A survey on Personalized Query Recommendation System for Database Exploration

Shiny Nair, Varghese S Chooralil

Abstract— An effective information mining process necessitates an extensive exploration of the database. Analysis and study of large volume of data available in data marts are inevitable for knowledge discovery. For example, for scientific exploration, scientists need to query large databases for scientific data. However, all users may not possess the expertise in Structured Query Language that is generally required to query relevant data from the database. Considering complex relational databases, seldom will novice users have the knowledge and expertise of the underlying schema of the database and association between relations and attributes. The QueRIE system that supports users by presenting personalized query recommendations. The QueRIE framework identifies similarities with previous users' information needs and recommends queries to the current user. This paper studies the different algorithms and systems for query recommendations with focus on a QueRIE framework instantiation that attempts to identify similar queries as recommendations to the user, based on a set of query fragments from users' session.

Index Terms— Data mining, fragment based, interactive data exploration, personalization, query fragments, query recommendation, tuple based.

1 INTRODUCTION

INTERACTIVE database exploration is a key task in information mining. There is a plethora of data available in database systems today for analysis and research. To extract knowledge from such huge volumes of data, users should be adept at querying databases for data that is relevant and precise. Structured Query Language is the standard language to query relational databases. However, users are not always skilled to write complex queries. Complex schemas, relationships between tables, ambiguous table and column names are major concerns among others, especially for an inexperienced user. Adhoc requests for information in databases are common in the industry and scientific community. However, if users do not have adequate knowledge about the data and its significance, then there is a risk of missing out on relevant data or potentially motivating information. Therefore, this calls for assistance for such users to present them with query recommendations. Query templates can be issued to the user that can be modified and submitted as per the requirement.

The QueRIE system presents personalized query recommendations to users based on previous session summaries. The users' querying behavior is analyzed to identify similar queries and matching patterns from past sessions and presented as recommendations to the current user. This is based on the principle that if two users exhibit similar querying behavior, then it is likely that they will be interested in accessing similar data. The objective of the QueRIE system therefore, is to assist users in finding relevant information in interactive database exploration. Although in certain disciplines, users are allowed to get access to data through web-enabled querying tools or interfaces, an extensive exploration of complex databases may not be feasible because of the aforementioned reasons. As data is continuously increasing and more tables are created making the schema more complex, users are faced with the challenge of extracting the relevant information. QueRIE attempts to generate query suggestions to the users and enables users to either directly submit or refine the queries rather than constructing a new query.

QueRIE is built on the simple premise based on the Web Recommender systems: If user A and user B both like the same book X, then if A likes the book Y, then it is likely that B will like the book Y as well. Similarly, if A and B both query data X, and if A queries data Y, then it is quite possible that B will be interested in data Y.

An approach to realize this concept is to leverage the collaborative filtering methods used in the Web recommender systems. However, the direct transfer of this paradigm to the database context poses serious challenges. SQL being a declarative language, it is possible to retrieve the same data using different SQL queries that are structurally different. It is possible to obtain identical query result set using multiple syntactically varying queries. Therefore, direct evaluation of SQL among users will be complicated and may not result in useful recommendations because of the query-equivalence problem. As an example, consider the queries: SELECT X.A FROM X JOIN Y WHERE X.A=Y.A AND X.A = 10; SELECT A FROM X WHERE A =10; Regardless of the structure or syntax of the SQL, both these queries will result in the same result set if there is a foreign-key relation on the attribute A between relations X and Y. User queries are not explicitly rated and this poses the second challenge. As there is no rating available for

- *Shiny Nair is currently pursuing Master of Technology degree program in Computer Science & Engineering in M.G. University, Rajagiri School of Engineering & Technology, Kerala, India. E-mail: nairshiny@gmail.com*
- *Varghese S Chooralil is is currently working as Assistant Professor, Department of Computer Science and Engineering, M.G. University, Rajagiri School of Engineering & Technology, Kerala, India. E-mail: varghesesc@rajagiritech.ac.in*

queries, it becomes difficult to ascertain the importance of a query in computing recommendations. Finally, the suggested queries should be intuitive such that the user can understand, modify and refine as required. Queries that are too "synthetic" tend to confuse the user and may not prove useful to the user.

QueRIE handles these issues by adopting a closed loop approach. The system accepts query from user, fragments the query into basic elements and computes similarities based on a signature of the user's querying behavior. Systems logs are used for mining queries and presenting recommendations matching the query pattern.

2 Related Work

Amazon.com recommendations: Item-to-item collaborative filtering (G. Linden, B. Smith, and J. York) [5] says that an item to item collaborative filtering technique can be used for personalizing the customer's online store. The Amazon online store is customized for individual customers based on the customer's preferences, interests, purchase patterns etc. Rather than restricting recommendations based on the items an individual purchases or explicitly rates, other features like customer interests, list of favorite items, demographic details can also be considered as important attributes that contribute to valuable recommendations. Amazon.com leverages these recommendations for various marketing and advertisements campaigns on its various websites' pages.

Collaborative technique is the widely adopted technique for generating web-based recommendations. This technique focuses on collecting, studying and analyzing huge amounts of data on customers' purchase behavior, preferences etc. and generating recommendations based on the similarity with other customers. Thus, items that have been liked and purchased by users in the past will influence the items that Amazon.com suggests to the current user. There are a number of challenges in web recommendation algorithms. The main challenge concerns with handling large volumes of data and generating real-time recommendations that is useful and relevant. There will be an obvious lack of information for new customers and a plethora of information available for old customers. Finally it has to be noted that the customer data is volatile. Every interaction of the customer provides additional information and should be taken into account right away in making interactive suggestions.

There are several approaches to solve recommendation problems. In the traditional collaborative filtering algorithm, recommendations are based on few customers who are most similar to the user. If N is the number of distinct items in the catalog, then an individual customer is represented as an N-dimension vector of items. The similarity between two users X and Y can be measured considering the cosine of the angle between the two corresponding vectors. Cluster modeling is another technique to generate recommendations that divides the customer base into segments. The task is thereby treated as a classification problem. Segments are generated, summarized and represented as vectors based on which the user's similarity is computed. Search-based or content-based method considers the problem as a search task for similar items. Similar keywords, subjects, personalities, authors etc. are used to construct search queries to suggest popular items as recommendations.

In item-to-item collaborative filtering, instead of relating the user to similar customers, items that a user purchases or rates are matched to similar items. A recommendation list is then compiled based on the most similar items. This approach is based on a similar-items table that is build by considering items that users incline to purchase together.

Contrary to the traditional methods, the item to item collaborative filtering scales independently of the number of users and product catalog items. The expensive similar table can be computed offline. This algorithm produces recommendations that are interactive, of good quality and scales to large data sets.

Personalized DBMS: An elephant in disguise or a chameleon? (G. Koutrika) [6] mentions Web databases that provide a keyword-based interface suffer from challenges like "too-may-answers" and "empty-answer". An empty answer can result if the query conditions are restrictive. On the contrary, relaxed filter conditions can result in many rows that satisfy the query. To alleviate this problem, user preferences can be incorporated into queries that will support alternative choices and priorities for the users thus contributing to the preference-aware query model. Preferences can be represented both qualitatively and quantitatively. Qualitative preferences are expressed using binary preference relations that compare tuples relatively. Quantitative preferences assign scores to specific tuples or query conditions using Scoring function. A tuple with a higher score is preferred over the other. If preferences are expressed qualitatively using preference relations, then they are blended into relational query languages through an operator (like skyline) that selects from its argument relation the set of the most preferred tuples according to a given preference relation. On the other hand, if preferences are specified quantitatively using scores, then tuples are assigned scores, and the answer of a query with preferences is defined as the ranked set of top-n results.

There are three possibilities in implementing preference-aware query processing: (i) query translation, (ii) special evaluation algorithms for preference operators implemented on top of the DBMS, and (iii) native implementation, specified inside the database engine. The first two are plug-in approaches. The query translation process involves the following steps: (i) query rewriting wherein the preferences are embedded as standard query conditions in the original query resulting in a set of new queries, (ii) materialization where the new queries are executed, and (iii) aggregation, that combines the partial results into a single ranked list. The properties and optimization levels of the new operators may vary depending on whether a qualitative or quantitative approach is adopted. If the operators are very generic, it may limit the range of optimization and may weigh down the benefits of a generic implementation and a careful design of the query optimizer is required. Embedding preferences inside the DBMS is a giant leap towards a personalized database system that will help to retrieve customized answers.

In FlexPref: A framework for extensible preference evaluation in database systems (J. Levandoski, M. Mokbel, and M. E. Khalefa) [7], context is included in Web database systems as

an additional parameter in preference evaluation. Preferences can be integrated inside or on top of databases. This supports the implementation of varied range of applications from personalized database systems to decision-making tools. The common existing approach for preference evaluation in database system is the on-top technique. In this approach, the DBMS is considered as a black box to the preference method. The preference evaluation method (like Top-K, Skyline etc.) is entirely decoupled from the database. The DBMS is not aware of the semantics of the preference method. Although this is a simple approach, there are several performance limitations related to the query and internal operations optimization. An improved approach is the built-in approach wherein the preference evaluation method is tightly coupled with the query processor. Database operations (like selection, join etc.) are customized for each preference method. This is a more efficient approach as the extensive work of evaluating, ranking etc. is injected inside the DBMS. However, several lines of code are required to implement and incorporate existing as well as future preference methods and therefore very little work is done on this.

The FlexPref framework adopts a centralized approach for preference evaluation that is both simple and efficient. The technique is simple as introducing a new preference method requires registration to three functions alone. These functions demonstrate the essence of the preference evaluation method. The approach is efficient as once the evaluation method is built with the core of the database system, it "lives" in the system, coupled with the database engine and query processor. This plug and play paradigm enables efficient execution. Consider, two data objects X and Y and a set of preferred objects S, then the three general functions that have to be implemented to enable registration of a new preference evaluation method are (i) PairwiseCompare(Object X, Object Y) – The function updates the score of X and returns 1 if Y can never be a preferred object, -1 if X can never be a preferred object and 0 otherwise. (ii) IsPreferredObject(Object X, PreferenceSet S) – The function returns true if X is a preferred object and can be included in S, false otherwise. (iii) AddPreferredToSet(Object X, PreferenceSet S) – The function includes X in S and rearrange or remove objects from S, if required.

These functions modularize the preference evaluation operations and at the same time do not require being aware of the query processor specifics, that results in efficient execution of preference queries.

A case for a collaborative query management system, (N. Khoussainova, M. Balazinska, W. Gatterbauer, Y. Kwon, and D. Suciu) [8] emphasizes the need for a query recommendation framework and outlines the architecture of a collaborative query management system for large-scale, shared-data environments. Traditional trial-and-error method and pre-defined query templates for constructing queries prove expensive in massive data sets. To support users to formulate queries, data mining techniques can be used to evaluate query logs to provide query recommendations leveraging knowledge from past queries issued by the current or other users.

The system architecture of the proposed collaborative query management system (CQMS) includes the CQMS client that provides four interaction modes (Traditional Interaction, Search and Browse Interaction, Assisted Interaction and Administrative Interaction) that communicates with the CQMS server through both standard SQL queries and meta-queries.

The CQMS server is build on top of a DBMS and comprises four components as shown in Fig. 1. The Query Profiler and the Meta-Query Executor are online components. The Query Profiler receives standard SQL queries as input, logs the queries in the Query Storage and forwards them to the DBMS. The component performs complex pre-processing of the queries like extracting and storing query features and logs statistics about query execution and samples from its query results.

The Meta-query Executor handles all queries over the Query Storage. The CQMS client generates these queries through the Search and Browse and Assisted Interaction modes. This component also looks into administrative requests such as access control configurations. The Query Miner and Query Maintenance are background components. The Query Miner analyzes the query storage. It performs tasks such as clustering queries based on similarity etc. Its objective is to extract useful information from the query log. It runs periodically in order to maintain up-to-date information. The query log is automatically maintained by the Query Maintenance component. There are no technical details provided in the paper, however on the implementation details of such a recommendation system.

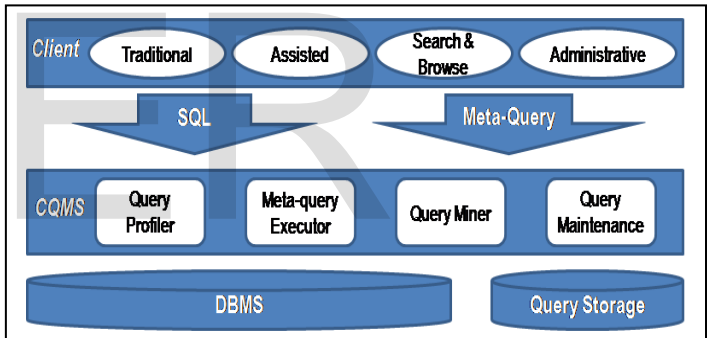

Fig. 1. CQMS System Architecture

FlexRecs: Expressing and combining flexible recommendations, (G. Koutrika, B. Bercovitz, and H. Garcia-Molina) [9] defines a framework that implements a high-level parameterized workflow including traditional relational operators and new operators that supports flexible and useful recommendations over structured data. FlexRecs implements a framework to provide recommendations that are not fixed and provide users with choices – e.g. University students who require guidance to pursue a particular major or to select courses in future semesters. The flexibility is realized with the use of parameters to the query that are determined by the users. This is implemented with the help of new SQL operators that is build on top of the SQL engine. The new SQL operators include (i) Extend operator that helps in creating extended attributes in the tuples of a relation, (ii) Recommend operator that performs the comparisons based on parameterized functions, (iii) Blend operator that combines recommendations to produce a unified relation.

The FlexRecs engine architecture includes the Workflow

manager to define the various workflows and end-users can invoke the defined ones. The Workflow parser is an important component that constructs an expression tree for the new operators defined for an input workflow. The Recommendation Plan Generator develops a recommendation execution plan for this expression tree. The sequence of SQL statements and function calls, the order of the execution of operators are specified in the plan. Finally, the Recommendation Generator executes the plan and returns the recommendations. It directs the SQL queries to the DB engine, combines any intermediate results and invokes the functions specified in the plan for comparing and blending tuples.

FlexRecs is thus a framework for declaratively defining recommendations integrating traditional relational and the new customized recommendation operators. FlexRecs enables designers to conveniently specify multiple recommendation paradigms and experiment with novel recommendation strategies and optimization techniques. Users can dynamically personalize recommendations tailored to their needs.

SnipSuggest: Context-aware autocompletion for SQL, N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu [10] describes a system that assists users in constructing context aware SQL queries on-the fly. As and when user types the query, user can select a particular clause and request the system to provide recommendations for specific clause. To generate its recommendations, SnipSuggest considers the query space as a Directed Acyclic Graph (DAG) and models each query as a set of features. Every possible set of features becomes a vertex in the DAG. The user's partially written query is transformed into a set of features and mapped onto a node in the DAG. A DAG represents the relationships between different clauses and recommends possible additions to various clauses in the current user's query. However, the system does not recommend complete queries and each query is treated independently of any previous one.

3 SQL QUERIE RECOMMENDATIONS

QueRIE is a system for Personalized Query Recommendations. It supports interactive and collaborative database exploration presenting users with customized and interesting query recommendations. Applications include, information mining over large-scale data warehouses, systems for ad-hoc analysis over big data, services for scientific-data exploration (Genome Browser, SkyServer) and so on.

3.1 The QueRIE Framework

The QueRIE framework is shown in Fig. 2. It is primarily a workflow in which the DBMS and the Recommendation Engine processes the active user's queries. The DBMS executes each query and returns the result set. The query is stored in the Query Log. The Recommendation Engine blends the current user's query with information gathered from the query logs of past users, and compiles a set of query recommendations for the user.

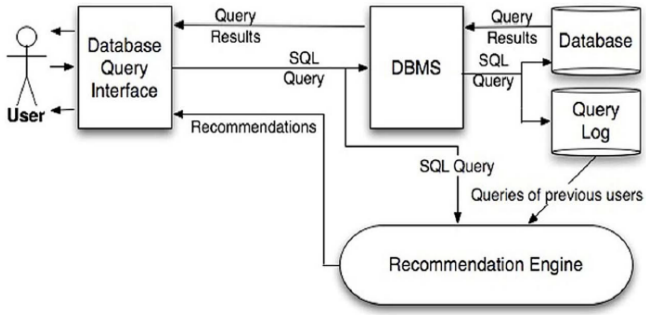

Fig. 2. QueRIE Architecture

Consider a scenario where users explore a relational database through a sequence of SQL queries. The objective of the exploration is to determine interesting information or confirm a particular hypothesis. The queries posted by a user during a single session or visit to the database are generally correlated.

This indicates that a user first inspects the result sets of the previous queries based on which the user formulates the next query. For example, a real user session, belonging to the SkyServer query logs, is shown in Fig. 3. This query pattern corresponds to an interactive exploration of the database: the user first understands the count of tuples satisfying a predicate, then proceeds to get the number of distinct objects corresponding to these tuples and finally extracts the objects that exist more than once in the database. This sequence of SQL statements also implies that the user is not familiar with the schema. The system could recommend the appropriate query (Query 4 in the Fig. 3) right after the first attempt (Query 1). This could be less time-consuming for the user. This is possible if a similar session already exists in the query logs and thus can be used to generate recommendations for the active user.

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*

Fig. 3. Sample SkyServer Query Log

3.2 Conceptual Framework

The notation S_i represents the session summary for user i. User i = 0 represents the current user whereas i = 1,...,n represents past users of the system. To generate recommendations for current user S_0, the framework first computes a "predicted" summary S^{pred}. The summary reflects the predicted degree of interest of S_0 considering different query characteristics, including those that already appear in user's queries, as well as new ones that have not yet been

used. The summary S^{pred} is then used as the "seed" for the generating recommendations.

The predicted summary is defined as: $S^{pred} = f(S_0, S_1,...,S_n)$. f is a function that combines information from both the active user's summary S_0 and the summaries $S_0, S_1,...,S_n$ of past users. The mixing factor α [0, 1] determines the importance of the S_0 (the current user's session) with respect to $S_0, S_1,...,S_n$ (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, α is a parameter of the system that can be configured depending on the type of the database and the user's querying behaviors.

The framework generates queries of highest importance based on the summary S^{pred}. These queries are then presented as recommendations to the active user. Overall, the framework as shown in Fig. 4 consists of the following components: (i) a model for session summaries, (ii) a method to compute the session summaries $S_0, S_1,...,S_n$, (iii) a method to compute S^{pred}, and (iv) a method to select queries based on S^{pred}.

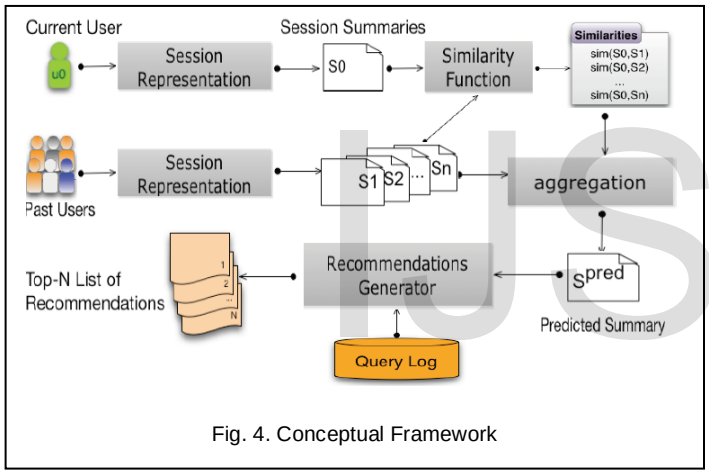

Fig. 4. Conceptual Framework

3.3 Tuple-Based Query Recommendation

In the tuple-based instantiation of the QueRIE framework, the session summary captures the tuples in the base tables that are touched by the queries in the session [1], [2]. Every coordinate of the weighted vector representing the session summary S_i corresponds to a distinct database tuple.

The length of the vector is assumed to be T that denotes the total number of tuples in the database. The weight $S_i[\tau]$ represents the importance of a given tuple $\tau \in T$ in session S_i. If the tuple appears in the result set for any one query issued in the session, then the weight will be non-zero. Two different weighting schemes are considered for setting $S_i[\tau]$, a binary scheme and a result-based scheme:

Binary Scheme:

$$S_Q[\tau] = \begin{cases} 1 & \text{if } \tau \text{ is a witness} \\ 0 & \text{if } \tau \text{ is not a witness} \end{cases} \quad (1)$$

Result-based Scheme:

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

Here ans(Q) is the result-set of Q. The importance of τ is diminished if Q returns many results, as this implies that query is not specific whereas a small cardinality indicates that the query is very restrictive, resulting in tuples that have high significance.

Session and Predicted Summary: Given vectors S_Q for each query Q by user i, the session summary S_i and Predicted Summary S^{pred} is defined as:

$$S_i = \sum_{Q \epsilon Q_i} S_Q \quad (3)$$

$$S^{pred} = \alpha . S_0 + (1 - \alpha) . \sum_{i=1,..,n} sim(S_i, S_0) . S_i \quad (4)$$

sim is a similarity function like cosine similarity that defines a metric between the two vectors. This approach is inspired by Web recommender systems and the recommendations are inclined towards users who exhibit similar behavior to the current user. The algorithm then recommends queries based on the computed S^{pred} that retrieve tuples of high predicted weights. For each candidate query Q, the similarity $sim(S_Q, S^{pred})$is computed. The candidate queries with the highest similarity are returned as recommendations to the user.

Analysis and discussion: The tuple-based approach is based on the individual witnesses to the user's queries and thus describes the user's querying behavior at a very fine level of detail. This increases complexity as, the session summaries grow linearly with the size of the database. 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. It is possible to implement this method more efficiently by using a Min-Hash probabilistic clustering technique that maps each session summary S_i to a "signature" $h(S_i)$ [2], [3]. The Jaccard similarity between vectors is reduced to the similarity of their signatures - $JaccardSim(S_i, S_0) = sim(h(S_i), h(S_0))$. Although this leads to an improvement in computational efficiency, there is loss of precision of the generated recommendations.

3.4 Fragment-Based Query Recommendation

The core idea of the fragment-based instantiation is to recommend queries whose syntactical features match the queries of the current user [3], [4]. It works in a similar manner to the tuple-based one. The two main differences are (i) session summary representation and (ii) similarity formulation. The coordinates of the session summaries correspond to fragments of queries rather than witnesses.

Fragments are identified as the following syntactical features of the queries in the session: relation and attribute references, join and selection predicates etc. The notion behind this approach is to recommend queries whose syntactical features are similar to the current user's queries. The objective is to identify fragments that appear together in several queries posed by different users, and use them in the recommendation process. QueRIE first calculates offline the pair-wise similari-

ties of all query fragments recorded in the query logs. Real-time predictions are then made based on these similarities taking into account the "rank" (or importance) of each fragment with respect to the current user session. 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.

Session and Predicted Summary: Session summary S_i is a vector whose cell S_i [Φ] contains a non-zero weight if the fragment Φ appears in at least one query of the session. For a given fragment Φ, a single query vector cell is defined as S_Q[Φ] as a binary variable that indicates if the fragment Φ exists in a query Q. Then S_i [Φ] represents the importance of Φ in session S_i. Conceptually, the length of the vector is equal to the number of possible fragments. Two different weighting schemes are considered, a binary scheme and a weighted scheme:

Binary scheme:

$$S_i = \bigvee_{Q \epsilon Q_i} S_Q \quad (5)$$

Weighted scheme:

$$S_i = \sum_{Q \epsilon Q_i} S_Q \quad (6)$$

Each coordinate S^{pred}[Φ] is computed as follows:

$$S^{pred}[\Phi] = \frac{\sum_{P \epsilon R} S_0[P] * sim(P,\Phi)}{\sum_{P \epsilon R} sim(P,\Phi)} \quad (7)$$

R is the set of top-k similar query fragments and k is a parameter of the framework. The fragment-similarity metric sim(P,Φ) evaluates the similarity of two fragments P and Φ in terms of their corresponding weights in the session summaries $S_1,...,S_n$. 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.

Analysis and Discussion: The fragment-based approach evidently captures information at a coarser level of detail. Consequently, there is a possibility to miss relevant and interesting correlations between users. However, the approach is advantageous as it can be implemented very efficiently. It allows for a scalable system as the space of fragments grows slowly. The summaries are very sparse as summaries are represented in terms of query fragments rather than tuples. This leads to faster similarity calculations. The fragment-to-fragment similarities can be computed offline and stored for very fast retrieval when recommendations need to be generated.

Query logs can contain several slightly dissimilar queries. Queries are relaxed in order to increase their cardinality. For instance, the WHERE clauses are relaxed by converting the numerical data and string literals to generic string representations. For example, strings are replaced by STR, hexadecimal numbers by HEXNUM and decimals by NUM. Subsequent to the query generalization, they are converted into fragments. For example, the fragments of Query 4 in Fig. 3 are: COUNT(*), REGION, REGION.TYPE PATMATCH, COUNT(*) COMPARE NUM. This relaxation in the query structure may result in missing out on interesting query recommendations

4 Conclusion

The QueRIE framework aims to generate useful SQL query recommendations to users of relational databases. An item-based approach is followed using query fragments to represent user sessions. Representing information at a coarser level of detail, results in some loss of accuracy in the predictions as compared to the tuple based approach. The fragment-based approach can be implemented efficiently as the space of fragments grows slowly and the summaries are very sparse. Recommendations can be generated faster as the fragment-to-fragment similarities can be computed offline and stored for quick retrieval with less computational overhead. This trade-off between computational efficiency and accuracy is worth pursuing, to be able to have a scalable implementation, running with real "big" data, with an acceptable loss in precision.

There are many interesting directions to explore in the future. The measure of impact, the query relaxation process has in the quality of recommendations can be analyzed. A sequence-based approach for recommendations can also be explored as further work. A more generic and scalable implementation of the QueRIE system can be explored. Another dimension would be to implement an access control mechanism in building the recommended queries.

References

[1] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis, "Collaborative filtering for interactive database exploration," in Proc. 21st Int.Conf. SSDBM, New Orleans, LA, USA, pp. 3–18, 2009.

[2] S. Mittal, J. S. V. Varman, G. Chatzopoulou, M. Eirinaki, and N. Polyzotis, "QueRIE: A recommender system supporting interactive database exploration," in Proc. IEEE ICDM, Sydney, NSW, Australia, 2010.

[3] J. Akbarnejad et al., "SQL QueRIE recommendations," PVLDB, vol. 3, no. 2, pp. 1597–1600, 2010.

[4] Magdalini Eirinaki, Suju Abraham, Neoklis Polyzotis, and Naushin Shaikh, "QueRIE: Collaborative Database Exploration", IEEE Transactions on Knowledge And Data Engineering, vol. 26, no. 7, pp.1778-1790, July 2014.

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

[6] "Personalized DBMS: An elephant in disguise or a chameleon?" IEEE Data Eng. Bull., vol. 34, no. 2, pp. 27–34, Jun.2011.

[7] J. Levandoski, M. Mokbel, and M. E. Khalefa, "FlexPref: A framework for extensible preference evaluation in database systems," in Proc. IEEE ICDE, Long Beach, CA, USA, 2010.

[8] N. Khoussainova, M. Balazinska, W. Gatterbauer, Y. Kwon, and D. Suciu, "A case for a collaborative query management system," in Proc. 4th Biennal CIDR, Asilomar, CA, USA, 2009.

[9] G. Koutrika, B. Bercovitz, and H. Garcia-Molina, "FlexRecs:Expressing and combining flexible recommendations," in Proc.SIGMOD Conf., New York,

NY, USA, pp. 745–757, Jun. 2009.

[10] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu, "SnipSuggest: Context-aware autocompletion for SQL," PVLDB,vol. 4, no. 1, pp. 22–33, 2011.