IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

Recommendation System for Answering Missing Tuples

Kanchan Pekhale[1], Dr. K. V. Metre[2]

PG Student, Computer Engineering Department, MET BKC Adgaon, Nashik, Maharashtra, India[1]

Professor, Department of Computer Engineering, MET BKC Adgaon, Nashik, India [2]

Abstract: In the recent years, the quality and the usability of database systems have received more attention. The performance of database systems has gained more improvement in the past decades so, they are more and more difficult to use. The why-not questions are needs of user to know why her expected tuples are not shown up in the query results i.e. the features of explaining missing tuples in database queries. Database system is having the capability that enables users to seek clarifications on expected query results as well as the absence of expected tuples (i.e. missing tuples). It would be very helpful to users if they referred why-not questions to seek clarifications on expected tuples in query results. There are two algorithms to answer why-not questions efficiently. These algorithms are able to return high quality explanations efficiently. Many users love to pose those kinds of queries when they are making multi-criteria decisions and user need approximate information from the huge Database.

Keywords: Top-k Question, Dominating Question, Skyline Refined Queries, ConQueR Method.

I. INTRODUCTION

In recent years, there is a growing effort to improve the usability of database systems. A search engine can help a user to answer a question by locating information sources based on keywords. The feature of explaining missing tuples in database queries is called why-not questions. It has received growing attentions in recent years. A why-not question is being posed when a user wants to know why her expected tuples are not shown up in the query result. Why-not questions are helpful to users to seek clarification on missing tuples from the result. Recently, a certain work is done on answering why- not questions on traditional relational/SQL queries. However, none of those can answer why-not questions on preference queries like top-k queries yet. Answering why-not questions on top-k queries is useful because users love to pose top-k queries when making multi-criteria decisions. However, they may feel lost when their expected answers are missing in the query result and they may want to know why: Is it because I have set k too small?, Or I have set my weightings badly?, Or because of both? For Answering the why-not questions on top-k queries there are two algorithms. Namely, a why-not top-k question and a why-not dominating question. Top-k dominating queries or simply dominating queries is a variant of top-k query that users may pose why-not questions on. The top-k dominating query returns k data objects which dominate the highest number of objects in a dataset. While a top-k dominating query frees users from specifying the set of weightings by ranking the objects based on the number of objects that they could dominate (e.g., if object x dominates nine objects while object y dominates four objects, then x ranks higher than y). For example, the agent of Jeremy Lin, a hot NBA player this year, may pose a top-100 dominating query about the best guards in NBA history. When Lin is not in the result, his agent may want to know the reason: Is that I have set my k too small? Finding the best explanations is actually computationally expensive for both the algorithms. In the why-not top-k question users can provide W as input, which slightly limits its practicability and in why-not top-k dominating question users need not to provide W as input. In the why-not paradigm, users are quite clear with which are the missing objects and explained why those objects are missing.

II. LITERATURE SURVEY

SQL Query Recommendation system aims at assisting non-expert users of scientific databases by tracking their querying behavior and generating personalized query recommendations. The system is supported by two recommendation engines and the underlying recommendation algorithms. First approach identifies potentially interesting parts of the database related to the corresponding data analysis task by locating those database parts that were accessed by similar users in the past. The second algorithm identifies structurally similar queries posted by the current user. Both approaches result in a recommendation set of SQL queries that is provided to the user to modify or directly post to the database. The drawback of this is, it mainly focus on the improving the Usability and the Quality of the database systems [2].

SnipSuggest aims to help the increasing population of non-expert database users, who need to perform complex analysis on their large-scale datasets, but it is difficult to writing SQL queries. As a user types a query, SnipSuggest recommends possible additions to various clauses in the query using relevant snippets collected from a log of past queries. As a user types a query, she can ask SnipSuggest for recommendations of what to add to a specific clause of her query. In response, SnipSuggest recommends small SQL snippets [3].

Explaining missing answers of queries is useful in various scenarios, including query understanding and debugging.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

When queries are used to define multiple views, one may ask why, employee information is missing from both the employee register and the payroll views. Artemis algorithms are able to generate explanations for a set of missing tuples over a set of queries that include selection, projection, join, union, and aggregation and grouping (SPJUA). For explanations, it encodes the problem into a set of constraints and it combines existing data with new data [4].

Database systems are having the capability that enables users to seek clarifications on unexpected query results. There are two existing models to explain why-not questions on query results. The first approach modifying some tuples in the database so, the result of the query on the modified database will include both the original result and the specified missing tuples. The second approach, explains missing tuples by identifying the manipulation operations in the query plan that are responsible for excluding the missing tuples [5].

The top-k dominating query returns k data objects which dominate the highest number of objects in a dataset. Identify the importance and practicability of the query and define some of its potential extensions. A simple evaluation method for top-k dominating queries is based on skyline computation. Advantage of the top-k query is that the user is able to control the number of results. It might not always be easy for the user to specify an appropriate ranking function [6].

For avoiding the above drawbacks answering the why-not questions on top-k queries are used, there are two algorithms to answer such why-not questions efficiently. First is a why-not top-k question and second is why-not dominating questions. The why-not questions are the users need to know why her expected tuples do not shown up in the query results i.e. the features of explaining missing tuples in database queries [1].

III. SYSTEM ARCHITECTURE

As shown in Fig. 1, the user can give an input query to the algorithms. If user can specify the weighting value, the input query is given to why-not top-k question otherwise the query is given to why-not top-k dominating question.

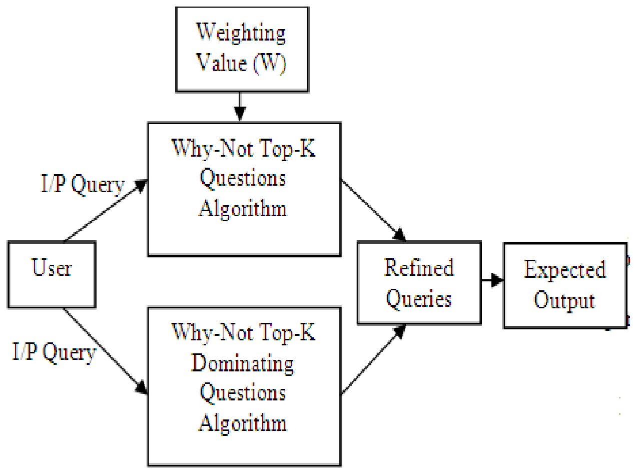
Weighting Value (W)
Why-Not Top-K Questions Algorithm
I/P Query
User
I/P Query
Why-Not Top-K Dominating Questions Algorithm
Refined Queries
Expected Output

Fig1. System Architecture

In the why-not top-k question users can provide W as input, which slightly limits its practicability and in why-not top-k dominating question users need not to provide W as input. For generation of result a query-refinement approach is used. That tells the user how to revise her original queries so that the missing answers can be returned to the result. This approach defines that a good refined query should be (a) similar - have few edits comparing with the original query and (b) precise - have few extra tuples in the result, except the original result plus the missing tuples. This helps users to quantify their preferences as a set of weightings.

A. WHY-NOT TOP-K QUESTION

Suppose there is only one missing object $\vec{m}$. First, execute a progressive top-k query q'_0 based on the weighting vector $\vec{w}_0$ in the users original query q_0, using any progressive top-k query evaluation algorithm and stop when $\vec{m}$ comes forth to the result set with a ranking r_0. If $\vec{m}$ does not appear in the query result, report to the user that $\vec{m}$ does not exist in the database and the process terminates.

If $\vec{m}$ exists in the database, then randomly sample a list of weighting vectors $S = [\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_s]$ from the weighting space. For each weighting vector $\vec{w}_i \in S$, then formulate a progressive top-k query q'_i using $\vec{w}_i$ as the weighting. Each query q'_i is executed by a progressive top-k algorithm, which progressively reports each top ranking object one-by-one, until the missing object $\vec{m}$ comes forth to the result set with a ranking r_i. So, after s+1 progressive top-k executions, we have s+1 refined queries $q'_i(r_i, \vec{w}'_i)$ where i = 0, 1, 2,...,s, with missing object $\vec{m}$ known to be rank r_i^{th} exactly. Finally, the refined query $q'_i(r_i, \vec{w}'_i)$ with the least penalty is returned to the user as the answer. The least Penalty can be calculated by using following formula:

$$\text{Penalty}(k', \vec{w}') = \lambda_k \frac{\Delta k}{(r_0 - k_0)} + \frac{\Delta w}{\sqrt{1+\sum w_0[i]^2}} \qquad (1)$$

In the above equation Δk and Δw are used to measure the quality of the refined query. Where $\Delta k = \max(0, k' - k0$and $\Delta w = w' - w0$. The k′ is a refined query value is smaller than the original k_0 value. r_0 is the rank of the missing object $\vec{m}$ under the original weighting vector $\vec{w}_0$. To capture a users tolerance to the changes of k and $\vec{w}$ on her original query q_0 define a basic penalty model that sets the penalties λ_k and λ_w to Δk and Δw, respectively, where $\lambda_k + \lambda_w = 1$.

B. WHY-NOT TOP-K DOMINATING QUESTION

Answering why-not top-k dominating questions is similar to the answering top-k why-not Questions. Assume with the case where there is only one missing object $\vec{m}$. First, execute a top-k dominating query q'_0 using a progressive top-k dominating query evaluation algorithm and stop when $\vec{m}$ comes forth to the result set with a ranking r_0. If $\vec{m}$ does not appear in the query result, we report to the user that $\vec{m}$ does not exist in the database and the process terminates. If $\vec{m}$ exists in the database, then draw a list of data value samples $S = [\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_s]$. For each data value sample $\vec{x}_i \in S$, we modify $\vec{m}$ values to be $\vec{x}_i$ and then execute a progressive top-k dominating query until $\vec{m}$ comes forth to the result set with a ranking r_i. So, after s+1 progressive top-k dominating executions, we have s+1 “refined queries and modified values" pairs: $\langle q'_0(r_0), \vec{m} = \vec{m}\rangle, \langle q'_1(r_1), \vec{m} = \vec{x}_1\rangle, \ldots, \langle q'_s(r_s), \vec{m} = \vec{x}_s\rangle$. Finally, the pair with the least penalty is returned to the user as the answer. A top-k dominating query is composed of a result set size k and a special score function, which scores an object $\vec{p}$ by the number of points that it can dominate. The query result is then the top-k objects with the highest scores.

C. SKYLINE REFINED QUERIES

Thus, a refined query is considered to be good if its dissimilarity and imprecision metrics are low [5]. Among all the possible refined queries for a why-not question, interested in the set of skyline refined queries defined as follows: Given two different refined queries Q1 and Q2, say that Q1 dominates Q2 if (1) both the metrics of Q1 are at least as low as those of Q2 and (2) for at least one of the metrics, Q1’s value is strictly lower than that of Q2’s. Defined a refined query Q′ to be a skyline refined query (or skyline query) if Q′ is not dominated by any other refined query. Thus, goal is to compute skyline refined queries to explain the question.

D. ConQueR METHOD

ConQueR, is Constraint-based Query Refinement, to explain why-not questions by automatically generating one or more refined queries. ConQueR is designed to be a similarity-driven approach where it tries to generate refined queries with low dissimilarity values before considering more precise refined queries that have higher dissimilarity values. Assume, Given a why-not question (S,C) for a query Q on database D, ConQueR will first consider refined queries Q′ that have the same query schema as Q. That is, ConQueR tries to derive Q′ by simply modifying selection predicate(s) in Q to explain the why-not tuples while minimizing the imprecision metric. If such refined queries exist, ConQueR will only generate skyline refined queries that all share the same query schema as Q. However, if no such refined query exists, ConQueR then looks for refined queries that have a slightly different query schema, and so on.

Thus, ConQueR effectively iterates over a sequence of query schemas QS_1,…, QS_k to search for refined queries: QS_1 is the query schema of the input query Q, and schema QS_{i+1} is considered only if there are no refined queries with schema QS_1,…,QS_i. The sequence of query schemas considered is of increasing dissimilarity metric values, and if QS_k is the first query schema in the sequence to contain refined queries, ConQueR will generate all skyline refined queries with schema QS_k as possible explanations to the why-not question.

IV. CONCLUSION

User does not understand why her expected answers are missing in the query result. For this, answering why-not questions on two types of top-k queries, the basic top-k query where users need to specify the set of weightings, and the top-k dominating query where users do not need to specify the set of weightings because the ranking function ranks an object higher, if it can dominate more objects. A refined query with approximately minimal changes to the k value and their weightings is returned to the user. It can be used for multi-criteria decision as well as for approximate information from database. The work will be performed on numeric data as well as non-numeric data. Both the algorithms will able to return high quality explanations efficiently.

REFERENCES

[1] Zhian He, Eric Lo, “Answering Why-Not Questions on Top-K Queries”, IEEE, vol.26, no.6, June 2014.

[2] J. Akbarnejad et al., “SQL query recommendations”, PVLDB, vol. 3, no. 2, pp. 15971600, 2010.

[3] M. Balazinska, N. Khoussainova, Y. C. Kwon, and D. Suciu, “Snip Suggest: Context-aware auto completion for SQL”, PVLDB, vol. 4, no. 1, pp. 2233, 2010.

[4] M. Herschel and M. A. Hernndez, “Explaining missing answers to SPJUA queries", PVLDB, vol. 3, no. 12, pp. 185196, 2010.

[5] Q. T. Tran and C.-Y. Chan, “How to ConQueR why-not questions”, in Proc. ACM SIG-MOD, New York, NY, USA, 2010, pp. 1526.

[6] M. L. Yiu and N. Mamoulis, “Efficent processing of top-k dominating queries on multi-dimensional data”, in Proc. VLDB, Vienna, Austria, 2007, pp. 541552.