International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE)
ISSN: 0976-1353 Volume 11 Issue 1 –NOVEMBER 2014.

Dynamic Query Forms for Database Queries

Rajulapati Siva Sankar[*1] and J. Raja Rajeswari[#2]

Student, Dept of Computer Science, Jogaiah Institute of Technology and Sciences College of Engineering, A.P, India

Asst Professor, Dept of Computer Science, Jogaiah Institute of Technology and Sciences College of Engineering, A.P, India

sivasankar.rajulapati@gmail.com

rajeswari506@gmail.com

Abstract— Modern scientific databases and web databases maintain large and heterogeneous data. These real-world databases contain over hundreds or even thousands of relations and attributes. Traditional predefined query forms are not able to satisfy various ad-hoc queries from users on those databases. This paper proposes DQF, a novel database query form interface, which is able to dynamically generate query forms. The essence of DQF is to capture a user's preference and rank query form components, assisting him/her to make decisions. The generation of a query form is an iterative process and is guided by the user. At each iteration, the system automatically generates ranking lists of form components and the user then adds the desired form components into the query form. The ranking of form components is based on the captured user preference. A user can also fill the query form and submit queries to view the query result at each iteration. In this way, a query form could be dynamically refined till the user satisfies with the query results. We utilize the expected F-measure for measuring the goodness of a query form. A probabilistic model is developed for estimating the goodness of a query form in DQF. Our experimental evaluation and user study demonstrate the effectiveness and efficiency of the system.

Index Terms—Query Form, User Interaction, Query Form Generation.

I. INTRODUCTION

Database systems support a simple Boolean query retrieval model, where a selection query on a SQL database returns all tuples that satisfy the conditions in the query. This often leads to the Many-Answers Problem: when the query is not very selective, too many tuples maybe in the answer. In this section, we formally define the Many-Answers Problem in ranking database query results, and also outline a general architecture of our solution. The Many-Answers Problem has been investigated outside the database area, especially in Information Retrieval (IR), where many documents often satisfy a given keyword-based query. Approaches to overcome this problem range from query reformulation techniques (e.g.,the user is prompted to refine the query to make it more selective), to automatic ranking of the query results by their degree of "relevance" to the query (though the user may not have explicitly specified how) and returning only the top-K subset.

In this paper we propose an automated ranking approach for the Many-Answers Problem for database queries. Our solution is principled, comprehensive, and efficient. We summarize our contributions below.

Any ranking function for the Many-Answers Problem has to look beyond the attributes specified in the query, because all answer tuples satisfy the specified conditions. However, investigating unspecified attributes is particularly tricky since we need to determine what the user's preferences for these unspecified attributes are. In this paper we propose that the ranking function of a tuple depends on two factors:

(a) a global score which captures the global importance of unspecified attribute values, and(b)

a conditional score which captures the strengths of dependencies (or correlations) between specified and unspecified attribute values. For example, for the query "City = Seattle and View =Waterfront", a home that is also located in a "School District = Excellent" gets high rank because good school districts are globally desirable. A home with also "BoatDock = Yes" gets high rank because people desiring a waterfront are likely to want a boat dock. While these scores may be estimated by the help of domain expertise or through user feedback, we propose an automatic estimation of these scores via workload as well as data analysis. For example, past workload may reveal that a large fraction of users seeking homes with a waterfront view have also requested for boat docks.

The next challenge is how do we translate these basic intuitions into principled and quantitatively Describable ranking functions? To achieve this, we develop ranking functions that are based on Probabilistic Information Retrieval (PIR) ranking models. We chose PIR models because we could extend them to model data dependencies and correlations (the critical ingredients of our approach) in a more principled manner than if we had worked with alternate

IR ranking models such as the Vector-Space model. We note that correlations are often ignored in IR because they are very difficult to capture in the very high dimensional and sparsely populated feature spaces of text data, whereas there are often strong correlations between attribute values in relational data(with functional dependencies being extreme cases),which is a much lower dimensional, more explicitly structured and densely populated space that our ranking functions can effectively work on.

II. Problem Definition

In this section, we formally define the Many-Answers Problem in ranking database query results, and also outline a general architecture of our solution.

2.1 Problem Definition

In this paper, we propose a Dynamic Query Form system: DQF, a query interface which is capable of dynamically generating query forms for users. Different from traditional document retrieval, users in database retrieval are often willing to perform many rounds of actions (i.e., refining query conditions) before identifying the final candidates [7]. The essence of DQF is to capture user interests during user interactions and to adapt the query form iteratively. Each iteration consists of two types of user interactions: Query Form Enrichment and Query Execution (see Table 1). Figure 1 shows the work-flow of DQF. It starts with a basic query form which contains very few primary attributes of the database. The basic query form is then enriched iteratively via the interactions between the user and our system until the user is satisfied with the query results. In this paper, we mainly study the ranking of query form components and the dynamic generation of query forms.

2.2 Contributions:

Our contributions can be summarized as follows:

• We propose a dynamic query form system which generates the query forms according to the user's desire at run time. The system provides a solution for the query interface in large and complex databases.

• We apply F-measure to estimate the goodness of a query form [30]. F-measure is a typical metric to evaluate query results [33]. This metric is also appropriate for query forms because query forms are designed to help users query the database. The goodness of a query form is determined by the query results generated from the query form. Based on this, we rank and recommend the potential query form components so that users can refine the query form easily.

• Based on the proposed metric, we develop efficient algorithms to estimate the goodness of the projection and selection form components. Here efficiency is important because DQF is an online system where users often expect quick response. The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 defines the query form and introduces how users interact with our dynamic query form. Section 4 defines a probabilistic model to rank query form components. Section 5 describes how to estimate the ranking score. Section 6 reports experimental results, and finally Section 7 concludes the paper

.

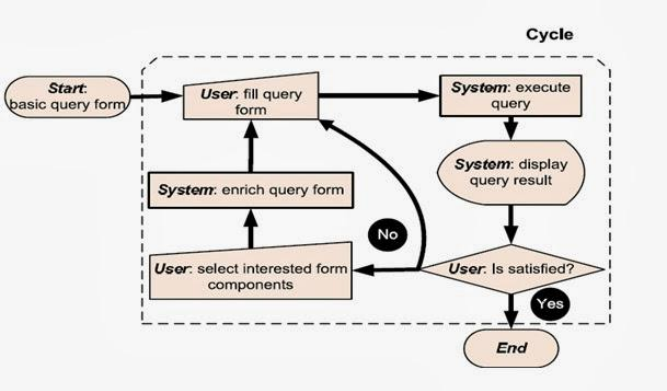

Fig. 1. Flowchart of Dynamic Query Form

III. RELATED WORK

How to let non-expert users make use of the relational database is a challenging topic. A lot of research works focus on database interfaces which assist users to query the relational database without SQL. QBE (Query-By-Example) [36] and Query Form are two most widely used database querying interfaces. At present, query forms have been utilized in most real-world business or scientific information systems. Current studies and works mainly focus on how to generate the query forms.

Customized Query Form: Existing database clients and tools make great efforts to help developers design and generate the query forms, such as EasyQuery [3], Cold Fusion [1], SAP, Microsoft Access and so on. They provide visual interfaces for developers to create or customize query forms. The problem of those tools is that, they are provided for the professional developers who are familiar with their databases, not for end-users [16]. [17] proposed a system which allows end-users to customize the existing query form at run time. However, an end-user may not be familiar with the database. If the database schema is very large, it is difficult for them to find appropriate database entities and attributes and to create desired query forms.

Automatic Static Query Form: Recently, [16] [18] proposed automatic approaches to generate the database query forms without user participation. [16] Presented a data-driven method. It first finds a set of data attributes, which are most likely queried based on the database schema and data instances. Then, the query forms are generated based on the selected attributes. [18] is a workload-driven method. It applies clustering algorithm on historical queries to find the representative queries. The query forms are then generated based on those representative queries. One problem of the aforementioned approaches is that, if the database schema is large and complex, user queries could be quite diverse. In that case, even if we generate lots of query forms in advance, there are still user queries that cannot be satisfied by any one of query forms. Another problem is that, when we generate a large number of query forms, how to let users find an appropriate and desired query form would be challenging. A solution that combines keyword search with query form generation is proposed in [12]. It automatically generates a lot of query forms in advance. The user inputs several keywords to find relevant query forms from a large number of pre generated query forms. It works well in the databases which have rich textual information in data tuples and schemas. However, it is not appropriate when the user does not have concrete keywords to describe the queries at the beginning, especially for the numeric attributes.

Autocompletion for Database Queries: In [26], [21], novel user interfaces have been developed to assist the user to type the database queries based on the query workload, the data distribution and the database schema. Different from our work which focuses on query forms, the queries in their work are in the forms of SQL and keywords.

Query Refinement: Query refinement is a common practical technique used by most information retrieval systems [15]. It recommends new terms related to the query or modifies the terms according to the navigation path of the user in the search engine. But for the database query form, a database query is a structured relational query, not just a set of terms.

Dynamic Faceted Search: Dynamic faceted search is a type of search engines where relevant facts are presented for the users according to their navigation paths [29] [23]. Dynamic faceted search engines are similar to our dynamic query forms if we only consider Selection components in a query. However, besides Selections, a database query form has other important components, such as Projection components. Projection components control the output of the query form and cannot be ignored. Moreover, designs of Selection and Projection have inherent influences to each other.

Database Query Recommendation: Recent studies introduce collaborative approaches to recommend database query components for database exploration [20] [9]. They treat SQL queries as items in the collaborative filtering approach, and recommend similar queries to related users. However, they do not consider the goodness of the query results. [32] Proposes a method to recommend an alternative database query based on results of a query. The difference from our work is that, their recommendation is a complete query and our recommendation is a query component for each iteration.

Dynamic Data Entry Form: [11] develops an adaptive forms system for data entry, which can be dynamically changed according to the previous data input by the user. Our work is different as we are dealing with database query forms instead of data-entry forms.

Active Feature Probing: Zhu et al. [35] develop the active featuring probing technique for automatically generating clarification questions to provide appropriate recommendations to users in database search. Different from their work which focuses on finding the appropriate questions to ask the user, DQF aims to select appropriate query components.

IV. RANKING METRIC

Query forms are designed to return the user's desired result. There are two traditional measures to evaluate the quality of the query results: precision and recall [30]. Query forms are able to produce different queries by different inputs, and different queries can output different query results and achieve different precisions and recalls, so we use expected precision and expected recall to evaluate the expected performance of the query form. Intuitively, expected precision is the expected proportion of the query results which are interested by the current user. Expected recall is the expected proportion of user interested data instances which are returned by the current query form. The user interest is estimated based on the user's clickthrough on query results displayed by the query form. For example, if some data instances are clicked by the user, these data instances must have high user interests. Then, the query form components which can capture these data instances should be ranked higher than other components. Next we introduce some notations and then define expected precision and recall.

Notations: Table 2 lists the symbols used in this paper. Let F be a query form with selection condition σF and projection attribute set AF . Let D be the collection of instances in ◃▹ (RF). N is the number of data instances in D. Let d be an instance in D with a set of attributes A = fA1,A2, ...,Ang, where n = jAj. We use dAF to denote the projection of instance d on attribute set AF and we call it a projected instance. P(d) is the occurrence probability of d in D. P(σF jd) is the probability of d satisfies σF . P(σF jd) 2 f0, 1g. P(σF jd) = 1 if d is returned by F and P(σF jd) = 0 otherwise. Since query form F projects instances to attribute set AF , we have DAF as a projected database and P(dAF) as the probability of

projected instance Daf in the projected database. Since there are often duplicated projected instances, P(dAF) may be greater than 1/N. Let Pu(d) be the probability of d being desired by the user and Pu(dAF) be the probability of the user being interested in a projected instance. We give an example below to illustrate those notations.

V. RANKING ALGORITHM

This algorithm is used for getting data from unstructured text documents. The main purpose of the algorithm is to get the relevant information for a query from unstructured text documents. The algorithm involves the following steps. Input: List of documents D[] = d1,d2,…….,dn Output: Query result documents q[] ={di,…..}

1. For i=1 to n do
2. From the given list of documents, a document is selected, say di.
3. The selected document contents are splitted into tokens(word).
4. The stop words are removed from these tokens and token list is generated as shown below

Ti[] ={ti,,t2,……}

5. The search keyword if compared with the list of tokens.
6. The matching count namely match count is calculated. Match count is the no of matching tokens with the keyword.
7. The match count for each document is stored in a separate array say match[]. A document match count is zero when no single token in the document matches with the keyword.
8. End for
9. From the match[] array remove the documents corresponding to a match count of zero.
10. Based a threshold, the documents are taken from the match count and placed in the Query result array q[]. The threshold can be any integer other than zero. For example if the threshold is 2, all documents in which two tokens are matching will be placed in the resultant query.
11. Finally the document list in the query list is displayed.

VI. CONCLUSION AND FUTURE WORK

In this paper we propose a dynamic query form generation approach which helps users dynamically generate query forms. The key idea is to use a probabilistic model to rank form components based on user preferences. We capture user preference using both historical queries and run-time feedback such as click through. Experimental results show that the dynamic approach often leads to higher success rate and simpler query forms compared with a static approach. The ranking of form components also makes it easier for users to customize query forms. As future work, we will study how our approach can be extended to non relational data.

As for the future work, we plan to develop multiple methods to capture the user's interest for the queries besides the click feedback. For instance, we can add a text-box for users to input some keywords queries. The relevance score between the keywords and the query form [12] can be incorporated into the ranking of form components at each step.

VII. REFERENCES

[1] Cold Fusion. http://www.adobe.com/products/coldfusion/.
[2] DBPedia. http://DBPedia.org.
[3] EasyQuery. http://devtools.korzh.com/eq/dotnet/.
[4] Freebase. http://www.freebase.com.
[5] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In Proceedings of VLDB, pages 81–92, Berlin, Germany, September 2003.
[6] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In Proceedings of WSDM, pages 5–14, Barcelona, Spain, February 2009.
[7] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In CIDR, 2003.
[8] S. Boriah, V. Chandola, and V. Kumar. Similarity measures for categorical data: A comparative evaluation. In Proceedings of SIAM International Conference on Data Mining (SDM 2008), pages 243–254, Atlanta, Georgia, USA, April 2008.
[9] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In Proceedings of SSDBM, pages 3–18, New Orleans, LA, USA, June 2009.
[10] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. ACM Trans. Database Syst. (TODS), 31(3):1134–1168, 2006.
[11] K. Chen, H. Chen, N. Conway, J. M. Hellerstein, and T. S. Parikh. Usher: Improving data quality with dynamic forms. In Proceedings of ICDE conference, pages 321–332, Long Beach, California, USA, March 2010.
[12] E. Chu, A. Baid, X. Chai, A. Doan, and J. F. Naughton.Combining keyword search and forms for ad hoc querying of databases. In Proceedings of ACM SIGMOD Conference, pages 349–360, Providence, Rhode Island, USA, June 2009.
[13] S. Cohen-Boulakia, O. Biton, S. Davidson, and C. Froidevaux.Bioguidesrs: querying multiple sources with a user-centric perspective. Bioinformatics, 23(10):1301–1303, 2007.
[14] G. Das and H. Mannila. Context-based similarity measures for categorical databases. In Proceedings of PKDD 2000, pages 201–210, Lyon, France, September 2000.
[15] W. B. Frakes and R. A. Baeza-Yates. Information Retrieval: Data Structures and Algorithms. Prentice-Hall, 1992.
[16] M. Jayapandian and H. V. Jagadish. Automated creation of a forms-based database query interface. In Proceedings of the VLDB Endowment, pages 695–709, August 2008.
[17] M. Jayapandian and H. V. Jagadish. Expressive query specification through form customization. In Proceedings of International Conference on Extending Database Technology (EDBT), pages 416–427, Nantes, France, March 2008.

RAJULAPATI SIVA SANKAR received her B.Tech Degree in IT from Jogaiah Institute of Technology And Sciences Kalagampudi West Godavari (Dt), in 2012. Presently, he is pursuing the M.Tech degree in CSE from JOGAIAH INSTITUTE OF TECHNOLOGY AND SCIENCES

COLLEGE OF ENGINEERING Kalagampudi. West Godavari (Dt).

J.RAJA RAJESWARI received the M.Tech degree from Jogaiah Institute of Technology And Sciences Kalagampudi West Godavari (Dt), in 2013.. Currently She is working as ASSISTANT Professor in JITS Engineering College, Kalagampudi. She has three years of experience in teaching.