**IMPACT: International Journal of Research in Engineering & Technology (IMPACT: IJRET)**
**ISSN(E): 2321-8843; ISSN(P): 2347-4599**
**Vol. 2, Issue 7, Jul 2014, 157-162**
**© Impact Journals**

# DYNAMIC QUERY FORMS WITH NoSQL

**VISHNU R[1] & SWAPNA HARI[2]**

[1]Research Scholar, Department of Computer Science & Engineering, Marian Engineering College,
Trivandrum, Kerala, India

[2]Assistant Professor, Department of Computer Science & Engineering, Marian Engineering College,
Trivandrum, Kerala, India

## ABSTRACT

The current trends in technology like Big data, Big user and Cloud computing that leads to the adoption of NoSQL. NoSQL means Not Only SQL. Today most of the applications are hosted in cloud and that are available through internet. They must support large number of users 24 hours a day, 365 days a year. This create an increase in number of concurrent users. So here needs a technique to handle large number of data. Proposes a novel dynamic query form interface(DQF) using NoSQL for database exploration of an organization. Here use a document oriented NoSQL database ie, MONGODB. MONGODB support dynamic queries that do not require predefined map reduce function. The generation of a query form is an iterative process and is guided by user. At each iteration, system automatically generate ranking list of form components and user adds the desired form component into query form then submit queries to view query result. There are two traditional measures to evaluate the quality of query result i.e.: precision and recall. From the quality measures we can derive overall performance measures as F-measure.

**KEYWORDS:** Query Form, NoSQL, User Interaction

## 1. INTRODUCTION

Query forms are the most widely used user interfaces for querying databases. Traditional query forms are designed and predefined by developers or DBA in various information management systems. Many web databases such as Freebase and DBPedia typically have thousands of structured web entities[2][3]. Therefore, it is difficult to design a set of static query forms to satisfy various ad-hoc database queries on those complex databases. The queries on a database are usually expressed in high level query languages such as SQL. This works well for many applications, but it is not a fully satisfying way of finding data. For naïve users these systems are difficult to use and understand, and they require a long training period. Clearly there is a need for easy to use, quick and powerful query methods for database retrieval. A query interface(DQF with NoSQL) is proposed which is capable of dynamically generating query forms for users. The essence of DQF is to capture user interests during user interactions and to adapt the query form iteratively. Dynamic query form systems were introduced to generate the query forms according to the user’s desire at run time. Modern databases become very large and complex and therefore it is very hard to manage using traditional relational database management systems. NoSQL technology has the answer to all these problems. NoSQL databases are often highly optimized key– value stores intended for simple retrieval and appending operations. These are used in big data & real-time web applications. It employs less constrained consistency models than traditional relational database management systems. The rest of the paper is organized as follows. Section 2 describes the system architecture.

Section 3 defines the query form interface and query results. Section 4 defines the ranking metric used. Section 5 describes the comparison of SQL and NOSQL in accordance with the dynamic query form and finally Section 6 concludes the paper.

## 2. SYSTEM ARCHITECTURE

The system proposed have the following modules along with functional requirements.

### Component Ranking Module

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. In this way, a query form could be dynamically refined till the user satisfies with the query results. The form components here refers to the selection and projection components. DQF provides a two-level ranked list for projection components. The first level is the ranked list of entities. The second level is the ranked list of attributes in the same entity. The selection attributes must be relevant to the current projected entities; otherwise that selection would be meaningless. Therefore, the system should first find out the relevant attributes for creating the selection components. Here first describe how to select relevant attributes and then describe a naive method and a more efficient one-query method to rank selection components.

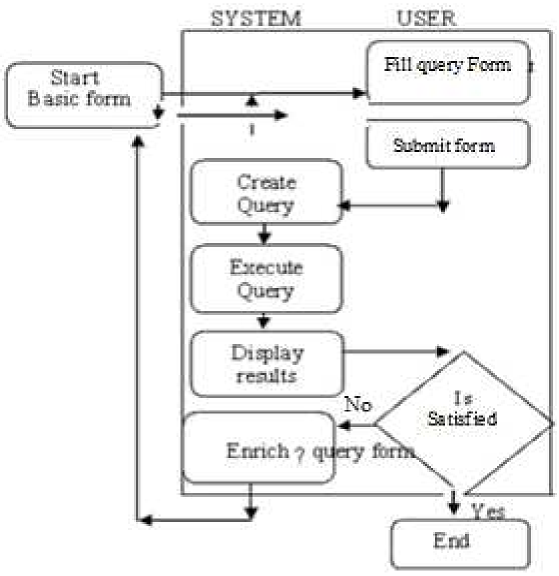

**Figure 1: Flowchart of Dynamic Query Form**

### Quality Metric Module

The quality of query result can be described by paying more importance to precision and recall. Precision is also called positive predicate value. It is the fraction of retrieved instance that are relevant. Recall is also called sensitivity. Recall is the fraction of relevant instance that are retrieved. We use expected precision and expected recall to evaluate expected performance of query form. Probabilistic model can be used to find precision and recall.

### Metadata Processor Module

Metadata can be defined as the data providing information about one or more aspects of the data. This provide user friendly interface to novel users. Map-reduce function is used to extract keys from collection. In map-reduce operation, our NoSQL database MONGODB[6] applies `map' phase to each input documents. The `map' function emit key-value pairs. For those keys have multiple values, MONGODB applies the `reduce' phase, which collects and condense the aggregated data. Map-reduce operations take the documents of a single collection as the input and can perform any arbitrary sorting and limiting before beginning the map stage. MapReduce can return the results of a map-reduce operation as a document, or may write the results to collections. The input and the output collections may be sharded. Mongodb application use DBref() method to relating documents. DBRefs are references from one document to another using the value of the first document's id- field, collection name, and, optionally, its database name. By including these names, DBRefs allow documents located in multiple collections to be more easily linked with documents from a single collection. As a result proposed system iteratively generate more condition that are desired by user.

**Table 1: SQL to MONGODB Mapping Chart**

| SQL Terms/Concept | MONGODB Terms/Concept |
|---|---|
| Database | Database |
| Table | Collection |
| Row | Document/BSON Document |
| Column | Field |
| Index | Index |
| Table Join | Embedded document or linking |
| Specify any unique column or column combination as primary key | In MONGODB, primary key is set to the _id field |

**Query Processor Module**

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. They are query form enrichment and query execution. Dynamic query form generates a ranked list of query form components to the user. So that user can select the desired form components from the current query form. Query execution is performed by submitting the current query form. Which displays the query results and based on this displayed results user can provide feedback to the system about the query results.

## 3. QUERY FORM INTERFACE

### 3.1 Query Results

To decide whether a query form is desired or not, a user does not have time to go over every data instance in the query results. In addition, many database queries output a huge amount of data instances. To avoid this ―Many-Answer‖ problem [4], we provide 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. Then, the user can click through interested clusters to view the detailed data instances. Figure 2 shows the flow of user actions. The compressed high-level view of query results is proposed in [5]. There are many one-pass clustering algorithms for generating the compressed view efficiently. Certainly, different data clustering methods would have different compressed views for the users. Also, different clustering methods are preferable to different data types. The importance of the compressed view is to collect the user feedback. From the collected feedback, the goodness of a query form can be estimated and so that we could recommend appropriate query form components. The click-through on the compressed view table is an implicit feedback to tell our system which cluster of data instances is desired by the user.

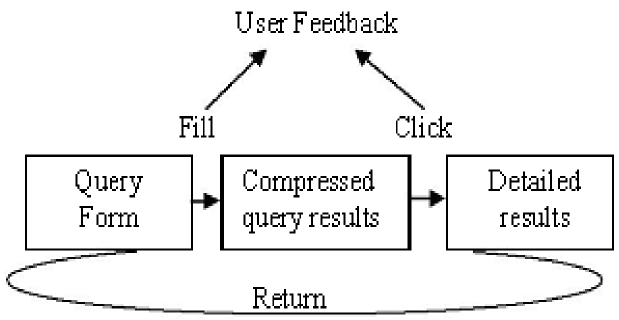

**Figure 2: User Actions**

## 4. RANKING METRICS

The two traditional measures to evaluate the quality of the query results are precision and recall [7]. 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. Both measures are based on user interested data instances. The user interest is estimated based on the user's click through on query results displayed by the query form. The data instances which are clicked by the user must have high user interests and the query form components which can capture these data instances should be ranked higher than other components. Given a set of projection attributes $A$ and a universe of selection expressions $\sigma$, the *expected precision* and *expected recall* of a query form *F are denoted as* $Precision_E(F)$ and $Recall_E(F)$.

$Precision_E(F)$ is defined as the expected number of data instances in the query result that are desired by the user from the total number of instances in the result. $RecallE(F)$ is defined as the expected number of data instances in the query result that are desired by the user from the expected number of instances desired by the user in the whole database. From these two measures, we can calculate the overall performance measure, *expected F-Measure* as shown in Equation 1. This *F-Measure* will give the goodness of the query form and thus we can refine the form until it satisfies the user conditions.

$$FScore_E\,(F) = \frac{(1+\beta^2).\,Precision_E(F).\,RecallE(F)}{\beta^2\,.Precision_E(F)+RecallE(F)} \qquad (1)$$

$\beta$ is a constant parameter to control the preference on *expected precision* or *expected recall*. $FScore_E(F_{i+1})$ is the estimated goodness of the next query form $F_{i+1}$. The aim is to maximize the goodness of the next query form, the form components are ranked in descending order of $FScore_E(F_{i+1})$. $FScore_E(F_{i+1})$ is obtained as follows.

$$FScore_E(F_{i+1}) = \frac{(1+\beta^2).\,Precision_E(F_{i+1}).\,RecallE(F_{i+1})}{\beta^2.Precision_E(F_{i+1})+RecallE(F_{i+1})} \qquad (2)$$

## 5. SQL VS NoSQL

The industry has been dominated by relational databases for 40 years, but application developers are increasingly turning to NoSQL databases to meet new challenges. Relational and NoSQL data models are very different. The relational model takes data and separates it into many interrelated tables. Each table contains rows and columns where a row might contain lots of information about a person and each column might contain a value for a specific attribute associated with that person, like his age. Tables reference each other through foreign keys that are stored in columns as well. NoSQL databases have a very different model. For example, a document-oriented NoSQL database takes the data you want to store and aggregates it into documents using the JSON format. Each JSON document can be thought of as an object to be used by your application. A JSON document might, for example, take all the data stored in a row that spans 20 tables of a relational database and aggregate it into a single document/object.

Aggregating this information may lead to duplication of information, but since storage is no longer cost prohibitive, the resulting data model flexibility, ease of efficiently distributing the resulting documents and read and write performance improvements make it an easy trade-of for web-based applications. Developers generally use object-oriented programming languages to build applications. It's usually most efficient to work with data that's in the form of an object with a complex structure consisting of nested data, lists, arrays, etc. The relational data model provides a very limited data structure that doesn't map well to the object model. Instead data must be stored and retrieved from tens or even hundreds of interrelated tables. Object-relational frameworks provide some relief but the fundamental impedance mismatch still exists between the way an application would like to see its data and the way it's actually stored in a relational database. Document databases, on the other hand, can store an entire object in a single JSON document and support complex data structures. This makes it easier to conceptualize data as well as write, debug, and evolve applications, often with fewer lines of code.

Another major difference is that relational technologies have rigid schemas while NoSQL models are schema less. Relational technology requires strict definition of a schema prior to storing any data into a database. Changing the schema once data is inserted is a big deal. With relational technology, changes like these are extremely disruptive and frequently avoided, which is the exact opposite of the behavior desired in the Big Data era, where application developers need to constantly and rapidly incorporate new types of data to enrich their applications. In comparison, document databases are schema less, allowing us to freely add fields to JSON documents without having to first define the changes.

The format of the data being inserted can be changed at any time, without application disruption. This allows application developers to move quickly to incorporate new data into their applications. NoSQL databases were developed from the ground up to be distributed, scale out databases. They use a cluster of standard, physical or virtual servers to store data and support database operations. To scale, additional servers are joined to the cluster and the data and database operations are spread across the larger cluster. Since commodity servers are expected to fail from time-to-time, NoSQL databases are built to tolerate and recover from such failure making them highly resilient.

## 6. CONCLUSIONS

If database schema is large and complex, it is not appropriate to find attributes, entities and creation of desired query form etc. This leads to dynamic query form system. This generates the query form according to user's desire at runtime. The system provides a solution for the query interface in large and complex database. Here F-measure is used to estimate the goodness of query form. F-measure is a typical metric to evaluate query result. The metric is appropriate for query form because query forms are designed to help users query the database. The goodness of query form is determined by the query result generated from query form. Based on this, rank and recommend the query form components so that users can refine the query form easily. Here efficiency is important because dynamic query form is an online system where user often expects quick response. Also used a NoSQL database system that is flexible to handle huge amount of data. As a future work, plan to develop multiple method to capture user's interest for queries beside click feedback can be developed and also relevance score between the keywords and the query form can be incorporated into the ranking of form components at each step. Converting relational database to NoSQL if this application is connected to another application having relational database can also be considered as a future work.

## REFERENCES

1. Liang Tang, Tao Li, Yexi Jiang, Zhiyuan Chen, "Dynamic Query Forms for Database Queries," IEEE Transactions on Knowledge and Data Engineering, 19 April 2013.
2. DBPedia. http://DBPedia.org.
3. Freebase. http://www.freebase.com.
4. 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.
5. B. Liu and H. V. Jagadish. Using trees to depict a forest. *PVLDB*, 2(1):133–144, 2009.
6. Mongodb. http://www.mongodb.org
7. G. Salton and M. McGill. *Introduction to Modern Informatio Retrieval*. McGraw-Hill, 1984.