The QueRIE system for Personalized Query Recommendations

Gloria Chatzopoulou[1] Magdalini Eirinaki[2] Suju Koshy[2] Sarika Mittal[2]
Neoklis Polyzotis[3] Jothi Swarubini Vindhiya Varman[2]

[1] Vanderbilt University [2] San Jose State University [3] UC Santa Cruz

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. We discuss the key components of QueRIE and describe empirical results based on actual user traces with the Sky Server database.

1 Introduction

Database systems are becoming increasingly popular in the scientific community, as tools to access and analyze large volumes of scientific data. Prominent examples include the Genome browser[1] that hosts a genomic database, and SkyServer[2] that stores large volumes of astronomical measurements. Such databases are typically accessed through a web interface that allows users to pose queries through forms or in some declarative query language (e.g., SkyServer users can submit SQL queries directly).

Despite the availability of querying tools, users of these systems may still find it challenging to discover interesting information. Specifically, users may not know which parts of the database hold useful information, may overlook queries that retrieve relevant data or might not have the required expertise to formulate such queries. An exhaustive exploration of the database is also practically impossible, due to the continuously growing size of the data. To address this issue, we designed the QueRIE[3] system that assists users in the interactive exploration of a large database. The core idea is to present a user with personalized query recommendations, which are relevant to the user's information needs and can serve as "templates" for query formulation. The user is able to directly submit or further refine these queries, instead of having to compose new ones.

QueRIE is built on a simple premise that is inspired by Web recommender systems: If a user A has similar querying behavior to user B, then they are likely interested in retrieving the same data. Hence, the queries of

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

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

user B can serve as a guide for user A. One obvious approach to realize this idea is to leverage well known collaborative filtering techniques, which were popularized in Web recommender systems. However, this transfer introduces several technical challenges. First, the use of declarative queries implies that users may retrieve the same data through queries that are syntactically different. This complicates greatly the computation of user similarity, since it is not trivial to simply compare the corresponding queries – we essentially have to solve the notoriously difficult query-equivalence problem. A second challenge is the absence of an explicit rating system for the queries posed by a user – how do we know which queries are important in the computation of user similarity? Finally, it is important to recommend queries that are intuitive, i.e., queries that the user can understand and refine if necessary. A user may not find a recommendation useful if the query is too "synthetic".

QueRIE addresses these challenges by employing a closed-loop approach. Specifically, QueRIE decomposes each query into basic elements that capture the essence of the query's logic. These elements are used to compute similarities between users, and also to compute 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 that are likely to be more intuitive than purely synthetic queries.

2 Related Work

Even though the problem of generating personalized recommendations has been broadly addressed in the Web context [11], only a handful of related works exist in the database context. Some work has been done in the area of personalized recommendations for keyword or free-form query interfaces [12]. In this scenario, a user queries a database using keywords or free-form text and the personalization system recommends items of interest. Our approach is different from this scenario because it aims to assist users who query relational databases using either ad-hoc or form-based queries. Also, our framework recommends queries instead of "items" from the database. Finally, QueRIE does not require from the users to explicitly declare their preferences beforehand in order to generate recommendations.

A multidimensional query recommendation system is proposed in [6, 7]. In this work the authors address the related problem of generating recommendations for data warehouses and OLAP systems. In this work, the authors propose a framework for generating 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 [9], where the authors outline the architecture of a collaborative query management system targeted at large-scale, shared-data environments. As part of this architecture, they suggest that data mining techniques can be applied to the query logs in order to generate query suggestions. The authors present a general outline of a framework for query recommendations pointing out that this is a challenging process. However, they do not provide any technical details on how such a recommendation system could be implemented.

Two recent works propose frameworks for query recommendations using the information recorded in the query logs [13, 8]. In [13], the authors propose a query recommender system that represents the past queries using the most frequently appearing tuple values. Then, after predicting which new tuples might be of interest to the end user, they reconstruct the query that retrieves them. This approach works better with relations that have discrete attribute values, contrary to scientific databases, where most attributes are numeric. The authors also propose a global ranking of the queries, based on the statistics of the database and not the query logs. Both approaches are evaluated in a preliminary empirical study, yet no discussion on scalability issues is provided. In [8] 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.

3 An Abstract Framework for Query Recommendations

This section discusses an abstract framework for generating query recommendations. The abstract framework is essentially a workflow that takes as input the queries of previous users and the queries of a current user, and outputs query recommendations. We discuss two possible instantiations of the workflow in the next section.

We consider a setting where users explore a relational database through SQL queries. Specifically, each user interacts with the database in the form of *sessions*, where each session consists of a sequence of SQL queries. We expect the queries in the same session to be correlated, as we assume that the user is trying to mine useful information from the data in an exploratory fashion, i.e., the formulation of the next query in the session is highly dependent on the results of previous queries. 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 queries that the user has posed thus far in a single session. We introduce the notion of a *session summary* to summarize the characteristics of the queries posed in the session. Contrary to Web recommender systems, where a session summary can be readily constructed as the items that a user visits/rates/purchases, there exist several ways to model a session of SQL queries. 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.

Our conceptual framework generates recommendations for S_0 in two steps. First, the framework computes a "predicted" summary S^{pred} that captures the importance of different query characteristics for user S_0 . Note that S^{pred} may contain characteristics that already appear in the queries of S_0, but also new ones that the user has not used yet. The summary S^{pred} is used in the second step of the recommendation workflow, as the "seed" to generate recommendations. We further detail the two steps below.

The predicted summary is computed as $S^{\text{pred}} = f(\alpha, S_0, S_1, ..., S_n)$, where f is a function that combines information from summaries $S_0, S_1, \ldots, S_n$. Clearly, the function will depend on the specific model for session summaries. Parameter α is a mixing factor, which controls the importance of S_0 (the current user's session) with respect to $S_1, \ldots, S_n$ (the sessions of other users). Recall that S^{pred} includes query characteristics that already appear in the queries of S_0. The reason is that we want to recommend queries that extend or restructure the queries already posed by S_0, in addition to completely different (yet relevant) queries. By setting $\alpha = 0$, we ignore completely the queries in S_0, whereas $\alpha = 1$ has the exact opposite effect, i.e., only the queries in S_0 affect the recommendations. We have found in our experiments that neither of these two extremes are satisfactory, which justifies the introduction of α in our framework. 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. (As an example, it is not meaningful to recommend to a user a movie that they have already watched.)

Summary S^{pred} is used to generate the queries that are presented to the user as recommendations. One

approach is to synthesize queries based on the characteristics present in S^{pred}, but this is not likely to work well for several reasons. First, it is important to recommend queries that have non-empty result sets, and hence it will be necessary to verify this property for every candidate recommendation. Second, it is extremely difficult to synthesize intuitive queries that combine specific characteristics, unless there is some semantic knowledge about the data and/or the schema (which is difficult to acquire in itself). Our approach is completely different: we mine the query log of the DBMS to select queries that have a good match to the characteristics described in S^{pred}. The recommendations are likely 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[4].

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 feature is that the framework forms a closed-loop, receiving SQL queries as input and generating SQL queries as output. This design follows from our starting assumption that users interact with the database through a declarative query language.

4 Tuple-based and Fragment-based Query Recommendations

We now describe two instantiations of the abstract recommendation framework, namely a *tuple-based* recommendation engine and a *fragment-based* recommendation engine. As discussed below, the two instantiations represent a trade-off between efficiency and quality of recommendations.

Tuple-Based Recommendation Engine. The session summary S_i is represented as a weighted vector, where every coordinate corresponds to a distinct database tuple. The weight $S_i[\tau]$ represents the importance of a given tuple τ in the session of user 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 session. Hence, sessions that contain equivalent queries will map to the same summary.

We consider two schemes for setting $S_i[\tau]$ for a witness τ: a frequency-based scheme, where $S_i[\tau]$ essentially corresponds to the number of queries that have τ as the witness; and a result-based scheme, where $S_i[\tau]$ is inversely proportional to the result sizes of the queries for which τ is a witness. The idea behind the last scheme is similar to the IDF concept from information retrieval–τ obtains a higher weight if it is a witness to queries with small results.

We compute S^{pred} as follows: $S^{\text{pred}} = \alpha \cdot S_0 + (1-\alpha) \cdot \sum_{i=1,\ldots,n} sim(S_i, S_0) \cdot S_i$, 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 that exhibit similar behavior to the current user. The difference is that we use the mix-in 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}, we output as recommendations the queries of past users whose witnesses match the high-weight tuples in S^{pred}. Specifically, for each candidate query Q (we maintain a sample of past queries as our candidate pool), we compute the similarity $sim(S_Q, S^{\text{pred}})$, where S_Q is the summary of a session that comprises solely query Q. The few candidate queries with the highest similarity metric are returned as the recommendation 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. Fortunately, it is possible to implement

[4]One obvious concern is that a query may not return any results if the database has been updated. However, this scenario is rather unlikely for scientific data sets, which are typically append-only.

this method efficiently by employing randomized sketching techniques (e.g., AMS sketches [3] or min-hash sketches [5]) to compress the summaries and compute the similarity metrics with a small loss in precision.

Fragment-Based Recommendation Engine. The fragment-based approach works similarly, except that 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.

Formally, a 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. 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. There are several possibilities for the weighting scheme, and we consider two in our work: binary, which indicates the presence of ϕ, and frequency-based, which assigns to $S_i[\phi]$ the count of queries that contain the fragment.

Under this model, S^{pred} corresponds to a vector that indicates the relevance of query fragments to the current user's exploration. Similar to the tuple-based engine, we compute S^{pred} as a blend of the user's summary S_0 and other users' summaries $S_1, \ldots, S_n$ weighted according to some similarity metric. The difference is that we employ a fragment-fragment approach, which is reminiscent of the item-item paradigm of collaborative filtering systems on the Web. Specifically, we first define a fragment-similarity metric $sim(\phi, \rho)$ that evaluates the similarity of the two fragments in terms of their corresponding weights in the session summaries $S_1, \ldots, S_n$. We use Jaccard's coefficient and cosine similarity for the binary and frequency-based schemes respectively, but our framework can accommodate any metric. Using this metric, each coordinate $S^{\text{pred}}[\phi]$ is computed as a blend of $S_0[\phi]$ and the expression $\sum_{\rho \in R}(S_0[\rho] \cdot sim(\rho, \phi)) / \sum_{\rho \in R} sim(\rho, \phi)$, where R is the set of k fragments (k is a parameter of our framework) that have the highest similarity to ϕ. Intuitively, ϕ obtains a high weight if S_0 contains fragments that co-occur frequently with ϕ in the queries of past users.

The generation of the recommended queries occurs in a similar fashion as the tuple-based approach: each candidate query Q is mapped to a summary S_Q, and the queries that have the highest similarity to S^{pred} are returned as recommendations.

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. The big advantage is that it can be implemented very efficiently, as the space of fragments grows slowly, the summaries are very sparse and the fragment-to-fragment similarities can be computed offline and stored for very fast retrieval when recommendations need to be generated. (This offline preprocessing is not possible in the tuple-based approach, since the similarity metric involves the complete vector S_0 of the current user.)

5 The QueRIE System Prototype

We implemented a prototype of our system that supports the two recommendation engines described in the previous section. The prototype is implemented in Java and runs on top of a standard relational DBMS. The details of our system are described in [10, 1].

Using the prototype, we evaluated the efficacy of our recommendation techniques using real user traces from the SkyServer database. Our results demonstrated that it is possible to generate meaningful query recommendations for a large set of users in the traces. Specifically, our experiments showed that: (a) using $\alpha = 0.5$ produces the best results in terms of recommendation quality, (b) the tuple-based engine generates perfectly precise recommendations (precision is equal to one) for 40% of the users in the test set, (c) the fragment-based engine is noticeably less effective in terms of precision, and (d) the fragment-based engine has much less computational overhead. A detailed discussion of the results can be found in [4, 2]

6 Conclusions and Future Work

Query recommendations can provide valuable guidance for interactive database exploration, particularly for users who lack the expertise to formulate complex queries or who are not familiar with the data. The proposed QueRIE system aims to provide a generic framework where we can develop and evaluate recommendation techniques for this novel domain.

As part of our future work, we intend to investigate a hybrid of the two recommendation engines that achieves a better trade-off between efficiency and precision. We also plan to extend our techniques to form-based query interfaces that are common for Web-accessible databases. Our fragment-based approach is readily applicable for single-form interfaces, but we wish to examine the case of multi-form interfaces where the query is formulated in several steps.

References

[1] J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. Swarubini V. Varman. SQL QueRIE Recommendations (demo paper). In *Proc. of the 36th International Conference on Very Large Data Bases (VLDB '10)*, 2010.

[2] J. Akbarnejad, M. Eirinaki, S. Koshy, D. On, and N. Polyzotis. SQL QueRIE Recommendations: a query fragment-based approach. In *4th International Workshop on Personalized Access, Profile Management, and Context Awareness in Databases (PersDB '10)*, 2010.

[3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (STOC '96)*, 1996.

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

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

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

[7] A. Giacometti, P. Marcel, E. Negre, and A. Soulet. Query Recommendations for OLAP Discovery Driven Analysis. In *Proc. of the ACM 12th International Workshop on Data Warehousing and OLAP (DOLAP'09)*, 2009.

[8] N. Khoussainova, Y.C. Kwon, M. Balazinska, and D. Suciu. SnipSuggest: Context-Aware Autocompletion for SQL. *PVLDB*, 4(1), 2011.

[9] Nodira Khoussainova, Magdalena Balazinska, Wolfgang Gatterbauer, YongChul Kwon, and Dan Suciu. A case for a collaborative query management system. In *Proc. of the 4th Biennal Conference on Innovative Data Systems Research (CIDR '09)*, 2009.

[10] S. Mittal, J. S. Vindhiya Varman, G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. QueRIE: A Recommender System supporting Interactive Database Exploration (demo paper). In *In Proc. of the 2010 edition of the IEEE International Conference on Data Mining series (ICDM '10)*, 2010.

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

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

[13] Kostas Stefanidis, Marina Drosou, and Evaggelia Pitoura. "You May Also Like" Results in Relational Databases. In *3rd International Workshop on Personalized Access, Profile Management, and Context Awareness in Databases (PersDB '09)*, 2009.