for the fragment- and tuple-based approaches respectively). A similar pattern is being observed when we evaluate the average precision and recall of the two instantiations. As shown in Figures 12 and 14, while a few more recommendations achieve perfect precision and recall in the tuple-based instantiation (8% and 10% respectively), the overall number of recommendations that achieve average precision ≥ 0.5 drops to 25% for the tuple-based instantiation whereas remains around 40% for the fragment-based one.

Even though the tuple-based approach captures the user's information needs at a finer level of detail, this comes at the cost of real-time computational efficiency,

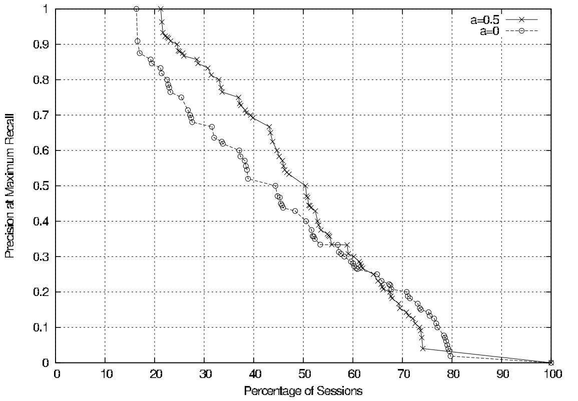


Fig. 7. Inverse CFD of Precision at Maximum Recall (top-3 recommendations)

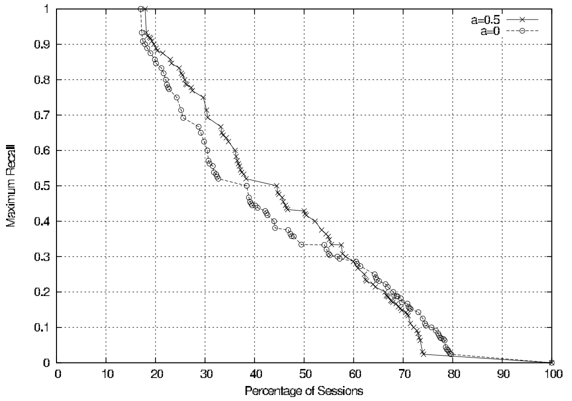


Fig. 8. Inverse CFD of Maximum Recall (top-3 recommendations)

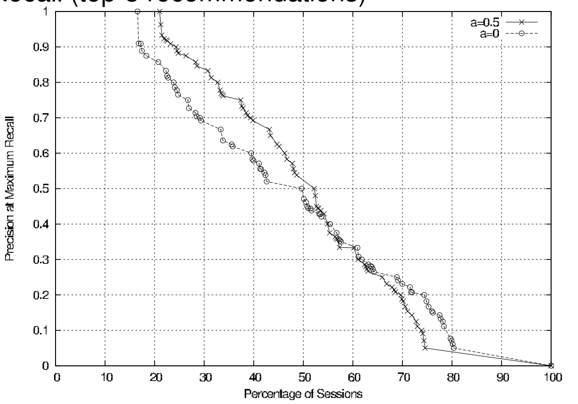


Fig. 9. Inverse CFD of Precision at Maximum Recall (top-5 recommendations)

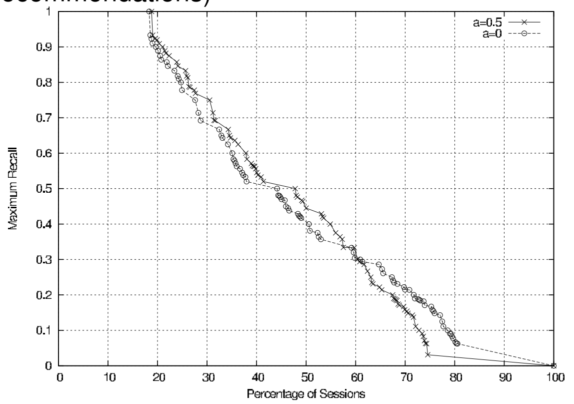


Fig. 10. Inverse CFD of Maximum Recall (top-3 recommendations)

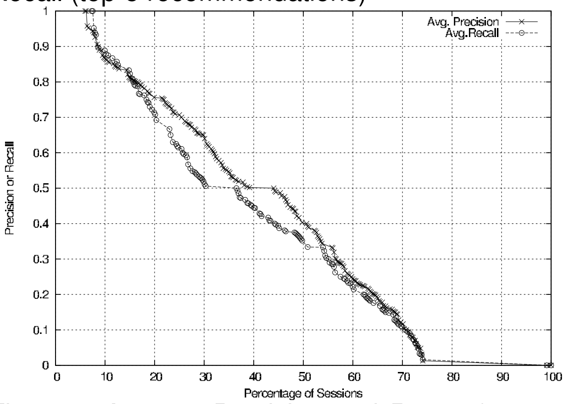


Fig. 11. Average Precision and Recall ($m = 3$, $\alpha = 0.5$)

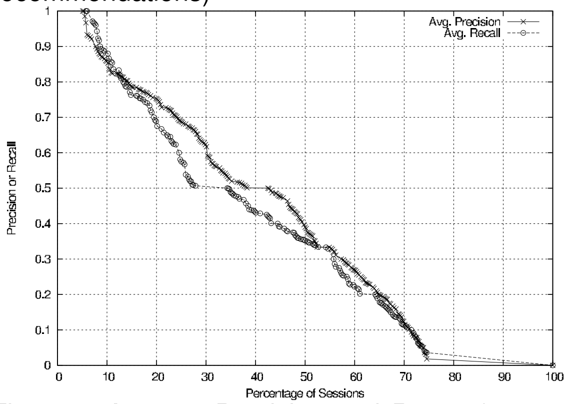


Fig. 12. Average Precision and Recall ($m = 5$, $\alpha = 0.5$)

as shown in Table 6; using the base tuple-based implementation in a real-time recommender system is prohibitive due to the time needed to generate each recommendation (several minutes). This was our motivation to extend the tuple-based approach by using the MinHash synopses. MinHash synopses reduce the time of recommendations from several minutes to 6 seconds (same as the fragment-based approach), but we expect a significant drop in the accuracy of the system due to the fact that the similarities between sessions are reduced to similarities between their hashes. We used the dataset of Table 4 to measure the prediction accuracy of the two instantiations. We present some results (for top-5 recommendations and $\alpha = 0.5$) in Table 7. We observe that the fragment-based engine outperforms the tuple-based/MinHash engine in terms of prediction accuracy and, given that the real-time response of the system is the same (6 sec), it is the optimal choice for the particular problem, as our initial intuition suggested.

7 Related Work

Even though the problem of generating personalized recommendations has been broadly addressed in the Web context [7], [12], only a few related works exist in the database context. A relevant line of research

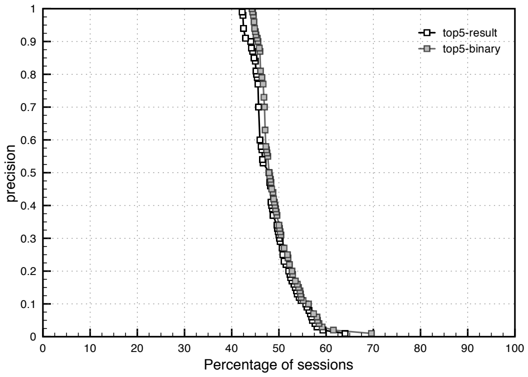


Fig. 13. Inverse CFD of Precision at Maximum Recall ($m = 5$, $\alpha = 0.5$) [2]

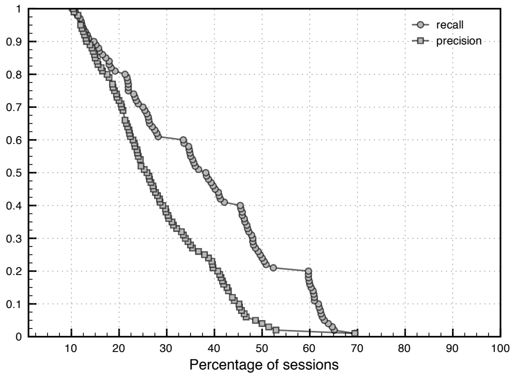


Fig. 14. Average Precision and Recall ($m = 5$, $\alpha = 0.5$) [2]

TABLE 6
Trade-offs between the three framework instantiations

	Fragment-based	**Tuple-based**	**Tuple-based using Min-Hash synopses**
Information representation	coarse	detailed	coarse
Phase 1: Session summaries' formation	Offline	Offline	Offline
Phase 2: Similarities - $S^{\textbf{pred}}$ calculation	Offline ($sim(\rho, \phi)$) - Online	Online ($sim(S_i, S_0)$) - Online	Online ($sim(h(S_i), h(S_0))$) - Online
Phase 3: Retrieval of similar queries	Online	Online	Online
Time to generate recommendations after query is posted	< 6 sec	> 5 min	< 6 sec

TABLE 7
Comparison of tuple-based/MinHash and fragment-based instantiations

	Fragment-based	Tuple-based with MinHash synopses
% **of sessions with precision** $= 1$	50%	35%
% **of sessions with recall** $= 1$	55%	40%
% **of sessions with precision** ≥ 0.5	85%	60%
% **of sessions with recall** ≥ 0.5	75%	65%
% **of sessions with average precision** ≥ 0.5	65%	40%
% **of sessions with average recall** ≥ 0.5	45%	50%

has to do with preference-aware query answering [13], an important problem in the context of web databases. Web databases provide a keyword-based query interface, and suffer from the "empty-answer" and "too-many-answers" problems. In that context, it's critical to provide the correct (personalized) answer to each user rather than the exact one to everyone for the same query [14]. Preferences may be expressed qualitatively, by embedding into relational query languages a special operator [15], [16], [17], [18], or quantitatively, by re-ranking or filtering the results (tuples) of the original query [19]. Lately context has also been added in such systems, as an additional personalization parameter [20], [21]. The common denominator of these works with ours is that all can be categorized under the "query personalization" area. However, our research mainly focuses on databases that serve a very specific purpose and have an SQL-based interface. In this context, rather than restructuring/enhancing the submitted query, or by re-ranking the retrieved relevant tuples (i.e. personalize the results), our objective is to assist the user exploring different parts of the database by providing new queries as recommendations and let the user decide on whether these are useful/relevant or not.

A multidimensional query recommendation system is proposed in [22], [23]. In this work, the authors propose a framework for generating 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 [24], where the authors outline the architecture of a collaborative query management system targeted at large-scale, shared-data environments. As part of this architecture, the authors suggest that data mining techniques can be applied to the query logs in order to generate query suggestions, without providing any technical details on how such a recommendation system could be implemented.

A few recent works propose frameworks for query recommendations over relational databases [25], [26], [27], [28], [29]. In [25], 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 do they perform recommendations in the traditional "personalized" form (i.e. finding similarities among users or items).

In [26], 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. The ideas presented in this position paper are extended in [28], where the authors present the ReDRIVE framework. The authors introduce the notion of interestingness of (attribute, value) pairs called faSets, appearing in the result of the original user query. In this work, the authors present efficient ways of calculating the interestingness of each pair by maintaining statistics of the database allowing them to estimate the frequencies related to it. While the ReDRIVE framework has some common characteristics with the tuple-based instantiation of QueRIE in that both consider the results (tuples) of the active user's query they differ in terms of the approach employed; the authors find interesting faSets (and subsequently use them to expand the user's queries to retrieve additional items as recommendations) considering the content of the database and the user's query results. Thus, contrary to our hybrid approach that employs query fragments and uses the past users' queries, ReDRIVE is following a content-based approach.

In [29] the authors propose SnipSuggest, a system meant to assist users when formulating SQL queries. Using the information in query logs, the system generates a DAG representing the relationships between different clauses in the queries. Based on the current user's query, the system ranks the outgoing edges and recommends the nodes/clauses of the most similar ones. This work differs in ours in several ways. SnipSuggest recommends possible additions to various clauses in the current user's query, and not complete queries. Moreover each query is treated independently of any previous one, even if they belong to the same user session.

Finally in [27] the authors propose the FlexRecs framework that enables the execution of various workflows that allow a user to customize the recommendation process and results by selecting the type of recommendation and the basis of similarity calculations on various aspects. The focus of this work is to create a framework that will assist users with highly diverse information needs - e.g. University students whose needs span from selecting their major to deciding which course to take next semester. However, FlexRecs is built on the premise that multiple recommender systems will exist that can be selected as part of each workflow. Thus, all aforementioned works including ours, are complementary in that they can serve as underlying recommendation engines for this type of framework.

Overall, these previous studies indicate the importance of the emerging problem of generating query recommendations. They also corroborate our previous observations on the challenges that need to be addressed in order to develop an effective solution.

8 Conclusions

In this paper we present the QueRIE framework that aims to generate useful SQL query recommendations to users of relational databases. Taking into consideration the findings of our previous work, where we developed a tuple-based instantiation of the framework using user-based similarities to generate recommendations, we decided to follow an item-based approach using query fragments to represent user sessions. As expected, representing information at a coarser level of detail, resulted in some loss of accuracy in our predictions. On the other hand, the fragment-based approach can be implemented very efficiently; the space of fragments grows slowly, the summaries are very sparse and, most importantly, the fragment-to-fragment similarities can be computed offline and stored for very fast retrieval when recommendations need to be generated. The experimental results showed that this trade-off between computational efficiency and accuracy is worth pursuing, since we are able to have a scalable implementation running with real, "big" data, with an acceptable loss in precision. In fact, when the tuple-based instantiation employs approximation techniques (MinHash synopses) to enable real-time calculations, the loss in precision is much greater than that of the fragment-based one.

This concludes the first part of our work in this research area. There are many interesting directions we would like to explore in the future. We would like to measure the impact the query relaxation process has in the quality of recommendations. Exploring a sequence-based approach is another interesting direction for future work, but it requires a careful reconsideration of several aspects of our framework. For instance, pure sequence information may not be sufficient to discover user similarities. Instead, we may have to consider the relative changes between queries in the sequence, e.g., that selection predicates becomes more selective as queries progress, in order to properly detect similarities. We also plan to focus on relational databases that have a form-based interface. While the fragment-based approach seems as a straightforward selection for such environments, new challenges related to the formulation of session similarity, the synthesis of recommendations and their presentation arise. Finally, as we aim at developing a more generic and scalable system, we are currently working on integrating alternative techniques for generating recommendations, such as matrix factorization methods.

References

[1] A. Thusoo, J. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy, “Hive - a petabyte scale data warehouse using hadoop,” in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, March, pp. 996–1005.

[2] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis, “Collaborative filtering for interactive database exploration,” in *Proc. of the 21st Intl. Conf. on Scientific and Statistical Database Management (SSDBM ’09)*, 2009.

[3] S. Mittal, J. S. V. Varman, G. Chatzopoulou, M. Eirinaki, and N. Polyzotis, “QueRIE: A Recommender System supporting Interactive Database Exploration,” in *IEEE Intl. Conf. on Data Mining series (ICDM’09) - in ICDM’10 proceedings because of editor’s error*, 2009.

[4] J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. S. V. Varman, “SQL QueRIE Recommendations,” in *Proc. of the 36th Intl. Conf. on Very Large DataBases (VLDB 2010)*, 2010.

[5] N. Alon, Y. Matias, and M. Szegedy, “The space complexity of approximating the frequency moments,” in *STOC ’96: Proc. of the 28th annual ACM symposium on Theory of computing*, 1996.

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

[7] G. Linden, B. Smith, and J. York, “Amazon.com recommendations: item-to-item collaborative filtering,” *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, Jan/Feb 2003.

[8] N. Koudas, C. Li, A. K. H. Tung, and R. Vernica, “Relaxing join and selection queries,” in *Proc. of the 33nd Intl. Conf. on Very Large DataBases (VLDB ’06)*, 2006, pp. 199–210.

[9] 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.

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

[11] B. M. X. Jin, Y. Zhou, “Task-oriented web user modeling for recommendation,” in *Proc. of User Modeling ’05*, 2005.

[12] B. Mobasher, *The Adaptive Web: Methods and Strategies of Web Personalization*, ser. LNCS. Springer, Berlin-Heidelberg, 2007, vol. 4321, ch. Data Mining for Personalization, pp. 90–135.

[13] K. Stefanidis, G. Koutrika, and E. Pitoura, “A survey on representation, composition and application of preferences in database systems,” *ACM Trans. Database Syst.*, vol. 36, no. 4, 2011.

[14] G. Koutrika, “Personalized dbms: an elephant in disguise or a chameleon?” *IEEE Data Engineering Bulletin*, vol. 34, no. 2, June 2011.

[15] S. Borzonyi, D. Kossmann, and K. Stocker, “The skyline operator,” in *IEEE Intl. Conf. on Data Engineering (ICDE)*, 2001.

[16] J. Chomicki, “Preference formulas in relational queries,” *ACM Trans. Database Syst.*, vol. 28, no. 4, pp. 427–466, 2003.

[17] W. Kiessling, “Foundations of preferences in database systems,” in *Intl. Conf. on Very Large DataBases (VLDB)*, 2002.

[18] W. Kiessling, M. Endres, and F. Wenzel, “The Preference SQL System - An Overview,” *IEEE Data Engineering Bulletin*, vol. 34, no. 2, June 2011.

[19] G. Koutrika and Y. Ioannidis, “Personalized queries under a generalized preference model,” in *Proc. of the 21st Intl. Conf. on Data Engineering (ICDE 2005)*, 2005.

[20] J. Levandoski, M. Mokbel, and M. E. Khalefa, “FlexPref: A Framework for Extensible Preference Evaluation in Database Systems,” in *IEEE Intl. Conf. on Data Engineering (ICDE)*, 2010.

[21] E. Pitoura, K. Stefanidis, and P. Vassiliadis, “Contextual database preferences,” *IEEE Data Engineering Bulletin*, vol. 34, no. 2, June 2011.

[22] A. Giacometti, P. Marcel, and E. Negre, “Recommending Multidimensional Queries,” in *Proc. of the 11th Intl. Conf. on Data Warehousing and Knowledge Discovery (DaWaK’09)*, 2009.

[23] A. Giacometti, P. Marcel, E. Negre, and A. Soulet, “Query recommendations for olap discovery driven analysis,” *Intl. Journal on Data Warehousing and Mining*, vol. 7, no. 2, 2011.

[24] 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 Conf. on Innovative Data Systems Research (CIDR 2009)*, 2009.

[25] X. Yang, C. M. Procopiuc, and D. Srivastava, “Recommending join queries via query log analysis,” in *25th Intl. Conf. on Data Engineering (ICDE 2009)*, 2009, pp. 964–975.

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

[27] G. Koutrika, B. Bercovitz, and H. Garcia-Molina, “Flexrecs: Expressing and combining flexible recommendations,” in *SIGMOD Conference*, June 2009, pp. 745–757.

[28] M. Drosou and E. Pitoura, “Redrive: result-driven database exploration through recommendations,” in *CIKM*, 2011, pp. 1547–1552.

[29] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu, “SnipSuggest: Context-Aware Autocompletion for SQL,” *PVLDB*, vol. 4, no. 1, 2011.

Magdalini Eirinaki is an Assistant Professor at the Computer Engineering Department, San Jose State University, California. Her research interests cover the areas of web mining and recommendation systems and, in particular, on personalization, interactive database exploration and mining of social networks. She has published several papers in refereed journals and international conference proceedings in the above areas. She received her Ph.D. degree in Informatics (Computer Science) from Athens University of Economics and Business in 2006.

Suju Abraham received the B.Tech degree in Electronics and Communication Engineering from TKM College of Engineering, Kollam, India, in 1993, and the MS degree in Software Engineering from San Jose State University, California, in 2010. While pursuing her MS degree, she co-authored papers in the area of interactive database exploration (under the name Suju Koshy). She is currently an Applications Developer at Lucille Packard’s Children’s Hospital. Her interests are developing mobile and other software applications with a specialization in the medical field.

Neoklis Polyzotis is an Associate Professor at UC Santa Cruz. His research focuses on on-line database tuning, scaling machine learning to large data, and declarative crowdsourcing. He is the recipient of an NSF CAREER award in 2004 and of an IBM Faculty Award in 2005 and 2006. He has also received the runner-up for best paper in VLDB 2007 and the best newcomer paper award in PODS 2008. He received his PhD from the University of Wisconsin at Madison in 2003.

Naushin Shaikh received the B. Eng. Degree in Computer Science from University of Pune, India, and the MS in Sofware Engineering from San Jose State University, California, in 2011. She is currently a Senior QA Engineer at Data Domain, EMC, Santa Clara, California.