Automating the Design and Construction of Query Forms [*]

Magesh Jayapandian H. V. Jagadish

University of Michigan
Ann Arbor, MI 48109-2122
{jmagesh,jag}@umich.edu

Abstract

One of the simplest ways to query a database is through a form, where a user can fill in relevant information and obtain desired results by submitting the form. Designing good static forms is a non-trivial manual task, and the designer needs a sound understanding of both the data organization and the querying needs. Furthermore, form design has two conflicting goals: forms should be simple to understand, and at the same time must provide the broadest possible querying capability to the user. In this paper, we present a framework for generating forms in an automatic and principled way, given the database schema and a sample query workload. We design a tunable clustering algorithm for establishing form structure based on multiple "similar" queries, which includes a mechanism for extending form structure to support other "similar" queries the system may see in the future. The algorithm is adaptive and can incrementally adjust the form structure to reflect the addition or removal of queries in the workload. We have implemented our form generation system on a real database and evaluated it on a comprehensive set of query loads and database schemas. We observe that our system can significantly reduce the numbers of forms needed for various query loads by exploiting similarities across queries, even after placing a strict bound on form complexity.

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-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 practice, form-based interfaces are used frequently, but usually, each form is designed ad hoc and its applicability is restricted to a small set of fixed queries.

Query languages were developed to specify to a database engine how a query should be evaluated and not to help users understand the semantics of the query and decide if it matches the query in their head. Furthermore, given a query in a declarative language, it is not always obvious how to create a corresponding form that is comprehensible to the user and captures the information required. Although the constraint predicates and return attributes are constant and dictated by the query, it does not directly specify the structural relationships between the attributes involved, nor does it suggest how they can be presented to a user in a meaningful way. It is easy for a person conversant with the query and the underlying data structure to design a form for it, and map the user-input fields to the appropriate query predicates. While such a form would do the job very nicely for the query at hand, it is usually not extensible, and brings to bear external human knowledge. As queries and schema become more complex, manual form generation is no longer easy, and hidden assumptions come into play much more frequently. On the other hand, creating forms automatically is far from trivial because it is difficult to satisfy simultaneously three important properties desired of any form-based interface: *conciseness*, *clarity* and *expressivity*. In this paper, we present an automated technique to generate a form-based query interface that maximizes these qualities.

2. Form Generation

To design a form for a declarative query, 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

[*]Supported in part by NSF 0438909, NIH 1-U54-DA021519-01A1, DARPA W81XWH-04-2-0012, and Michigan Economic Development Corporation-Life Sciences Corridor Fund GR-238.

elements in groups, label them suitably, and lay them out in a meaningful way on the form. Our algorithm is presented in Fig. 1. We extend the form generation technique to design forms for an entire set of queries. Given a set of interesting queries, the naïve approach would be to build one form for each one of them. However, queries against a single schema will more often than not have similarities between them, which we can exploit to minimize redundancy. To control a form's readability, we introduce a *form complexity threshold* (FCT): a measure of complexity that we would like no form to exceed. However, this threshold may be unenforceable if set too low, or if some of the queries involved are too complex. In such cases, even a single query may have complexity that exceeds the FCT. In the general case, FCT is satisfied by splitting a query cluster covered by a complex form into smaller clusters covered by simpler forms. To measure how useful a form is, we define its *expressivity* as how many different queries it can express.

3. Evaluation

To evaluate system performance we conducted three sets of experiments. We used queries in the XMark benchmark [1, 2], which are relatively disjoint, and built a corresponding set of forms. We also obtained a query trace of real web databases hosted on our EECS department's database server and used them to build forms incrementally.

XMark Benchmark: The XMark benchmark [1, 2] was designed to test the various capabilities of an XQuery implementation. The queries are broad in scope and bear little similarity to one another. Such a setup is far from ideal for our system, as we expect that users of a real database often have interests similar to other users, and tend to pose queries that have parts in common with queries posed by others. However, we saw this diverse query set as an opportunity to test the worst case performance of our system.

The benchmark has 20 queries in total which is fairly tiny compared to a real query log. We ran our form-generator on this query set for varying thresholds on form-complexity and our results are displayed in Fig. 2. Despite the high variance between queries, we only needed to create 7 forms for a reasonable upper bound of 25 elements per form. We computed the expressivity of the forms with the same thresholds and observed, as expected, that the number of queries expressible using the form-set increased when the threshold was raised. It was interesting to note that by placing a reasonable bound of 10 elements-per-form, the set of 7 forms that were generated (using only 20 queries) could be used to express close to 5 million different queries. But the expressivity does not go up smoothly as the FCT is increased. Depending on precisely which forms get retired and which new forms get pressed into service, there may even be a small drop in expressivity as FCT is increased.

```
Algorithm GenerateForm
Input: A query $Q$ (as an Evaluation Plan)
Output: A form $F$
// Element Construction and Grouping
Create a new form-group $g$ and add it to the form-tree $T$;
foreach operation $o \in Q$ when traversed top-down do
    case $o$ is a "selection"
        Create a constraint-element using the selection
        predicate;
        Put this constraint-element in $g$;
    case $o$ is a "projection"
        Create a result-element using each projected
        attribute;
        Put these result-elements in $g$;
    case $o$ is an "aggregate function"
        Create an aggregate-element using the the group-by
        attribute, the grouping-basis and the aggregate
        function;
        Put this aggregate-element in $g$;
    case $o$ is a "join"
        Create a join-element using the two (left and right)
        attributes of the join condition;
        Put this join-element in $g$;
        Create a new group $g'$ as a child of $g$ in $T$;
        Set $g \leftarrow g'$;
end
// Element and Group Labeling
foreach form-group $g \in T$ do
    Label $g$ relative to its parent group (use absolute path if
    $g$ is the root);
    foreach form-element $e \in g$ do
        Label $e$ relative to $g$;
    end
end
```

Figure 1: Algorithm `GenerateForm`

The overall trend in the relationship between the number of form-elements and expressivity is exponential, in accordance with our theoretical analysis.

EECS Query Trace: We now report on experiments run with a real workload obtained from a log consisting of queries posed during a single day, to web databases hosted on a MySQL server in our EECS department. We analyzed 10,000 queries in the workload, and observed the following characteristics: In all, 17 different databases were accessed by users over the approximately 1-day period, and queries involved a total of 115 different tables (not necessarily the complete set of tables). The queries themselves had an average close to 1 selection predicate and 5 projected attributes. On average, 1 out of every 13 queries involved an aggregate computation, and about 1 out of every 64 queries involved a join. Fig. 3 shows the effect of threshold on the size of the form-set we generated. The results do not differ much from the previous experiment, except that now well under

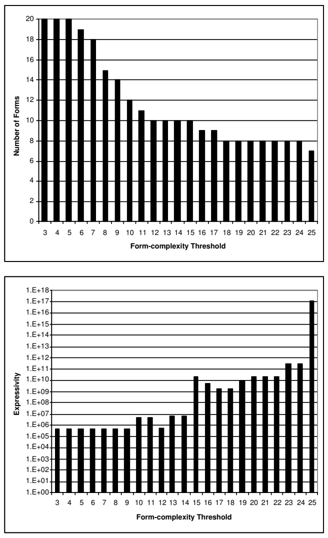

Figure 2. Forms for XMark Queries

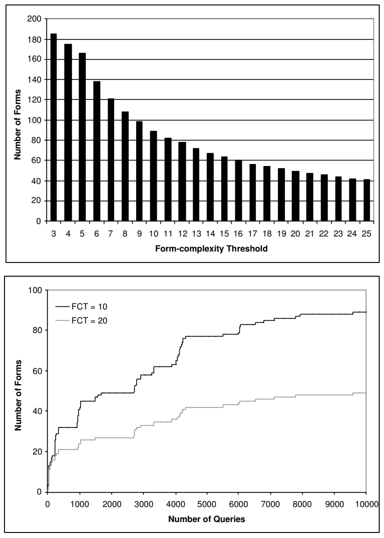

Figure 3. Forms for EECS Query Trace

100 forms can cover 10,000 queries. We also see that the curve is smoother than for the smaller XMark query set.

Given the full workload, it is more interesting to understand the dynamics of form creation. Therefore, we created the forms incrementally, starting from scratch. Fig. 3 shows the growth in the size of the form-set with each query analyzed. We performed this experiment with two different complexity-thresholds: 10 form-elements-per-form, and 20 form-elements-per-form. We observed an initial learning period in which a large number of forms are created, but following this, the rate of form creation dropped significantly. With a complexity-threshold of 10, the first 1,000 queries required the creation of 41 forms, while the next 9,000 queries only required 48 more forms. With a threshold of 20, only 24 forms were needed to express the first 1,000 queries, and only 25 more for the remaining 9,000. As we can see from the figure, the curve flattens towards the end of the workload, as fewer and fewer new queries are found substantially dissimilar to ones seen before.

Discussion: Our experiments show how the complexity threshold affects size and expressivity of the form-set, but the results are of little use if no meaningful insight can be obtained from them. The asymptotic nature of Figs. 2 and 3 helps us choose ideal values for this threshold in the 3 scenarios, to best balance complexity versus size. Based on our observations, we could recommend a DBA to set the threshold at about 12-15, since beyond this range, increased complexity shows little gain in terms of interface simplicity. The same can be said for the expressivity of the interface as well, further strengthening the case for this trade-off point. Obviously, this is a design choice that depends heavily on the distribution of queries, but having analyzed several disparate workloads (some not presented in this paper), we believe that this is a reasonable rule of thumb for a practitioner.

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 and the tedious manual effort required to construct them. Specifically, we introduced an algorithm to generate a set of forms automatically given the expected query workload. We presented a study of system performance using a real query trace, as well as queries from a standard XML benchmark. We feel that such an automated self-managing interface-builder will help bring novice users closer to the rich database resources they need to use, and maximize their efficiency with a sizeably reduced learning curve. While our focus has been on pushing the limits of automated form generation with minimal human input, it is conceivable that appropriate hints from a human could produce even better results. Techniques to exploit such complementarity are the subject of future work.

References

[1] XMark: An XML Benchmark Project: http://www.xml-benchmark.org/.

[2] A. R. Schmidt et al. The XML Benchmark Project. Technical Report INS-R0103, CWI.