International Journal of Engineering Research and General Science Volume 3, Issue 3, May-June, 2015
ISSN 2091-2730

AN INTERACTIVE QUERY FORM INTERFACE FOR DATABASE EXPLORATON

Mr. Avinash Sanganna[1], Dr. Mohammed Abdul Waheed[2]

[1]PG Scholar, CSE Department, Visvesvaraya Technological University,
Postgraduate Centre and Regional Office Kalaburagi 585105, Karnataka, India
avinash.jk@gmail.com

[2]Associate Professor, CSE Department, Visvesvaraya Technological University,
Postgraduate Centre and Regional Office Kalaburagi 585105, Karnataka, India
dr.mawaheed@gmail.com

ABSTRACT- Modern scientific databases and web databases maintain large and heterogeneous data. The static query forms are not able to satisfy various ad-hoc queries on those types of databases. Through customize forms user can modify but he must be familiar with the database schema. Hence it proposes an interactive query form which is able to generate query forms at runtime. The generation of a query form is an iterative process until the user is satisfied. At each iteration, the system generates clusters to represent results and the user can choose the cluster then the attributes will be choose by the system and it will calculate F-measure of those attributes and update the query form by adding those components. It utilizes the expected F-Measure for measuring the goodness of a query form.

Keywords -- Data Clustering, F-Measure, Query Form, Query Form Generation, Query Form Enrichment, User Interaction.

1. INTRODUCTION

A database is only as useful as its query interface allows it to be. If a user is unable to convey to the database what he or she wants from it, even the richest data store provides little or no value. Writing well-structured queries, in languages such as SQL and XQuery, can be challenging due to a number of reasons, including the user's lack of familiarity with the query language and the user's ignorance of the underlying schema. A form is a simple and intuitive query interface frequently used to provide easy database access. It requires no knowledge, on the part of the user, of how the data is organized in storage and no expertise in query languages. For these reasons, forms are a popular choice for most of today's databases. Creating a forms-based interface for an existing database requires careful analysis of its data content and user requirements. Many existing database management and development tools, such as Easy Query [2], Cold Fusion [1], SAP and Microsoft Access, provide several mechanisms to let users create customized queries on databases. However, the creation of customized queries totally depends on users' manual editing [3]. If a user is not familiar with the database schema in advance, those hundreds or thousands of data attributes would confuse him/her.

1.1 Motivation

The effectiveness of a manually designed forms-based interface largely depends on the developer's understanding and estimation of its user's needs. This is evident from observable differences between two or more interfaces designed to serve the same purpose but by different UI designers.For example, consider the task of buying a used car. There are several database-backed websites that help users buy used vehicles and several of them provide forms based interfaces to help a user find exactly the type of car he or she is looking for. Specifically, the set of queries that they allow users to ask about the desired car are not the same. This can make some more desirable for a specific information need even if the data is the same in all of them. We analyzed the interfaces provided by five such

websites: Car.com, Cars.com, AutoTrader.com, Cars Direct and eBay Motors. While all of these websites serve the same purpose (helping a user find and buy a used car) and have the same underlying data (used car listings) with more or less the same set of attributes for each listing, the ways in which their query forms are structured and presented to users are quite different. Our goal in this paper is to generation of interactive forms-based interface while keeping the interface simple.

2. RELATED WORK

A lot of research works focus on database interfaces which assist users to query the relational database without SQL. QBE (Query-By-Example) [6] and Query Form are two most widely used database querying interfaces. Current studies and works mainly focus on how to generate the query forms.

Modified Query Form: The tools provided by the database clients make great efforts to help developers generate the query forms, such as Easy Query [2], Cold Fusion [1] 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 [3].H.V. Jagadish proposed a system which allows end-users to customize the existing query form at run time [7]. If the database schema is very large, it is difficult for end user to find appropriate database entities and attributes.

Automated creation of forms: M. Jayapandian presented a data-driven method [3]. 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.

Automating the design and construction of query forms: H.V. Jagadish presented a workload-driven method [8].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 [3],[8] is that, 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 query form would be challenging.

Combining keyword search and forms: A solution for aforementioned approaches [3], [8] is proposed in [9].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 but it is not appropriate when the user does not have concrete keywords to describe the queries.

3. METHODOLOGY

3.1 Architectural Overview

For a declarative query, to design a form, we must first analyze it and identify its constraints and the required results. Then we use information gathered from this analysis, as well as from the schema of the database, to create the necessary set of form-elements. Finally, we arrange these elements in groups, label them suitably, and lay them out in a meaningful way on the form. Thus our challenge is to design a good set of forms without having an actual query log at hand.

In most cases the schema complexity is simply due to the richness of the data. This complexity is reflected in the queries to the database, many with more than one entity of interest. In this paper, we propose an Interactive Query Form [IQF] system, is 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 [4].

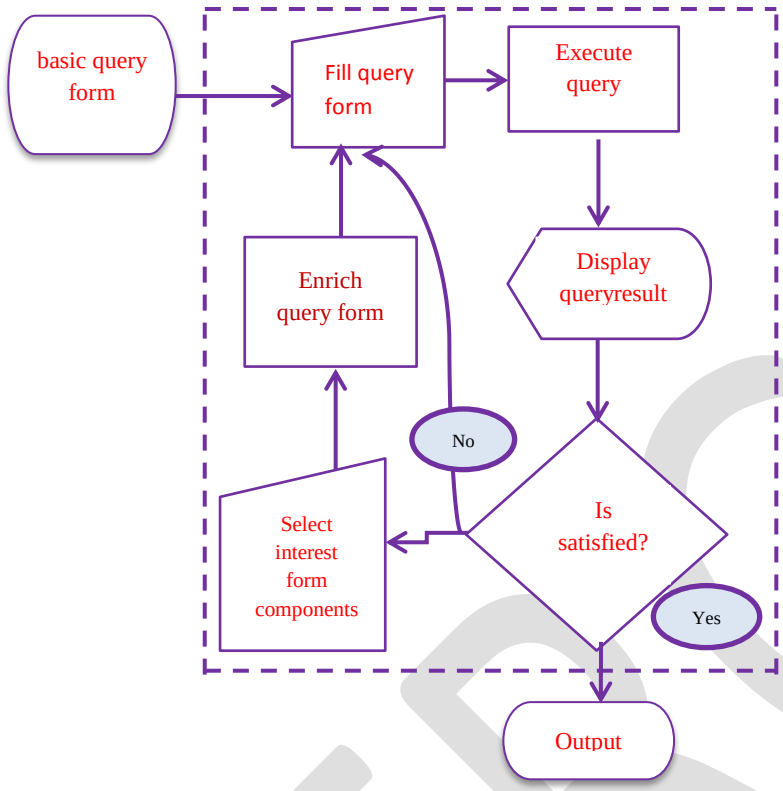

Figure1. Flowchart of interactive query form.

Fig. 1 shows the work-flow of IQF. 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. The general data clustering and F-Measure plays a vital role in this paper.

We can break the forms interface design problem down into two challenges discussed below.

- The first challenge to address is determining the schema fragment(s) most likely to be of interest to a querying user. Schemas can be extremely complex in real-world databases, but actual queries issued to a database typically focus on a small subset of its schema.
- The second challenge in automated form design is to partition the filtered collection of schema elements into groups such that the entities, attributes and relationships present in a single group can meaningfully interrelate on a form to express user queries.

The iteration consists of two types of user interactions: Query Form Enrichment and Query Execution (see TABLE 1).

Table 1: Interactions between user and interactive query form.

Query Form Enrichment	1) IQF recommends a ranked list of query form components to the user. 2) The user selects the desired form components into the current query form.
Query Execution	1) The user fills out the current query form and submit a query. 2) IQF executes the query and shows the results. 3) The user provides the feedback about the query results.

3.2 Contribution

Our contributions can be summarized as follows:

- We propose an interactive query form system which generates the query forms according to the user's desire at run time.
- We apply F-measure which is a typical metric to estimate the goodness of a query form [5]. The goodness of a query form is determined by the query results generated from the query form.

3.3 Query Forms

In this section we formally define the query form. Each query form corresponds to an SQL query template.

Definition 1: A query form F is defined as a tuple(A_F, R_F, σ_F, $\bowtie(R_F)$), which represents a database query template as follows:

F = (SELECTA_1,A_2, ...,A_K

FROM $\bowtie(R_F)$ WHEREσ_F),

Where A_F= $\{A_1, A_2, ...,A_K\}$are k attributes for projection, k >0.R_F =$\{R_1,R_2 , ...,R_n\}$ is the set of n relations (or entities) involved in this query, n >0.Each attribute inA_F belongs to one relation in R_F .$\sigma_F$is a conjunction of expressions for relations, $R_F \bowtie(R_F)$ is a join function to generate a conjunction of expressions for joining relations ofR_F.

3.4 Query Result

Many database queries output a huge amount of data instances. In order to avoid this we only output a compressed result table to show a high-level view of the query results first. Each instance in the compressed table represents a cluster of actual data instances. Fig. 2 shows the flow of user actions.

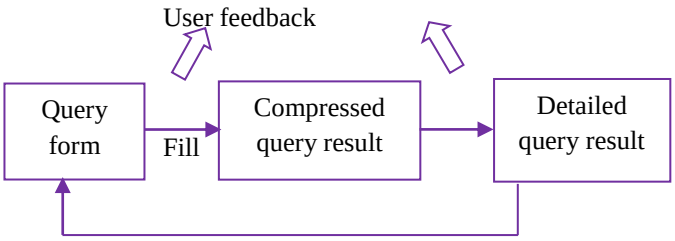

Figure 2. User Actions.

Another important usage of the compressed view is to collect the user feedback. In real world, end-users are reluctant to provide explicit feedback. The click-through on the compressed view table is an implicit feedback to tell our system which cluster (or subset) of data instances is desired by the user.

3.5 Ranking Metric

Query forms are designed to return the user's de-sired result. There are two traditional measures to evaluate the quality of the query results: precision and recall [5]. 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 click-through on query results.

3.6 Estimation of F-Measure

Interactive query form provides a two-level ranked list for the components. The first level is the ranked list of entities. The second level is the ranked list of attributes in the same entity.

The ranking score estimation is achieved by using F-Measure. Given a set of projection attributes A and an universe of selection expressions σ, the expected F-Measure of a query form $F = (A_F, R_F, \sigma_F, \bowtie(R_F))$ is $FScore_E(F)$, i.e.,

$$FScore_E(F_i) = \frac{(1+\beta^2).\,Precision_E(F_i)\,.\,Recall_E(F_i)}{\beta^2.Precision_E(F_i) + Recall_E(F_i)} \quad (1)$$

Notations:TABLE 2 lists the symbols used in this paper. Let F be a query form with selection condition σ_F and projection attribute set A_F. Let D be the collection of instances in $\bowtie(R_F)$. N is the number of data instances in D. Let d be an instance in D with a set of attributes $A = fA_1, A_2, ..., A_ng$, where $n = jAj$. We use d_{AF} to denote the projection of instance d on attribute set A_F and we call it a projected instance. $P(d)$ is the occurrence probability of d in D. $P(\sigma_F\, jd)$ is the probability of d satisfies σ_F.$P(\sigma_E jd)$€ {0, 1}.

Table 2: Symbols and Notations.

F	query form
RF	set of relations involved in F
A	set of all attributes in $\bowtie(R_F)$
$A_r(F)$	set of relevant attributes of query form F

σF	set of selection expressions of query form F
d	data instance in $\bowtie(R_F)$
D	the collection of data instances in $\bowtie(R_F)$
N	number of data instances in D
Q	database query
DQ	results of Q
α	fraction of instances desired by users

Algorithm 2: QueryConstruction

Data:$Q=\{Q_1, Q_2, ...,\}$is the set of previousqueries executed on F_i.

Result:$Q_{one}$is the query ofOne-Query

Begin

$\sigma_{one} \leftarrow 0$

for$Q \in Q$ **do**

$\sigma_{one} \leftarrow \sigma_{one} \vee \sigma$

$A_{one} \leftarrow A_{Fi} \cup A_r(F_i)$

$Q_{one} \leftarrow GenerateQuery(A_{one}, \sigma_{one})$

Algorithm 2 describes the algorithm of the One-Query's query construction.

The function Generate Query is to generate the database query based on the given set of projection attributes A_{one} with selection expression σ_{one}.When the system receives the result of the query from the database engine, it calls the second algorithm of One-Query to find the best query condition. The query results will be clustered using general data clustering algorithm i.e., k-Medoid have been used in this paper. The clusters will be compacted by using the abstract clustering algorithm. Then user will choose clusters based on that f-measure will be calculated and the result will be displayed for the user.

Algorithm 3: FindBestLessEqCondition

Data:α is the fraction of instances desired by user,$D_{Qone}$is the query result of Q_{one}, $A_s$is theselection attribute.

Result:s*is the best query condition ofA_s.

begin

// sort by A_s into an ordered set D_{sorted}

Dsorted ←Sort(DQ_{one} , *As*)

$s \leftarrow \emptyset$, $fscore^{*} \leftarrow 0$

$n \leftarrow 0$, $d \leftarrow \alpha\beta^2$

for $i \leftarrow 1$ **to** $|D_{sorted}|$ **do**

$d \leftarrow D_{sorted}[i]$

$s \leftarrow$ "$A_s \leq d_{As}$"

// compute fscore of "$A_s \leq d_{As}$ "

$n \leftarrow n + Pu(dA_{Fi})P\ (dA_{Fi})P\ (\sigma F_i\ |d)P\ (s|d)$

$d \leftarrow d + P\ (d_{AFi})P\ (\sigma_{Fi}\ |d)P\ (s|d)$

$fscore \leftarrow (1 + \beta^2) \cdot n/d$

if $fscore \geq fscore$ **then**

$s^{*} \leftarrow s$

$fscore^{*} \leftarrow fscore$

3.7 PERFORMANCE EVALUATION

3.7.1 Experimental Setup

We implemented interactive query forms as a web-based system using Java Development Kit [JDK] 1.6 with Java Server Page. The runtime web interface for the query forms using open-source JavaScript library jQuery 1.4. We are using MySQL as the database engine. These experiments are planning to run using a machine with Intel Core 3 CPU @2.83GHz, 1GB main memory, and running on Windows XP SP2.

Data Sets: Database: Educational database.

The below fig.3(a) shows the F-Measure graph which is used to calculate the goodness of the query form. Ranking score is a supervised method to measure the accuracy of the recommendation. The fig.3(b) shows the Average Ranking Score of the Interactive Query Form. The run-time cost of ranking projection and selection components for IQF depends on the current form components and the query result size**.**

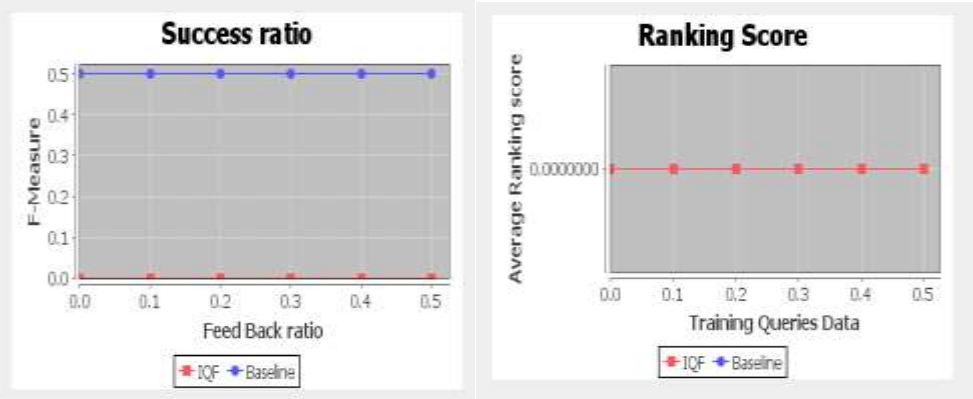

Figure 3. (a) Average F-Measure. (b) Average Ranking Score.

4. CONCLUSION

Query interfaces play a vital role in determining the usefulness of a database. A form-based interface is widely regarded as the most user-friendly querying method. In this paper, we have developed mechanisms to overcome the challenges that limit the usefulness of forms, namely their restrictive nature. In this paper we propose an interactive query form generation approach which helps users to dynamically generate 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.

REFERENCES:

[1] Cold Fusion. http://www.adobe.com/products/coldfusion/.

[2] EasyQuery. http://devtools.korzh.com/eq/dotnet/.

[3] 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.

[4] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In CIDR, 2003.

[5] G. Salton and M. McGill. Introduction to Modern Information Retrieval. McGraw-Hill, 1984.

[6] M. M. Zloof. Query-by-example: the invocation and definition of tables and forms. In Proceedings of VLDB, pages 1–14, Framingham, Massachusetts, USA, September 1975.

[7] 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.

[8] M. Jayapandian and H. V. Jagadish. Automating the design and construction of query forms. IEEE TKDE, 21(10):1389– 1402, 2009.

[9] 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