Extending relational database functionality with data inconsistency resolution support

Ilya Pevzner
Courant Institute, NYU, USA
pevzner@cs.nyu.edu

Arthur Goldberg
Courant Institute, NYU, USA
artg@cs.nyu.edu

Abstract

Resolving *inconsistent data* is a problem of critical practical importance. Inconsistent data arises whenever an attribute takes on multiple, inconsistent, values. This may occur when a particular entity is stored multiple times in one database, or in multiple databases that are combined. We've developed an architecture and methodology that extends relational databases with support for systematic data inconsistency resolution. We plan to employ a probabilistic data model and a machine learning approach.

1. Introduction

Combining the information from multiple databases is an issue of critical practical importance. We observe that even if the source databases are free from uncertainty and are consistent internally, the cross-database query may produce uncertain results. Standalone databases may also include inconsistent data since they are often populated from multiple sources. Consider a real world object described in multiple databases. If the object is not identified by the same unique id (UID) in the databases, we face the *object identity problem* where the system has to examine the object descriptions to determine if they apply to the same object. The problem is non-trivial when the object descriptions contain different but correct information in the common key fields. The process of determining if object descriptions apply to the same object is called matching and in general produces uncertain results. We call this "*matching uncertainty*". After object descriptions have been recognized to describe the same object, they may still contain different values for the same property. If this is the case, the actual 'correct' value of the property is uncertain. The different values for the same property of the object could arise for multiple reasons, for example, input or data errors (e.g. 'Pevzner' misspelled as 'Pezner') or natural variation (e.g. 'Arthur' or 'Art'). We call this *"inconsistency uncertainty"*. The goal of data inconsistency resolution is to "merge" such inconsistent data into a single "best" datum or a probability distribution.

We are seeking a general and practical solution to the data inconsistency resolution problem in the form of an extension to relational database functionality. The solution will include a data model suitable to represent both matching and inconsistency uncertainties, a technique to construct such representations from the inconsistent data (i.e. merging methodology), SQL extensions needed to support this model and an architecture that allows an implementation with acceptable performance on large data sets. Finally, the merging methodology needs to be easily adaptable to various application domains, and require minimal user intervention.

We are planning to develop a framework where various models of uncertainty and methodologies for data inconsistency resolution can be evaluated. This framework will help us develop a practical model of uncertainty and a library of data inconsistency resolution methodologies. We will test our system on several practical applications and evaluate it by how rapidly and accurately it merges inconsistent data and how easily the system can be applied to new domains, schemas and data.

2. Uncertainty models

Extending relational databases to handle uncertain information has been an active area of research. A survey of the area can be found, for example in [1]. Early attempts included experimenting with various semantics of *nulls* and extending relational data model with special operators supporting those semantics (see e.g. [2]). Further research focused on the following two general approaches. The first approach uses the possibility theory to define *fuzzy databases* (see [3] for an extended list of references), while the second approach uses probability theory to define *probabilistic databases* (see e.g. [1], [4], [5], [6], [7], [8]).

Although nulls have become an integral part of both theory and practice of relational databases, they lack the expressive power we need to represent the matching and inconsistency uncertainties. Fuzzy set theory and fuzzy databases are more expressive then nulls and have been fairly well studied but they are limited in that they do not allow one to express uncertainty quantitatively. The probabilistic approach is the least studied one, although there have been multiple techniques attempted. This approach is attractive because it offers greater expressive power and the underlying probability theory provides a good theoretical foundation.

There are many ways to extend the relational model with probability ([1], [4], [5], [6], [7], [8]). The choice of such probabilistic model is driven by the desired theoretical properties and practical requirements. Our current working model, as described in [9], is based on the Type-1 and Type-2 probabilistic relations described in [1].

Type-1 probabilistic relations are generalizations of the traditional relations obtained by adding a supplementary attribute *w(R, t)*, indicating the probability that a tuple *t* belongs to relation *R*. We use this type of probabilistic relations to represent matching uncertainty.

Type-2 probabilistic relations are generalizations of traditional relations where an attribute value can be either a constant or a *probabilistic set* with elements in the domain of the attribute. A probabilistic set *F* with elements from set *U* is defined as a pair (U, w_F) where w_F: $U \rightarrow [0,1]$ is a probability distribution satisfying $\sum_{u \in U} w_F(u) \leq 1$. We use this type of probabilistic relations to represent inconsistency uncertainty.

The final choice of the probabilistic model will be addressed in future work.

3. Merging Methodologies

Since we are extending a general-purpose system with data inconsistency resolution functionality, our merging methodology has to be universal (i.e. work for any relational schema and any valid instance of that schema) or at least configurable at the schema level and possibly tuneable at the database instance level. In the latter case, any user intervention required for instance level tuning has to be minimal.

Existing systems ([10], [11], [12]) either rely exclusively on specifying the exact merging rule for each potentially conflicting attribute at the schema-level or require providing such rules as part of the query. Either approach requires that the rules for merging are understood at the application development time. Since such rules are domain-specific and depend on the properties of the instance data, they are typically discovered by either interviewing a domain expert or mining (or "eyeballing") some sample of the data. Such rule discovery is not a trivial endeavor and is arguably one of the most difficult practical problems in database application development. Furthermore, since the properties of the instance data may change with time, the merging rules often need to be updated even after the application has been deployed.

In our approach, the merging methodology will be integrated within the extended database system. The application will simply rely on the system to analyze the properties of matching instances and present a consistent view of conflicting data. The whole process will require minimal user intervention. Since our system will need to adjust its behavior by analyzing the instance data, this approach belongs to the general area of machine learning (see e.g. [13]).

Experts in machine learning still consider developing a technique more of an "art" then a "science". A typical machine learning solution includes a combination of techniques, often with some problem-specific heuristics and optimizations (see e.g. [14]). For this reason, the development of a universal machine learning technique for data inconsistency resolution that will work for all relational schemas and all instances is probably not achievable. We are more likely to find that some machine learning techniques perform better for some types of data while other techniques perform better for other. Similarly, for some types of data no machine learning technique will work while for others, a trivial machine learning technique will perform sufficiently well.

We believe that a practically useful (and non-trivial) system can be engineered without devising a universal machine learning technique. Such a system would include an abstract machine-learning component and include a library of machine learning techniques, each useful for some types of data. The extended database system should provide facilities to manage this library and evaluate methodologies for a specific database instance.

In [9], we described a maximum entropy based data inconsistency resolution methodology. We are currently working on a Bayesian dependency-based methodology, which we believe will reduce the required user intervention.

We expect that with time, the library of data inconsistency resolution methodologies will be rich enough to provide an acceptable solution to most practical problems.

4. SQL Extensions

To provide access to the new database functionality described in the previous section, we need to extend SQL to (a) deal with the selected uncertainty model (b) manage the library of machine learning algorithms and (c) support matching and merging operations. In this section we summarize the extensions we described in [9] that address (a) and (c) in the framework of the uncertainty model based on Type-1 and Type-2 probabilistic relations introduced in [1]. The (b) requirement will be addressed in future work.

In the rest of this section we provide a review of the SQL extensions described in [9].

4.1 The MATCH predicate

The MATCH predicate can be used in the WHERE clause of the SELECT statement. It takes a pair of tuples of attributes that are tested for object identity and the reference to the corresponding instance of the object identification methodology. MATCH returns true if the model predicts that the tuples describe the same object. Any select statement using one or more MATCH predicates results in a probabilistic relation where the probability associated with each tuple is computed by the object identification methodology.

Example 1. Consider two data source relations S1 and S2 with the following data describing people's names, social security and telephone numbers.

S1

Id	Name	SSN
50	John	111-22-3333
60	Johnny	222-33-4444

S2

Id	Name	Phone
100	Jon	212-555-1212
200	Johnny	646-444-1212

The following query is seeking a list of people together with their telephone and social security numbers, obtained by matching their names.

```
SELECT S1.NAME, S1.SSN, S2.PHONE FROM S1, S2
WHERE MATCH('NAME_MATCHER', S1.NAME, S2.NAME)
```

The resulting Type-1 probabilistic relation is below:

Name	SSN	Phone	*w(R, t)*
John	111-22-3333	212-555-1212	.6
John	111-22-3333	646-444-1212	.8
Johnny	222-33-4444	212-555-1212	.5
Johnny	222-33-4444	646-444-1212	.9

The probabilities ***w(R, t)*** are computed by the matching algorithm denoted as 'NAME_MATCHER'.

4.1 The MERGE function

The MERGE function can be used in the select list of the SELECT statement. It takes a list of conflicting values for an attribute and a reference to the corresponding merging methodology. MERGE returns a table with the columns (v, w_F) where w_F is the probability that the object property takes the value v. The probability w_F is computed by the merging methodology. Any select statement with one or more MERGE functions results in a probabilistic relation with probabilistic sets associated with the attributes corresponding to the positions where the MERGE functions appear in the select list. Each table (v, w_F) returned by MERGE defines the corresponding probabilistic set F.

Example 2. Consider two source relations S1 and S2:

S1

SSN	Name
111-22-3333	John
222-33-4444	Johnny

S2

SSN	Name
111-22-3333	Jon
222-33-4444	John

The following query is seeking a list of people with their correct names.

```
SELECT S1.SSN,
MERGE('NAME_MERGER',(S1.NAME,S2.NAME))AS NAME
FROM S1, S2
WHERE S1.SSN=S2.SSN
```

The resulting Type-2 probabilistic relation is below:

SSN	Name
111-22-3333	{(John,.7), (Jon, 25)}
222-33-4444	{(Johnny, .2), (John, .6)}

The probabilities in the probabilistic sets are computed by the merging algorithm denoted as 'NAME_MERGER'.

4.1 The PROB function

The PROB function provides access to probabilities in the probabilistic relations. PROB can be used in the WHERE clause or in the SELECT list of the SELECT statement that produces a probabilistic relation. PROB either takes no parameters or accepts a name of the attribute from the select list. If no parameters are passed, PROB returns the probability associated with the current row. If an attribute name is specified, PROB references the probability associated with the specified attribute. In this form, PROB can only be used in the WHERE clause.

For instance, to limit the output of the query used in Example 1 we could write:

```
SELECT S1.NAME, S1.SSN, S2.PHONE
FROM S1, S2
WHERE MATCH('NAME_MATCHER', S1.NAME, S2.NAME)
AND PROB>.5
```

In this case PROB refers to the probability associated with the current row. The result is a Type-1 probabilistic relation is shown below.

Name	SSN	Phone	*w(R, t)*
John	111-22-3333	212-555-1212	.6
John	111-22-3333	646-444-1212	.8
Johnny	222-33-4444	646-444-1212	.9

To limit the output of the query in Example 2 we could write:

```
SELECT S1.SSN,
MERGE('NAME_MERGER',(S1.NAME,S2.NAME))AS NAME
FROM S1, S2
WHERE S1.SSN=S2.SSN AND PROB(NAME)>.2
```

In the above query PROB refers to the probability within the probabilistic set returned by the MERGE function. The result is a Type-2 probabilistic relation shown below.

SSN	Name
111-22-3333	{(John, .7), (Jon, .25)}
222-33-4444	{(John, .6)}

5. Extended Database Architecture

5.1 Query Processing

The architecture of the extended database includes the query processing module, application interfaces and development tools. The query-processing module includes integrated support for object identification and inconsistency resolution. The high-level diagram of the query-processing module is presented on Figure 1.

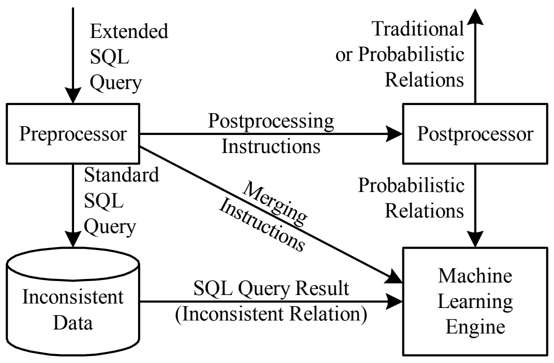

Figure 1. Extended SQL query processing diagram

On Figure 1, a user or application issues a query with SQL extensions; the preprocessor converts the extended SQL query into standard a SQL query, generates merging instructions (derived from MERGE function calls) for the machine learning engine and post-processing instructions (derived from PROB function calls) for the postprocessor and sends the standard SQL query to the database with inconsistent data. The machine-learning engine receives the result of the query (inconsistent relation), uses merging instructions to compute the resulting probabilistic relation, and sends the relation to the post-processor. Finally, the post-processor evaluates the post-processing instructions and produces the final result. The result can be a probabilistic relation or one or more classical relations representing some or all of the resulting possible worlds.

Further design details and optimization techniques will be addressed in future work.

5.2 Application Interfaces

To provide compatibility with applications written against traditional relational systems, our system will provide a standard SQL interface. When this interface is used, the merging instructions are generated from application-specific configuration parameters. The post-processing instructions are fixed, as the query results must always be presented as a single traditional relation. The postprocessor will generate such a relation for the most probable possible world. By design, any tools designed for traditional relational databases will work with the extended database using this interface.

To support the applications that use the SQL extensions to select and customize merging methodologies and specify conditions on probabilities, our system will provide the extended SQL interface. The queries issued using this interface always result in a single traditional relation, thus the application is isolated from the complexity of dealing with probabilistic relations or multiple possible worlds. If the conditions on probabilities specified in an extended SQL query make it impossible to provide a result as a traditional relation, the postprocessor will generate an error condition. Since the existing database tools work with traditional relations, it is still possible to use them with this interface, although modifications may be required to support the SQL extensions.

To support the advanced applications that require access to probabilistic relations or multiple possible worlds, the extended database will provide an advanced interface. This interface is different from the extended SQL interface, in that it may return probabilistic relations. For this interface, new database tools will need to be developed.

The three interfaces described above are shown on Figure 2.

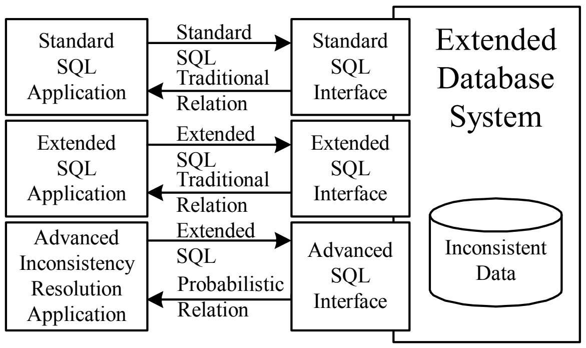

Figure 2. Application Interfaces

5.3 Development tools

Our extended database system will include the tools for development, integration, training, tuning and evaluation of machine learning methodologies for data inconsistency resolution. These tools would be similar to ModelMaker, a ChoiceMaker [15] development environment for matching methodologies.

6. Experimenting with real-world data

As discussed in the methodology section above, the performance of machine learning methodologies will largely depend on the specific data set. This makes it very important to find several realistic data sets to use for development, testing and tuning of the methodologies and validation of the complete system.

Finding a significant corpus of publicly available real-world data suitable for the task has been a non-trivial endeavor. Although there are numerous of public data sets available for machine learning research (see, e.g. [16]), those data sets seem to fall in one or more of the following unsuitable categories.

The first category includes data sets that do not contain any duplicates or inconsistencies. Such data sets are commonly used in research; however, databases with duplicates frequently occur in practice—witness the $100 million market for deduplication software. One approach to creating test data from such a set would be to apply some artificial perturbation to the data, but doing so in a way that preserves the real-world dependencies is tricky, and even if successful may not address the problems of the other two categories. The second category includes data sets that are too noisy or do not contain any dependencies between attributes. Such data sets often result from some artificial data perturbation techniques. Many production databases contain enough interdependencies between attributes to use some attributes to help correct other attributes. The third category includes data sets that do not have enough objects or attributes to reliably train a machine learning methodology. We believe that large business data sets where manual data cleaning is impractical and our approach is most relevant do not fall in this category.

We have recently identified a data set that seems to satisfy the requirements. It is a database with information about approximately 11 million medical research articles [17]. The database can be leased free of charge from The National Library of Medicine. We are currently experimenting with 2,391,822 affiliations that includes the name of the institution and optionally a department, street address and e-mail address for the main author of a paper. To avoid the object identity problem, we chose 523,140 annotations that contain an e-mail address. The merging problem for this data involves determining the correct department and street address for a given e-mail address. To simplify the parsing of the free-form address, we chose 182,892 records with US addresses. Of those records, 32,505 e-mail addresses have duplicate records for department and street address. This data set is promising because it contains sufficiently large number of records, with easily identifiable duplicates and potentially large number of conflicts resulting from merging data from different source databases.

7. Summary

The general nature and importance of data inconsistency resolution in practical application is well recognized. We've identified the following issues in extending the relational database functionality with a data inconsistency resolution function: 1) choice the uncertainty model 2) choice of merging methodology 3) design of SQL extensions 4) system architecture and 5) validating the system with real data. We've further described our initial approach to each of those issues and identified possible directions for future work.

8. References

[1] E Zimanyi and A. Pirotte. Imperfect Information in Relational Databases. In *Uncertainty Management in Information Systems*, A. Motro and P. Smets, Eds., Kulwer Publ., 1997.

[2] J. Biskup. A foundation of Codd's relational maybe-operations. *ACM TODS*, 8(4), December 1993.

[3] K. V. S. V. N. Raju and Arun K. Majumdar. Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems. *ACM TODS*, 13(2), June 1988.

[4] V.S. Lakshmanan, N. Leone, R. Ross and V.S. Subrahmanian. ProbView: A Flexible Probabilistic Database System. *ACM TODS*, 22(3), September 1997.

[5] L. Getoor, N. Friedman, D. Koller and A. Pfeifer. Learning probabilistic relational models. In *Relational Data Mining*, S. Dzeroski and N. Lavrac, Eds., Springer-Verlag, 2001.

[6] D. Dey and S. Sarkar. A Probabilistic Relational Model and Algebra. *ACM TODS*, 21(3), September 1996

[7] R. Cavallo and M. Pittarelli. The Theory of Probabilistic Databases. In *Proc. VLDB*, 1987.

[8] D. Barbara, H. Garcia-Molina and D. Porter. The Management of Probabilistic Data. *IEEE TKDE*, 4(5), October 1992.

[9] I. Pevzner and A. Goldberg MDQ -- A System for Resolving Data Inconsistencies in Multiple Relational Databases. In *Proc. BNCOD Ph.D. Summer School*, S. D. North, B. Eaglestone, Eds., Sheffield Unversity, 2002.

[10] H. Galhardas, D. Florescu, D. Shasha and E. Simon. An Extensible Framework for Data Cleaning. *Proc. ICDE*, 2000.

[11] P. Anokhin and A. Motro. Resolving Inconsistencies in the Multiplex Multidatabase System. *GMU Technical Report* 99_07, May 1999.

[12] F. Naumann and M. Haussler. Declarative Data Merging. In *Proc. ICDQ*, 2002.

[13] T. M. Mitchel. *Machine Learning*. McGraw Hill 1997.

[15] M. Buechi, A. Borthwick and A. Goldberg. The MEDD White paper. *ChoiceMaker Technologies*, 2001

[16] UCI Machine Learning Repository (http://www.ics.uci.edu/~mlearn/MLSummary.html)

[17] MEDLINE Database (http://www.nlm.nih.gov/databases/leased.html)