Kalyani Matey *et al*, International Journal of Computer Science and Mobile Applications,
Vol.3 Issue. 1, January- 2015, pg. 24-32 **ISSN: 2321-8363**

Survey on Data Mining Using Dynamic Query Forms Technique

Kalyani Matey[1], Ms. Archana Nikose[2]
[1]M.Tech Semester-IV student, Department of CSE, RTM Nagpur University, Priyadarshini. Bhagwati College of Engineering, Nagpur (M.S), India, [1] matey.kalyani@gmail.com
[2]Assitant Professor, Department of CSE, Priyadarshini Bhagwati College of Engineering, Nagpur (M.S.), India
[2] nikose.archu@gmail.com

Abstract

Different kinds of databases such as modern scientific database, web database. These are maintaining large and heterogeneous data. This kind of database contains sometimes hundreds or thousands types of relations and attributes. Traditional query forms which are already defined are not able to satisfy various types of ad-hoc queries with the help of user on those databases. So that user's requirements are not fulfil successfully. Thus, a new query form generated which overcome the problems of previously generated query forms and the name to be given as Dynamic Query Forms. Dynamic query forms are the new database query interface which able to generate query forms dynamically. The primary function of dynamic query forms are- 1) Capture the user preferences and rank those query forms components. 2) Assist user to make their decision. The generation of query form is intentionally guided by user and iterative process. The system automatically generates ranking lists of form components and the user then adds the desired form components into the query form. Based on the captured user preference ranking of form components can be done. User can also fill the query form and submit queries to see the query result. A query form could be dynamically refined till the user satisfies with the query results. The expected F-measure for the query form goodness measuring is utilized in this approach. In DFQ, for estimating the goodness of a query form, a probabilistic model is developed. The effectiveness and the efficiency of DQF are proved after experimental evaluation and user study.

Keywords: Query Forms, User Interaction, Query Form Generation, DQF, databases, query forms, F-measure

1. Introduction

The most widely used user interface for querying database is 'Query form'. The developers and DBA's in various information management systems have designed and predefined the traditional query forms. The modern databases have become very large and composite with the rapid development of web informatics and scientific databases. In natural sciences, such as diseases and genomics, there are over hundreds of entities for biological and chemical data resources in the databases [1], [2]. Many web databases, like DBPedia and Freebase, usually have over thousands of structured web entities [3], [4]. Therefore, it is hard to structure a set of static query forms to satisfy different ad-hoc database queries on those composite databases. Many current database management and development tools, like SAP and MS Access, lets the user develop customized queries on databases, by providing several mechanisms. Conversely, the development of customized queries totally based upon manual editing's of user [5]. Those hundreds and thousands of data attribute will confuse the user if they are not familiar with database schema in advance. The non-technical users make usage of relational database a challenging task. Therefore, in recent years many researches were focused on database interfaces to assist users to query the relational databases without using SQL. This paper proposes a Dynamic Query Form system (DQF), is a query interface capable of dynamically producing query forms for the users. Unlike traditional document retrieval, before identifying the final candidate, the users in database retrieval are mostly willing to execute several rounds of action [6].

The important feature of DQF is:

1) During the user interactions, capture the user interest.

2) Iteratively adapt the query forms.

Each of this iteration is made up of two types of user interactions. They are:

1) Query Execution, and

2) Query Form Enrichment.

The work flow of the DQF starts with a basic query form containing very few primary attributes of the database. It is then enriched iteratively by the means of interactions between the user and the system, unless the user is pleased with the query outcome. Mainly the study of query form components and the dynamic production of query forms are done in this paper. This paper proposes a dynamic query form system, generating the query forms in accordance with the run time desire of the user. The system presents an elucidation for the query interface in large and composite databases. The technique applies F-measure for approximation of the goodness of a query form [7]. Using this, we can rank and recommend the probable query form components, so that the users can filter the query form effortlessly. By using the proposed metric, to approximate the goodness of the protrusion and assortment of form components, have developed an efficient algorithms. As the DQF is an online tool and users usually expect quick response, Efficiency is very important. Here, this paper proposes a dynamic query form generation technique which will assist users to dynamically generate query forms. The key idea is based on user preferences, to use probabilistic model to rank form components. By using both, runtime feedback and historical queries, system captures the user preferences.

2. Literature Survey

“Automating the Design and Construction of Query Forms [6]” M. Jayapandian proposed the 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 that paper, they 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, they introduced an algorithm to generate a set of forms automatically given the expected query workload. They presented a study of system performance using a real query trace, as well as queries from a standard XML benchmark. They felt 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 sizably reduced learning curve. While their 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. But they not applied the techniques to exploit such complementarities.

“USHER: Improving Data Quality with Dynamic Forms [8]” K.Chen proposed the system which shows the probabilistic approaches can be used to design intelligent data entry forms that promote high data quality. USHER leverages data-driven insights to automate multiple steps in the data entry pipeline. Before entry, we find an ordering of form fields that promotes rapid information capture, driven by a greedy information gain principle. During entry, we use the same principle to dynamically adapt the form based on entered values. After entry, we automatically identify possibly erroneous inputs, guided by contextualized error likelihood, and re-ask those questions to verify their correctness. Our empirical evaluations demonstrate the data quality benefits of each of these components: question ordering allows better prediction accuracy and the re-asking model identifies erroneous responses effectively. There are a variety of ways in which this work can be extended. Our intention to answer this problem in greater depth with user studies and field deployments of our system. On the modelling side, our current probabilistic approach assumes that every question is discrete and takes on a series of unrelated values. Relaxing these assumptions would make for a richer and potentially more accurate predictive model for many domains. Additionally, we want to consider models that reflect temporal changes in the underlying data. Our present error model makes strong assumptions both about how errors are distributed, and what errors look like. On that front, an interesting line of future work would be to learn a model of data entry errors and adapt our system to catch them. Finally, we plan to measure the practical impact of our system, by piloting USHER with our field partners, the United Nations Development Program's Millennium Villages Project in Uganda, and a community health care program in Tanzania. These organizations data quality concerns were the original motivation for this work, and thus serve as an important litmus test for our system.

"Static Checking of Dynamically Generated Queries in Database Applications" [13] Carl Gould presented a sound, static analysis technique for verifying the correctness of dynamically generated SQL query strings in database applications. Our technique is based on applications of a string analysis for Java programs and a variant of the context-free language reach ability algorithm. We have implemented our technique and have performed extensive testing of our tool on realistic programs. The tool has detected known and unknown errors in these programs, and it is rather precise with low false-positive rates on our test programs. For future work, there are a few interesting directions. To increase the usability of our tool for debugging, it would be interesting to map an error path that we find in the automaton to the original Java source. One possible approach is to carry line numbers of the flow-graph nodes in the source code to the automaton, so that a path in the automaton can be associated with a set of source lines in the original Java program. A related problem is to check the correct uses of the query results in the source program.

"Combining Keyword Search and Form for Ad-Hoc Querying of Databases [7]" E. Chu investigates the approach of using keyword search to lead users to forms for ad hoc querying of databases. We consider a number of issues that arise in the implementation for this approach: designing and generating forms in a systematic fashion, handling keyword queries that are a mix of data terms and schema terms, filtering out forms that would produce no results with respect to a user's query, and ranking and displaying forms in a way that help users find useful forms more quickly. Our experience suggests several conclusions. One is that a query rewrite by mapping data values to schema values during keyword search, coupled with filtering forms that would lead to empty results, is an attractive approach. Another conclusion is that simply displaying the returned forms as a flat list may not be desirable some way of grouping and presenting similar forms to users is necessary. Substantial scope for further work remains. In particular, developing automated techniques for generating better form descriptions, especially in the presence of grouping of forms, appears to be a challenging and important problem. Also, exploring the tradeoffs between keyword search directly over the relational database and our approach is an intriguing topic. Certainly forms can express queries not expressible in basic keyword search; however, it is possible to ameliorate this somewhat by augmenting basic keyword search with some structured constructs. Discovering which approach is most useful to users is an open question. Even on queries that are expressible by both approaches, there is a basic philosophical difference: our approach returns a ranked list of relevant queries (expressed in what are hopefully user-friendly forms), whereas the traditional keyword search approach returns ranked lists of relevant answers. Determining under which circumstances each is most appropriate is an important task.

3. Related Work

The non-expert users make use of the relational database is a challenging topic. Many works focus on database interfaces which assist users to query the relational database without SQL. QBE (Query-By-Example) and Query Form are two most widely used database querying interfaces .The query forms have been utilized in most real-world business or scientific information systems. Related Work

1) Query by Example (QBE)

It is a database query language for relational databases. It is the first query language used to create visual tables where the user can enter commands. A lot of graphical front-ends for databases use the ideas from query by example today. QBE limited only for the purpose of retrieving data, Query by example was later extended to allow other operations, such as insert, select, deletes, forms, updates, create tables.

The motivation behind QBE is that a parser can convert the user's actions into statements expressed in a database manipulation language, such as SQL. A front-end can minimize the burden on the user to remember the importance of SQL, and it is easier and more productive for end-users (and even programmers) to select tables and columns by selecting them rather than typing in their names. The problem of query by example is relational completeness and ordering problem.

EXAMPLE FORM

.....Name: Bob

..Address:

.....City:

....State: TX

..Zip code:

Resulting SQL: SELECT * FROM Contacts WHERE Name='Bob' AND State='TX'

2) Automated Ranking of Database Query

Automated ranking of the results of a query is a popular *aspect of the query model in Information Retrieval (IR) that* we have grown to depend on. The database systems support, a selection query on a SQL database returns all tuples that satisfy the conditions in the query. The following two scenarios are not handled by a SQL system:

a. *Empty answers*: When the query is selective, the answer may be empty. In that case, it is desirable to have the option of requesting a ranked list of approximately matching tuples without having to specify the ranking function that captures “proximity” to the query. An FBI agent or an analyst involved in data exploration will find such functionality appealing.

b. *Many answers*: When the query is not selective, many tuples may be in the answer. In that case, it will be desirable to have the option of ordering the matches automatically that ranks more “globally important” answer tuples higher and returning only the best matches. A customer browsing a product catalogue will find such functionality attractive. The problem of automated ranking of database query result is the ranking functions might fail to perform this is because many tuples may tie for the same similarity score. It can be also arise for empty answer problem also. The query forms are generated based on the selected attributes. Then apply clustering algorithm on historical queries to find the representative queries. The forms are then generated based on those representative queries. The problem of the approaches is that, if the database schema is complex and large, user queries could be quite diverse. Even if we generate lots of query forms, there are still user queries that cannot be satisfied by any one of query forms. The problem is that, when we generate a large number of query forms, to let users find an appropriate and desired query form would be challenging. The solution that combines keyword search with query form generation is proposed in. It automatically generates a lot of query forms in advance. The user gives some keywords to find relevant query forms from a large number of regenerated query forms. It works well in the databases which have rich textual information in data schemas. However, it is not appropriate when the user does not have concrete keywords to describe the queries especially for the numeric attributes.

3) Querying using Instant-Response Interfaces

The problem of searching for information in large databases has always been a daunting task. In current database systems, the user has to overcome a multitude of challenges. The major challenge is that of schema complexity: large organizations may have employee records in varied schema, typical to each department. The user may not be aware of the exact values of the selection and provide only a partial or misspelled attribute value. The user to issue queries that are meaningful in terms of result size a query listing all employees in the organization would not be helpful to the user, and could be expensive for the system to compute. Lastly, we do not expect the user to be proficient in any complex database query language to access the database. The problem of assisted querying using instant response interface is the user’s information need is explicit .Attempt to apply efficient index structures for basic keyword querying.

4) Forms-based Database Query Interface

Forms-based query interfaces are widely used to access databases today. The forms-based interface is often a key step in the deployment of a database. The form is an interface is capable of expressing only a very limited range of queries. The set of forms as a whole must be able to express all possible queries that any user may have. For creating an interface that approaches this ideal is surprisingly hard. To maximize the ability of a forms-based interface to support queries a user may ask when bounding both the number of forms and the complexity of any form. A database schema and content we present an automated technique to generate a good set of forms that meet the above criteria. A careful analysis of real or expected query workloads are useful in designing the interface the query sets are often unavailable or hard to obtain prior to the database even being deployed. The generating a good set of forms just using the database itself is a challenging yet important problem. The problem of automated creation of forms based database querying interfaces is to use history log and use association mining for Related Entities.

5) Form Customization

A form-based query interface is usually the preferred means to provide an unsophisticated user access to a database. The interface is very easy to use, requiring no training, but it also requires little or no knowledge of how the data is structured in the database. A typical form is static and can express only a very limited set of queries. Query specification is limited by the expertise and vision of the interface developer at the time the form was created. The modifications are themselves specified through filling forms to create an expression in an underlying form manipulation expression language we define. To modify forms is not much greater than form filling. A form editor is used to implements form manipulation language. A query generator that modifies the form's original query based on a user's changes. The tool provides an effective means for specifying complex queries. The problem of form customization is limiting the usefulness of forms: their restrictive nature. Existing database clients and tools make great efforts to help developers design and generate the query forms, such as Easy Query [8], Cold Fusion [6], SAP, Microsoft Access and so on. The existing databases provide visual interfaces for developers to create or customize query forms. This tool is used by the professional developers who are familiar with their databases, not for end-users a system which allows end-users to customize the existing query form at run time. An end-user may not be familiar with the database. If the database is very large and complex, it is difficult for them to find appropriate database entities and attributes and to create desired query forms.

4. Comparison

Title	Features	Problems
Query-by example	Provides a simple interface for a user to enter queries.	1)Relational completeness 2)Ordering problem
Automated Ranking of Database Query	To build a generic automated ranking infrastructure for SQL databases.	1)Ranking function might fail to perform

Instant Response Interfaces	Interface developed to assist the user to type the database queries	1) The users information need is explicit 2) Attempt to apply efficient index structures for basic keyword querying
Forms-based Database Query Interface	Automatic approaches to generate the database query forms without user participation	1) Not appropriate when the user does not have concrete keywords to describe the queries.
Form Customization	A system which allows end-users to customize the existing query format run time	1)Database schema is very large so it is difficult to create desired query forms

5. Proposed Scheme

For a declarative query, to design a form, we must first 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 analyse it and identify its constraints and. 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. 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. 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. Figure 1 shows the work-flow of Dynamic Query Form. 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. After submitting query form by user Dynamic Query Form System perform different task iteratively. First It create the query in accordance with the user query form generation after that system will execute the query and display the result. If user satisfied with the result then it end the process otherwise user select the interested form components from the displaying result and as per selected components by user system enrich the form and process again going to be work from initial fill query form stage in iteratively manner.

Modules :

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

1. Query Form Generation
2. Query Creation
3. Query Execution
4. Query Form Enrichment

Our aim in the proposed system is to 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. We propose a dynamic query form generation approach which helps users dynamically generate query forms. 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.

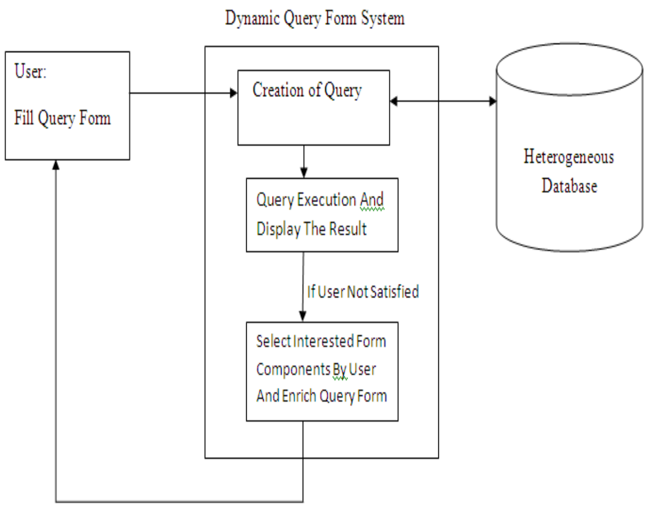
Dynamic Query Form System
User:
Fill Query Form
Creation of Query
Heterogeneous Database
Query Execution And Display The Result
If User Not Satisfied
Select Interested Form Components By User And Enrich Query Form

Figure 1 : Architecture Of Dynamic Query Form Procedure

1.1 Query form interface

This consists of two parts as explained in [8], namely Query forms and Query results.

1.1.1 Query forms

We have formally defined the query form in this section. Each of the queries does belong to a SQL query template. As ad-hoc join is not the part of the query form and is basically invisible to users, it is not handled in our approach by dynamic query forms. There are very limited number of options for users, if concerned to 'Aggregation' and 'Order By' in SQL. For instance, 'Order by' could only be 'decreasing order' and 'increasing order' and 'Aggregation' could only be AVG, MAX, and MIN and so on. To include these options, our approach can be easily extended by implementing them as dropdown boxes inside the user interface of the query forms [9].

1.1.2 Query Results

To choose whether a query form is required or not, there is not much time users have to waste on to go to every data instance in the query result. To add to this problem, a huge amount of data instances are generated as the output by many databases. To avoid this 'Many-Answer' problem [10], to show a high level observation of the query results earliest, we only yield a compressed result table. A cluster of actual instances are represented by each instance in the compressed table. Then, to view the comprehensive data instances, the user can briefly click through interested clusters. The compressed high level aspect of query results is proposed in [11]. Many one pass algorithms have been developed for generating the compressed view powerfully [12], [13]. Because of the efficiency issue, we have chosen the incremental data clustering framework [12], in this implementation. Different compressed views for user are proposed by different data clustering methods. Also, different clustering methods are proposed for different data types. We have implemented clustering just to offer a better aspect of the query results for the user. Any other clustering algorithm can be used if necessary.

1.2 Ranking Metrics

Query forms are designed such that it would return the user's desired result. To evaluate the quality of the query result, two traditional measures are available: Recall and Precision [7]. We can use the expected recall and expected precision to evaluate the query forms expected performance, because the query forms are capable of producing different queries for different inputs and different precisions and recalls can be achieved by different queries that outputs different query results. Naturally, current user's interest in query result is expected proportion of expected precision. The interest of user in approximated using the user's click through on display of query results by the query form.

1.3 Estimation of Ranking Score

The ranking score estimation phase consists of only two phases. The two phases are: Ranking projection form components, and second is, ranking selection form components.

1.3.1 Ranking Projection Form Components:

The DQF has provided a two level ranked list for the purpose of projection of components. The first level is ranked entities. This level describes how to rank attributes of each entity and that too locally. The second level is the ranked list of attributes in the same entity. This level describes the ranked lists of attributes in the same entity. Instinctively, the entity should be ranked higher in list, if those entities have more number of high scoring attributes.

1.3.2 Ranking Selection Form Components:

The selection of attributes will be absolutely meaningless, if the selection attributes are not relevant to the currently projected entities. Therefore, for creating the selection components, first, the system should try and find out the relevant attributes. This section is further divided in three phases. The three phases are:

a)Relevant attribute selection:

In this, the related or similar attributes are selected. This attributes are then grouped.

b)Ranking selection components*:*

In this step, the components groups are acquired, taken from the first step. These components are then ranked according to their usage.

c) Diversity of Selection components:

Two selection components may have a number of redundancies or overlays. Therefore, a high diversity should be provided in order to select the recommended components. Diversity is the recent major topic in recommendation system and web search engines as proposed in [14] and [15]. Though, simultaneously maximizing the precision and the diversity is an NP-Hard problem [14]. In an interactive system, it cannot be implemented efficiently. In this DQF system, it is observed that the same attribute may construct the most redundant components. Hence, for each attribute, only the best selection components are recommended.

5. Conclusion

In this project 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.

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 for the future work, our aim is to how this approach can be extended to non-relational data. 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 combining keyword search and forms for ad-hoc querying of databases can be incorporated into the ranking of form components at each step.

References

[1] J. C. Kissinger, B. P. Brunk, J. Crabtree, M. J. Fraunholz, B. Gajria, A. J. Milgram, D. S. Pearson, J. Schug, A. Bahl, S. J. Diskin, H. Ginsburg, G. R. Grant, D. Gupta, P. Labo, L. Li, M. D. Mailman, S. K.McWeeney, P. Whetzel, C. J. Stoeckert, J. . D. S. Roos, "The plasmodium genome database: Designing and mining a eukaryotic genomics resource", Nature, 2002.
[2] S. Cohen-Boulakia, O. Biton, S. Davidson, and C. Froidevaux, "Bioguidesrs: querying multiple sources with a user-centric perspective", Bioinformatics, 2007.
[3] M. Jayapandian and H. V. Jagadish, "Automated creation of a forms-based database query interface", In Proceedings of the VLDB Endowment, 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] L. Tang, T. Li, Y. Jiang, Z. Chen, "Dynamic Query Forms for Database Queries", IEEE transactions on knowledge and data engineering, 2013.
[7] 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, 2009.
[8] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum, "Probabilistic information retrieval approach for ranking of database query results", ACM Trans. Database Syst. (TODS), 2006.
[9] B. Liu and H. V. Jagadish, "Using trees to depict a forest", PVLDB, 2009.
[10] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams", In Proceedings of VLDB, 2003.
[11] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases", In Proceedings of SIGMOD, 1996.
[12] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong, "Diversifying search results", In Proceedings of WSDM, 2009.
[13] 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.
[14] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Static Checking of Dynamically Generated Queries in Database Applications, Rhode Island, USA, September 2009.
[15] G. Wolf, H. Khatri, B. Chokshi, J. Fan, Y. Chen, and S. Kambhampati. Query processing over incomplete autonomous databases. In *Proceedings of VLDB*, pages 651–662, 2007.