Dynamic Query form Generation for Mining Highly Optimized Transactional Databases

Pooja Suthar\textsuperscript{1} Ms. Jasmin Jha\textsuperscript{2}
\textsuperscript{1}M.E Student \textsuperscript{2}Assistant Professor
\textsuperscript{1,2}Department of Computer Engineering
1,2 L. J. Institute of Engineering & Technology, Ahmedabad, Gujarat, India

Abstract— Modern scientific databases and web databases maintain large and heterogeneous data. DQF is a novel database query form interface, which is able to dynamically generate query forms. The objective of DQF is to allow a normal user with less or no knowledge of querying language to generate dynamic query forms as per their requirements. The enhancement of a query form is an iterative process and is guided by the user. User selects require query form components to add in desired query form and a query form could be dynamically refined till the user satisfies with the query results. A probabilistic model is developed for implementing a query form in DQF. Our experimental evaluation demonstrate the efficiency and customisability of the system. Result shows that the dynamic approach often leads to higher success rate and simpler query forms compared with a static approach.

Key words: Dynamic Query Forms, Dynamic Query Form Generation, Dynamic Query Form searching

I. INTRODUCTION

In large databases such as text databases, web databases, scientific databases contains large data. Those databases contain large number of data instances that has various relations and attributes. The previously defined query forms are not able to fulfil large ad-hoc queries by users on those databases. Dynamic query form is a new approach for generating database query interface that is used to dynamically generate query forms. Use of DQF is to take a user’s interests, preferences and rank QF components, guiding user to make their decisions without even typing queries. The DQF generation is an iterative process. It is guided by the user. This system itself generates lists of query form components and then user adds the required form components into the query form. A user may also modify pre-generated query forms and by filling in the query form and submit it to see the result. A query form is dynamically enhanced until the user is satisfied with the displayed query results.

Previous systems are predefined and designed by developers, professionals, DBAs in various information and data management systems. As with development of web data and traditional database lead today’s databases to become very large and complex. Databases have over thousands of tuples and hundreds of entities. Many of the web databases, like DBPedia\textsuperscript{7} and Freebase\textsuperscript{9} typically have thousands of web entities. It is hard to design a group of static QF to satisfy various database queries on such complex and large databases. The data development and database management tools, like, EasyQuery\textsuperscript{13}, Oracle BI, Siebel, Microsoft BI, SAP, Cold Fusion\textsuperscript{11}, and MS Access, they provide such functionalities to allow users generate customized queries. If the user is not known to the schema of the database then those large number of data attributes will confuse the user.

Query form is the most widely used and manageable user interfaces for querying the data in databases Traditional QFs are either static query forms or customized QFs that are predefined and/or designed by developers, DBAs in various IMSs. DQF system is a query data interface that’s capable of dynamically creating QF for users. DQF is different from traditional data retrieval techniques in which users in DB result retrieval perform many rounds of actions before identifying the final query form. The main objective of DQF is to take user interests during user’s interactions with system and to update the query form with every iteration\textsuperscript{11}.

Previously proposed automatic approaches for generating query forms for databases that didn’t contain user participation. That presented an approach called data-driven approach. First of all it searches a group of attributes that are most likely required based on the schema and instances. After that query forms are created based on the selected data attributes.

II. RELATED WORK

In [1], Liang Tang et al presented Dynamic Query Form System-DQF that is a querying interface that allows dynamically generating query forms. It is different from traditional document retrieval system where users are often willing to perform number of rounds of operation to get desired result. In this system, DQF captures user likelihood during user’s interactions and update the query form by every iteration. The main two functions performed in this system was query form enrichment and query execution.

DQF first generates basic query form and enrich this form by iteratively via interaction with users. It applies F-measure to find the wellness of a QF that is taken by the query results created from the QF\textsuperscript{11}. DQF ranks and recommends the desired query form components so that users can find the query form more easily. The query form is first generated using attributes of projection, set of relations on joined relations. From this query form, system will generate queries and execute. It generates the query form and iteratively improves the query form and ranks the components based on suitability to user’s interest. It computes the DQF which shows the running time grows almost linearly with respect to the query result size\textsuperscript{11}.

In [2], B. Ramaganesh and Lokanath explained Query structure that is a standout amongst the most generally utilized client between faces for questioning databases. Not the same as customary record recovery, clients in database recovery are regularly ready to perform numerous steps of activities such as refining question conditions before recognizing the actual
Every cycle comprises of two sorts of client connections: Query Form Enrichment and Query Execution. In query form enrichment, it suggests a rank list of components of QF to the user. The user selects the required form components into the current QF. In second part, Query execution, the user fills data in current QF and submits a query. It runs the query and displays the results. At the end, a user gives the feedback regarding the query results. Query enrichment is done by selecting ranked attributes from the attribute set. if user clicks some data instances then those data instances should have high user interests. The query form components that can capture these data instances should be ahead in ranking than other.

In [3], Magesh Jayapandian and H. V. Jagadish explained automating the task of query form creation to significantly reduce, if not eliminate, the task of developer in the process. It is an approach of designing a form interface that increases the expressivity while considering specific upper-bounds on interface complexity[2]. Most current databases, have much more complex schemas with large number of entities, relationships and attributes. Many of these databases are normalized to a large scale such that it increases the number of entities. It is needed to capture the data of interest[2]. In most cases the schema complexity is defined simply by the richness of the data. This complexity is showed in the querying the database, many with entity of interest are more than one[2].

In this paper, they described an approach that breaks design of forms interface problem into three challenges. They are

1) The first challenge is identifying the schema components that are most likely to be querying by the user. It also describes techniques for estimating queryability based upon the database schema and data content, and use it to remove some schema elements that are less queried.

2) The second challenge in automated form design is to partition the filtered collection of schema elements into groups. It is done in such a way that the attributes, entities and relationships present in a single group are interrelate on a form to explain user queries.

3) The third challenge is to change each of the groups of schema elements into a form that a user can use to express a required query.

This form generation system can indeed produce forms, of manageable number and complexity that are capable of posing a majority of user queries to a given database, using just its schema and its data content. The experiment done on database used in this paper shows that it supported more than twice as many queries as the form for a major commercial website, Monster.com[2]. However the system does not satisfy the most complex of queries which can be of interest to users.

In [4], M. Jayapandian et.al explains a form-based query interface, which only requires filling blanks to specify query parameters, is valuable since it helps make data accessible to users with no knowledge of formal query languages or the database schema. In this paper the form generation technique is to design forms for an entire set of queries and applies form complexity threshold (FCT): a measure of complexity that to control a form’s readability [3]. This paper presents a study of system performance using a real query trace, as well as queries from a standard XML benchmark [3]. It describes an automated self-managing interface-builder will help bring novice users closer to the rich database resources they need to use.

In [5], it has presented form-based interfaces combined with and keyword search for ad-hoc querying databases. It takes a target database as input and then create and index a set of query forms offline by system. A user with a question to be answered issues standard keyword search queries at query time; but the system returns forms relevant to the question instead of returning tuples[4]. Then the user builds a structured query with one of the forms and submits it back to the system for execution. To let users build ad hoc queries with forms, it should create forms that are easy to use and that support a wide range of queries[4]. Here, D be a database instance and SD be the schema of D.

They describe form generation as a four step procedure:

1) Step-1 Determine a subset of SQL as the target language to implement the queries supported by forms.
2) Step-2 Specify a set of skeleton templates specifying the main clauses and join conditions based on the chosen subset of SQL and SD.
3) Step-3 Finalize the template by changing skeleton templates based on the required form specificity.
4) Step-4 Map each template to a form.

For keyword search in forms, this paper discusses three methods- Naïve, Double-index OR and Double Index-AND, Double Index Join.

Naïve simply retrieves a form if the form contains at least one (OR semantics) or all (AND semantics) of the terms from a keyword query[4]. If a user will get no answer if user specifies a data value and system will use Naïve-AND. If system use Naïve-OR, and if the user includes in the query at least one schema term i.e., a term that matches a table or attribute name then some forms would be returned. However, the data terms i.e., terms that match data values, if any, would be completely ignored, which is not very satisfying [4].

In DI-OR, add all schema terms with respect to the data terms in a keyword query from a user, and to evaluate it using OR semantics. In DI-AND approach with AND semantics, they enlarge the original query by creating all possible queries that result from replacing user-supplied data terms with schema terms, use AND semantics for each query, and return the union of the query results to the user[4].

DIJ is similar to DI-AND but it filters out “Dead” forms. They modified DI-AND to filter a common type of “dead” forms with the guarantee that no live forms will be eliminated[4]. Before returning a form, it is used to see if the form will return an answer if instantiated with the data terms in the user query.
III. QUERY FORM INTERFACE

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

B. Definition
A query form F is defined as a tuple (AF, RF, σF, ><(RF)), which represents a database query template as follows:

\[ F = (\text{SELECT} \ A_1, A_2, \ldots, A_k \ \text{FROM} \ ><(RF) \ \text{WHERE} \ \sigma_F), \]

where AF = \{A_1, A_2, \ldots, A_k\} are k attributes for projection, k > 0. RF = \{R_1, R_2, \ldots, R_n\} is the set of n relations (or entities) involved in this query, n > 0.

Each attribute in AF belongs to one relation in RF. σF is a conjunction of expressions for selections (or conditions) on relations in RF. ><(RF) is a join function to generate a conjunction of expressions for joining relations of RF.

In the user interface of a query form F, AF is the set of columns of the result table. σF is the set of input parameters to generate different queries. RF and ><(RF) are not visible in the user interface, which are usually generated by the system according to the database schema. For a query form F, ><(RF) is automatically constructed according to the foreign keys among relations in RF. Meanwhile, RF is determined by AF and σF. RF is the union set of relations which contains at least one attribute of AF or σF. Hence, the components of query form F are actually determined by AF and σF. As we mentioned, only AF and σF are visible to the user in the user interface. In this paper, we focus on the projection and selection components of a query form. Our dynamic query form can be easily extended to include those options by implementing them as dropdown boxes in the user interface of the query form.

IV. METHODOLOGY
The main disadvantage of the existing system is, it is created and used by developers and professionals that have knowledge of querying language. A normal user who doesn’t have enough knowledge about query language can’t easily access the system. The previous system don’t allow to create as well as search query forms dynamically together in one system. User can dynamically create query forms yet it doesn’t allow efficient searching on previously generated query forms.

A. Process Is As Follows
The system starts with a basic query form that has very few attributes of the database. The basic query form enrichment is done iteratively by the interactions of the user and the system until the user gets satisfied query result. In order to extend the system performance, security and I will make the system suitable for private database applications using user authentication. The user is provided a keyword search option as an entry to database access. The user is provided with text-box to input keywords. Based on these keywords, the query forms can be searched and its components can be updated at each step.
C. Query Form Enhancement

User selects the query form to which user wants to enhance either by adding or removing attribute from the query form. User dynamically choose the attributes for the query form and save it. User can also delete query form if needed.

D. Query Form Searching

Query Form searching is provided in the system in which system allows user a keyword based search in which user provides a keyword that is used to search query form either by its name or by attribute

1) Search by Name

In this type of searching, User wants to search a query form by its name, chooses search by name and provides a keyword that is compared with the name of all query forms and displays the result.

2) Search By Attributes

In this type of searching, User wants to search a query form by its attribute, chooses search by attribute and provides a keyword that is compared with the attributes of all query forms and displays the resultant query forms that contains that attribute.

V. Evaluation

I implemented the dynamic query forms as a web based system using MVC .net 4 with C# language. The dynamic web interface for the query forms used open source java script library jQuery 1.4. I used SQL Server 2005 as the database engine. All experiments were run using a machine with Intel Core 2 CPU @2.83GHz, 3.5G main memory, and running on Windows 7.

To demonstrate the effectiveness of the approach, I performed several experiments on a randomly generated query forms. The dataset is a transactional dataset containing several records and 44590 instances. The experiment is performed on core i5 processor with 4 GB RAM.

To achieve result analysis, I have carried out five test queries that are generated from dynamic query forms. All these queries are used to test the scalability of the system.

<table>
<thead>
<tr>
<th>Query</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>SELECT Id, year, firstname, lastname, conference, leag, gp, minutes, reb, asts, stl, turnover, fga FROM player_allstar</td>
</tr>
<tr>
<td>Q2</td>
<td>SELECT firstname, lastname, leag, gp, minutes, pts, dreb, oreb FROM coaches_career</td>
</tr>
<tr>
<td>Q3</td>
<td>Select team, location, name, leag from teams</td>
</tr>
<tr>
<td>Q4</td>
<td>Select team, year, leag, o_fgm, o_fga, o_ftm, o_fta, o_oreb from team_season</td>
</tr>
<tr>
<td>Q5</td>
<td>Select reb, asts, stl, blk, turnover, gp, minutes, pts FROM player_allstar</td>
</tr>
</tbody>
</table>

Table 5.1: Test Queries derived from query forms

A. Efficiency Analysis

The run-time cost for form components for DQF is depended on the selected form components and the returned query result size. So we selected 5 test queries that has large
data set result size. The retrieving result is based on the time required by the query form to give actual data from database. All the result are in millisecond. The time for execution is between 1 to 1.5 seconds for our database where the results takes up to 10000 records.

Based on all these records the following figures shows the result of running time of all the queries from Q1 to Q5 against increase in data records. From the figures it is clear that all the records up to 10000 are mined in less than 2 seconds.

![Fig. 5.1: Execution time by varying number of records for Q1](image1)

![Fig. 5.2: Execution time by varying number of records for Q2](image2)

![Fig. 5.3: Execution time by varying number of records for Q3](image3)

**B. Complexity Analysis**

Complexity of query forms are number of attributed allowed to generate query forms that are feasible for end users. The complexity analysis is taken for the system where various number of dynamic query forms are analysed. Based on the taken query forms the result analysis shows that the number of attributes should be range of 3 to 15 that will give feasible result for end users.

<table>
<thead>
<tr>
<th>No of Attributes</th>
<th>No of rows</th>
<th>No of data instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>920</td>
<td>13800</td>
</tr>
<tr>
<td>5</td>
<td>884</td>
<td>12376</td>
</tr>
<tr>
<td>8</td>
<td>606</td>
<td>10478</td>
</tr>
<tr>
<td>12</td>
<td>448</td>
<td>1776</td>
</tr>
<tr>
<td>13</td>
<td>68</td>
<td>480</td>
</tr>
<tr>
<td>14</td>
<td>90</td>
<td>550</td>
</tr>
<tr>
<td>15</td>
<td>54</td>
<td>216</td>
</tr>
</tbody>
</table>

**Table 5.2: data instances by varying no of attributes**

![Fig. 5.4: Execution time by varying number of records for Q4](image4)

![Fig. 5.5: Execution time by varying number of records for Q5](image5)

![Fig. 5.6: Data instances by varying number of attributes](image6)
Fig 5.6 shows that when number of attributes per dynamic query form increases, the retrieved data instances decreases that will be feasible for end uses to manage the data instances up to 15 attributes. Therefore, the complexity of attributes per dynamic query form should be taken wisely attributes per form. As per result analysis the number of attribute per form should be between 3 to 15 attributes per form to manage its results.

VI. CONCLUSION

The paper illustrates method used for mining large databases in an efficient way. In modern era, Dynamic Query Form is beneficial in large databases that have over hundreds or thousands of attributes and relations. Existing systems focused on statically generating and customizing query forms. It did not allow users to dynamically generate query form at run time. Dynamic Query Form allows dynamically generating updated query form that will be helpful in required searching. Its dynamic approach will lead to simpler query forms compared with existing approaches and also allow efficient search on generated Dynamic Query Forms.

As a future work, the system can be extended for different database types that will allow user to add any type of database to the system to bind and generate efficient dynamic query forms from the system.

REFERENCES


