Using Queries to Associate Metadata with Data

Divesh Srivastava
AT&T Labs–Research
divesh@research.att.com

Yannis Velegrakis
University of Trento
velgias@dit.unitn.it

Abstract

As relational databases proliferate and become increasingly complex, both in their internal structure and in their interactions with other databases and applications, there is a growing need to associate a variety of metadata with the underlying data. Even though the need has been apparent, a simple, elegant approach to uniformly model and query both data and metadata has been elusive. In this paper, we argue that the relational model augmented with queries as data values is a natural way to uniformly model data, arbitrary metadata and their association.

1. Motivation

Databases are becoming increasingly complex, both in their internal structure (e.g., thousands of tables) and in their interactions with other databases and applications (e.g., mediators and workflows). In successfully understanding, maintaining, querying, integrating and evolving these databases, metadata plays an important role. Metadata is data about data, a secondary piece of information that is separate in some way from the primary piece of information to which it refers. Metadata examples include schema, integrity constraints, comments about the data, ontologies, quality parameters, annotations, provenance information, security policies, or statistical data characteristics.

Metadata is used in many different fields. In corporate environments, databases deployed at the core of important business operations may contain erroneous, inaccurate, out-of-date, or incomplete data with a significant impact on query results. In such environments, data is usually tagged with quality parameters to communicate suitability, accuracy, freshness or redundancy, and schema structures can be annotated with textual description to communicate their semantics. In scientific domains where data may be collected from various sources, cleansed, integrated and processed to produce new forms of data enhanced with new analysis results [8], provenance information can be provided as metadata in the form of annotations [1] or special structures stored in the database [12], allowing users to apply their own judgment to assess credibility of query results. In heterogeneous environments metadata can be used to resolve semantic conflicts [9].

Over the years, numerous proposals have been made by researchers for augmenting the data model and the query capabilities of a database in order to facilitate metadata management (e.g., [1, 6, 2]). These proposals are about metadata of different kinds (semantics), and forms (atomic values or complex structures), but each such proposal is tailored to specific kinds of metadata, and is not directly applicable to other kinds, at least not without some major modifications. Past attempts at building generic metadata stores (e.g., [5]) have employed highly expressive modeling tools to explicitly represent the various artifacts.

In this work, we initiate a study that aims to accommodate in one framework the different kinds of metadata, the different structures, the different ways that metadata is associated with data and the different ways in which it is used in queries. We argue that in order to achieve this we need a framework that is simple and abstracted from the specifics of each kind of metadata. Having observed that the operations one needs to perform on metadata are similar to those people do with data, we propose the use of standard data management techniques for metadata, so that both data and metadata can be managed in one single framework. We advocate that the relational model is adequate for such a purpose. Metadata with complex structures can easily be modeled through relations and attributes. These relations have no special semantics, thus, the same piece of information can be viewed either as data or as metadata. It can also be queried using a relational query language, even independently of its association to data.

Instead of using explicit values to form associations between data and metadata, we propose the use of queries as data values in the relational tables for this purpose. Using queries as data values is not entirely new. Relational DBMSs already store in the catalog tables the definition queries of the views that have been defined in the database. In this case, however, queries are considered schema information and despite the fact that they can be queried using SQL, they are not considered part of the instance data.

Our proposal raises such metadata to the level of data, and provides a unified mechanism for modeling and querying across data and metadata. Apart from the system catalog tables, queries as values have also been proposed in the INGRES system [11], and later similar functionality has been adopted in Oracle [3]. Queries as values have also been studied in the context of relational algebra [7]. Here we study how this idea can be used in the service of metadata management.

2. Queries as Data Values

Example 2.1 *Consider a communications company database with the table* `Customers` *as shown in Figure 1. The contents of the table are generated by integrating data from a number of physically distributed sources. When a mistake is detected in the table, it is important to know its origin in order to correct it. To make this information available to the user, the data in the* `Customers` *table needs to be annotated with its provenance information, i.e., the origin database name, its IP address and the protocol used to access it. One way to achieve this is to alter the table* `Customers` *by adding three new columns for each of its attributes [1, 4]. Such a solution may affect the way existing applications use the table, may degrade performance, or may not even be possible to implement due to lack of authorization for such a change. An alternative solution is to to store the provenance information in a separate table (*`Provenance`*) as illustrated in Figure 1. Column* `Rf1` *can be used to specify the relationship between the specific tuple and the data it annotates. It may contain* `Name` *values assuming that* `Name` *is the key in* `Customers`*. For instance, tuple* $[\texttt{BCT}, \texttt{NJDB}, 147.52.7.8, \texttt{http}]$ *in* `Provenance` *would indicate that the data tuple for the* `BCT` *customer was obtained from the* `NJDB` *source. This modeling approach has two main drawbacks. First, it results in a lot of information repetition. Assume that it has been asserted that all the New Jersey customers originate from the same data source. To record that, a tuple like the one just mentioned will have to be repeated in table* `Provenance` *for every New Jersey customer. Furthermore, if a new New Jersey customer is inserted, the respective tuple will have to be created in* `Provenance` *for that customer. The second drawback is that this mechanism cannot be used to model the fact that a* `Provenance` *tuple may not refer to the whole* `Customers` *tuple but only to a subset of its attributes.*

What we propose in order to deal with these issues is to use queries as data values in the table columns in order to intensionally describe the data to which a metadata information tuple is associated. To find whether a particular data value is associated with a (metadata) tuple, one only needs to check if the data value is part of the relation described by the query expression.

Example 2.2 *In the example database of Figure 1, column* `Rf1` *of table* `Provenance` *contains queries instead of atomic values. The first tuple with query* $q1$ *in column* `Rf1` *intensionally describes that the provenance of all the customers with location 'NJ' is the* `NJDB` *data source. Furthermore, through the attributes of its* select *clause, it specifies that this is true only for attributes* `Name`, `Type` *and* `PhoneLine`*. It states nothing about attributes* `Loc` *and* `CircuitID`*. In a similar fashion, the second tuple specifies that data source '3State' is the origin of the* `Loc`, `PhoneLine` *and* `CircuitID` *values of all the business* `Customers`*.*

Having the provenance information as a separate table comes with the additional advantage that it can be queried independently of the data it is associated to. For instance, in the case where a data administrator needs to know the set of data sources that have contributed data, independent of what data that is, she can issue a query on the `Provenance` *table.*

With the proposed mechanism, new metadata can be easily added on top of other existing metadata by simply creating a new table and storing the appropriate queries as data values in one of its columns. This is an important feature since the same piece of information may be viewed as data by one application and as metadata by another. Furthermore, the mechanism allows metadata to refer to either a single attribute or set of attributes, and also to have complex structures, i.e., multiple attributes of different types.

To implement this functionality, we extend the relational model with a new user defined atomic type, referred to as q-type, which is used to store queries as values in relational tables. Such queries are referred to as q-values.

To be able to dynamically execute q-values, we assume the existence of a function $eval$ whose role is to evaluate a query expression that is provided to it as argument. However, in contrast to other approaches that use query expressions as data values [11, 7], we do *not* consider *eval* as part of the relational algebra. Such an extension would have required the use of a nested relational model, i.e., a non first normal form model, which may be more powerful but comes with a significantly higher query evaluation cost. Furthermore, it would have created results with unspecified schema, since the evaluation of q-values with different numbers and types of attributes in their select clause would have resulted in multiple non union-compatible relations.

For storage and retrieval purposes q-type column contents are viewed as atomic types. However, for comparison purposes, we would like to view them as relations. Since a q-value is nothing more than an intensional description of a virtual relation, the functionality needed is the one that allows one to check whether certain values exist in the relation described by a q-value. If they do, it is said that the q-value *references* these values. This relationship can be used to perform selections and joins that are based on q-values.

Customers

Name	Type	Loc	PhoneLine	CircuitID
AFLAC	bus	NJ	4078417332	245-6983
J. Lu	res	NY	2019394460	245-7363
H. Ford	res	NJ	2159537607	245-7564
AMEX	bus	NY	3178763540	343-5002
NJC	bus	NJ	9730918327	981-5002
BCT	bus	NJ	9734858504	273-6019
...	...	...	...	...

Provenance

Rf1	Source	IP	Protocol
q1	NJDB	147.52.7.8	http
q2	3State	148.62.1.11	ftp
...	...	...	...

Permissions

Rf2	Users
q11	Administrators
q12	Guests
...	...

q1: select Name,Type,PhoneLine
from Customers where Loc='NJ'

q2: select Loc,PhoneLine,CircuitID
from Customers where Type='business'

q11: select * from Provenance
where IP LIKE '147.%'

q12: select Name from Customers
where Loc='NY'

Figure 1. A database with metadata information stored in regular tables as data.

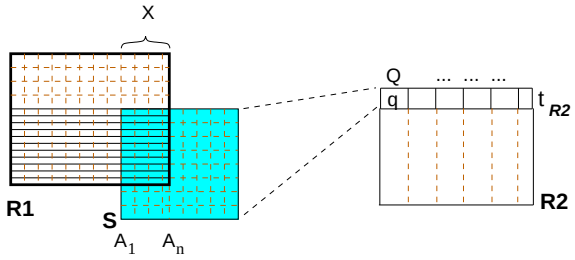

Figure 2. q-type reference

The mechanism is graphically explained in Figure 2. To express this relationship in queries, SQL is extended with a new operator $\doteq$. The semantics of that operator are illustrated through the following example.

Example 2.3 *Assume that a data administrator would like to know what data sources are related to customer 'AFLAC'. This translates to selecting from the* Provenance *table those tuples having a* q*-value in attribute* Rf1 *that references* Name *and that name is 'AFLAC'. This can be expressed as:*

select distinct p.Source from Provenance p
where p.Rf1[Name] $\doteq$ ['$AFLAC$'].

As a different example, assume that the data administrator has discovered that in the database in Figure 1 the customers with names starting with "A" violate the format policy and would like to know the source from where they originate. She knows that the provenance information is stored in the Provenance *table where attribute* Rf1 *specifies the association between the data and the metadata, so she issues the following query:*

select distinct p.Source from Customers c, Provenance p
where p.Rf1[Name] $\doteq$[c.Name] and c.Name *LIKE 'A%'*

What the query does is to select all the customers whose name starts with letter "A". For every such tuple c *it checks if there is a tuple* p *in* Provenance *with a* q*-value* q *in* p.Rf1 *such that relation* $eval(q)$ *has an attribute* Name *and there is at least one tuple in it with the value in column* Name *equal to the value of* c.Name*. If yes, then tuples* c *and* p *pair up. The answer of the above query on the instance of Figure 1 is the tuple* ['NJDB'].

Note that the case where all the q-values of a q-type column are of the form *"*select * from *R* where *A='const'*, with *const* being some constant value, that may be different each time, is a simulation of the value-based traditional join of the relational model.

3. Conclusion

There is a clear need to associate a variety of metadata with the underlying data in order to understand, maintain, query, integrate and evolve databases. We presented a simple, elegant approach to uniformly model and query data and arbitrary metadata, that is based on the use of queries as data values. The approach has been implemented in the MMS System [10].

References

[1] D. Bhagwat, L. Chiticariu, W. C. Tan, and G. Vijayvargiya. An Annotation Management System for Relational Databases. In *VLDB*, pages 900–911, 2004.

[2] S. Chawathe, S. Abiteboul, and J. Widom. Representing and Querying Changes in Semistructured Data. In *ICDE*, pages 4–19, 1998.

[3] D. Gawlick, D. Lenkov, A. Yalamanchi, and L. Chernobrod. Applications for Expression Data in Relational Database System. In *ICDE*, pages 609–620, 2004.

[4] F. Geerts, A. Kementsietsidis, and D. Milano. MONDRIAN: Annotating and querying databases through colors and blocks. In *ICDE*, 2006.

[5] M. Jarke, R. Gallersdorfer, M. A. Jeusfeld, and M. Staudt. ConceptBase - A Deductive Object Base for Meta Data Management. *J. Intell. Inf. Syst.*, 4(2):167–192, 1995.

[6] G. Mihaila, L. Raschid, and M-E. Vidal. Querying "Quality of Data" Metadata. In *In Third IEEE META-DATA Conference, Bethesda, Maryland*, apr 1999.

[7] F. Neven, J. V. Bussche, D. V. Gucht, and G. Vossen. Typed Query Languages for Databases Containing Queries. In *PODS*, pages 189–196, 1998.

[8] R. Rose and J. Frew. Lineage Retrieval for Scientific Data Processing: A Survey. *ACM Comp. Surveys*, 37(1):1–28.

[9] M. Siegel and S. E. Madnick. A Metadata Approach to Resolving Semantic Conflicts. In *VLDB*, pages 133–145, 1991.

[10] D. Srivastava and Y. Velegrakis. MMS: Using Queries As Data Values for Metadata Management (Demo Paper). In *ICDE*, 2007.

[11] M. Stonebraker, J. Anton, and E. N. Hanson. Extending a Database System with Procedures. *ACM TODS*, 12(3):350–376, 1987.

[12] Y. Velegrakis, R. J. Miller, and J. Mylopoulos. Representing and Querying Data Transformations. In *ICDE*, pages 81–92, 2005.