Trio: A System for Data, Uncertainty, and Lineage*

(Demonstration Description)

Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth,
Shubha Nabar, Tomoe Sugihara, and Jennifer Widom

Stanford University InfoLab
http://infolab.stanford.edu/trio

1 Introduction

In the *Trio* project at Stanford, we are building a new kind of database management system: one in which *data*, *uncertainty* of the data, and data *lineage* are all first-class citizens in an extended relational model and SQL-based query language. In an initial vision paper for the Trio project [5], we motivated the need for these three aspects to coexist in one system, and detailed numerous potential applications including scientific data management, data cleaning and integration, information extraction systems, and others. (Specific example application scenarios will be discussed in Sections 2 and 4.)

Since the inception of the project, we have:

1. Studied the space of representation schemes for uncertain data, and properties of various schemes [3, 4].

2. Proposed a new scheme called *ULDB*s. ULDBs extend the relational model with simple forms of uncertainty that, when combined with lineage, yield nice properties and strong expressiveness [1].

3. Proposed a SQL-based query language for ULDBs called *TriQL* (pronounced "treacle"). TriQL modifies the semantics of SQL to take uncertainty and lineage into account, and introduces new constructs to query uncertainty and lineage directly [2].

4. Implemented a first working prototype of our model and language by building on top of a conventional DBMS [2].

*This work was supported by the National Science Foundation under grants IIS-0324431 and IIS-0414762, by a grant from the Boeing Corporation, and by Stanford Graduate Fellowships from Cisco Systems and Sequoia Capital. Sugihara is on leave from NEC Corporation.

VLDB '06, September 12-15, 2006, Seoul, Korea.

We demonstrate our initial prototype, illustrating through sample applications how uncertainty and lineage are represented in ULDBs, how TriQL operates over ULDBs (from both the user and the system perspective), and in general how data, uncertainty, and lineage can work together to support interesting new functionality.

The next section presents the basic principles of the ULDB model, along with the TriQL query language. Section 3 provides an overview of the architecture and features of the initial Trio prototype. Section 4 describes the data sets, application scenarios, and user interfaces in the system demonstration.

2 Uncertainty-Lineage Databases

This section introduces the new data model and query language underlying the demonstrated system.

2.1 The ULDB Model

We present the ULDB model through examples using a highly simplified "crime-solver" application. Tables `Drives(person,car)` and `Saw(witness,car)` capture (possibly uncertain) driver information and crime-vehicle sightings, respectively.

ULDBs extend the standard SQL relational model with four new features:

1. *alternatives*, representing uncertainty about the contents of a tuple

2. *maybe* ('?') annotations, representing uncertainty about the presence of a tuple

3. numerical *confidence* values optionally attached to alternatives and '?'

4. *lineage*, connecting tuple alternatives to other alternatives from which they were derived

We discuss these four features in turn.

Alternatives

ULDB relations are comprised of *x-tuples* (and therefore are called *x-relations*). Each x-tuple consists of one or more *alternatives*, where each alternative is a regular tuple over the schema of the relation. For example, if a witness Amy saw either a Honda, Toyota, or Mazda, then in table `Saw` we have:

(witness, car)
(Amy,Honda) || (Amy,Toyota) || (Amy,Mazda)

This x-tuple logically yields three *possible instances* for table `Saw`, one for each alternative. In general, the possible instances of an x-relation R correspond to all combinations of alternatives for the x-tuples in R. For example, if a second tuple in `Saw` had four alternatives, then there would be 12 possible instances altogether.

In cases like the example above in which only some attributes are uncertain, users may abbreviate using *or-sets*:

witness	car
Amy	{Honda,Toyota,Mazda}

'?' (Maybe) Annotations

Suppose a second witness, Betty, thinks she saw a car but is not sure. However, if she saw a car, it was definitely an Acura. In ULDBs, uncertainty about the existence of a tuple (more generally of an x-tuple) is denoted by a '?' annotation on the x-tuple. Thus we have:

(witness, car)	
(Amy,Honda) || (Amy,Toyota) || (Amy,Mazda)	
(Betty,Acura)	**?**

The '?' on the second x-tuple indicates that this entire tuple may or may not be present (so we call it a *maybe x-tuple*). Now the possible instances of an x-relation include not only all combinations of alternatives, but also all combinations of inclusion/exclusion for the maybe x-tuples.

This `Saw` table has six possible instances: three choices for Amy's car times two choices for whether or not Betty saw an Acura. For example, one possible instance of `Saw` is the two tuples `(Amy,Honda),(Betty,Acura)`, while another instance is just `(Amy,Mazda)`.

Confidences

Numerical *confidence* values may be attached to the alternatives of an x-tuple. Suppose Amy decided she saw either a Honda or a Toyota with confidence 0.7 and 0.3 respectively, while Betty's confidence in seeing a vehicle is 0.6. Then we have:

(witness, car)	
(Amy,Honda):0.7 || (Amy,Toyota):0.3	
(Betty,Acura):0.6	**?**

In [1] we formalize an interpretation of these confidences in terms of probabilities, which is the current default in the Trio system (although other interpretations could be used instead). Thus, if Σ is the sum of confidences for the alternatives of an x-tuple, then we must have $\Sigma \leq 1$. Implicitly, '?' is given confidence $(1 - \Sigma)$ and denotes the probability that the tuple is not present. Each possible instance of a ULDB has a probability, and we show in [1] how probabilities of possible instances are derived in a consistent and complete fashion from confidence values.

An important special case of ULDBs is when every x-tuple has only one alternative with a confidence value that may be < 1. This case corresponds to the traditional notion of *probabilistic databases*, which are thus subsumed by ULDBs.

Lineage

In general, lineage augments data with information about how the data was derived. ULDBs support both *internal* lineage, connecting derived data to other data within the database, and *external* lineage, connecting data to outside sources [1, 5]. The initial Trio system focuses on internal lineage.

Lineage in ULDBs is recorded at the granularity of tuple alternatives: lineage connects a derived x-tuple alternative to the x-tuple alternatives from which it was derived. By default, lineage is generated by Trio automatically whenever a TriQL query is executed.

Consider the join of base tables `Saw` and `Drives` on attribute `car`, followed by a projection on `person` to produce a table `Suspects(person)`. We show some sample data for all three tables, including lineage for the derived data in `Suspects`. We use (i, j) to denote the j^{th} alternative of the x-tuple with ID i. For example, the lineage of Jimmy's presence in table `Suspects` is the second alternative of tuple 21 in table `Saw`, together with the second alternative of tuple 31 in table `Drives`.

ID	Saw (witness, car)
21	(Cathy,Honda) || (Cathy,Mazda)

ID	Drives (person, car)
31	(Jimmy,Toyota) || (Jimmy,Mazda)
32	(Billy,Honda)
33	(Hank,Honda)

ID	Suspects		
41	Jimmy	**?**	lineage(41,1) = { (21,2), (31,2) }
42	Billy	**?**	lineage(42,1) = { (21,1), (32,1) }
43	Hank	**?**	lineage(43,1) = { (21,1), (33,1) }

An interesting and important effect of lineage is that it imposes restrictions on the possible instances of a ULDB, effectively coordinating the uncertainty in derived data with the uncertainty in the data from which it was derived. Consider derived table `Suspects`. Even though there is a

'?' on each of its three tuples, not all combinations are possible. If Billy is present in `Suspects` then alternative 1 must be chosen for tuple 21, and therefore Hank must be present as well. Jimmy is present in `Suspects` only if alternative 2 is chosen for tuple 21, in which case neither Billy nor Hank can be present. The above ULDB has six possible instances, determined by the two choices for tuple 21 times the three choices (including '?') for tuple 31.

Another important effect of lineage is that it enables efficient on-demand computation of confidence values on query results. Although result confidences can be computed during query processing, in some cases it can be significantly more efficient to compute them as a separate step based on lineage. See [1] for details.

2.2 TriQL: The Trio Query Language

We now introduce *TriQL*, Trio's SQL-based query language, again through examples. Except for built-in functions and predicates for querying confidence values and lineage (discussed below), TriQL uses the same syntax as SQL.[1] For example, our join query producing `Suspects` is written in TriQL exactly as expected:

```
SELECT Drives.person INTO Suspects
FROM Saw, Drives
WHERE Saw.car = Drives.car
```

Logically, if we executed this query as regular SQL over each of the possible instances of `Saw` and `Drives`, we would obtain the possible instances of the result. Of course in reality we do not enumerate possible instances, and instead execute queries on x-relations directly. In [2] we give an operational semantics for TriQL queries that respects the formal possible-instances semantics. Although details are omitted, the reader can rest assured that the above query produces the `Suspects` result as shown in the previous section, including the lineage to base tables `Saw` and `Drives`.

Querying Confidences

TriQL provides a built-in function `conf()` for accessing confidence values. Suppose we want our `Suspects` query to only use sightings having confidence > 0.5 and driver information having confidence > 0.8. We write:

```
SELECT Drives.person INTO Suspects
FROM Saw, Drives
WHERE Saw.car = Drives.car
  AND conf(Saw)>0.5 AND conf(Drives)>0.8
```

Querying Lineage

For querying lineage, TriQL introduces a built-in predicate designed to be used as a join condition. If we include predicate `lineage`(R, S) in the `WHERE` clause of a TriQL query with x-relations R and S in its `FROM` clause, then we are constraining the joined R and S tuple alternatives to be connected by lineage. For example, suppose we want to find all witnesses contributing to Hank being a suspect. We can write:

```
SELECT Saw.witness
FROM Suspects, Saw
WHERE lineage(Suspects,Saw)
  AND Suspects.person = 'Hank'
```

In the `WHERE` clause, `lineage(Suspects,Saw)` evaluates to true for any pair of alternatives x_1 and x_2 from `Suspects` and `Saw` such that x_1's lineage includes x_2. We could instead write this query directly on the original tables by considering the query that produced `Suspects` (somewhat like manual *view unfolding*), but the `lineage()` predicate provides a more general construct that is insensitive to query history.

TriQL also provides a transitive version `lineage*()` of the lineage predicate. The Trio system currently implements `lineage*` by keeping track of the lineage structure in the database, and using the structure to translate `lineage*()` into a fixed set of `lineage()`-based joins.

As a final TriQL example incorporating both lineage and confidence, the following query finds persons who are suspected based on high-confidence driving of a Honda:

```
SELECT Drives.person
FROM Suspects, Drives
WHERE lineage(Suspects,Drives)
 AND Drives.car = 'Honda'
 AND conf(Drives)>0.8
```

3 The Trio System

We now briefly describe the architecture and features of the demonstrated Trio prototype. More details can be found in [2].

3.1 Software Architecture

The initial Trio system is built entirely on top of a conventional relational DBMS: ULDBs are represented in relational tables, and TriQL queries and commands are rewritten automatically into SQL commands evaluated against the representation.

The core system is implemented in Python and presents a simple API that extends the standard Python DB 2.0 API for database access (Python's analog of JDBC). The Trio API supports TriQL queries instead of SQL, query results are cursors enumerating x-tuple objects instead of regular tuples, and x-tuple objects provide programmatic access to their alternatives, including confidences and lineage. Using the Trio API, we built a generic command-line interactive client similar to that provided by most DBMS's, and a full-featured graphical user interface, *TrioExplorer*, discussed in Section 4.1 below.

[1] A follow-on version of TriQL includes additional new constructs for querying and manipulating uncertainty and confidence values; see `http://infolab.stanford.edu/trio`. The version presented here is the one demonstrated in the initial prototype.

Our prototype is built on the *Postgres* open-source DBMS, but we intentionally rely on very few Postgres-specific features. Porting to any other DBMS providing a DB 2.0 API would be straightforward.

3.2 Functionality

The user may create a ULDB x-relation R with any standard relational schema, with or without confidence values. This x-relation is represented as a conventional table storing tuples in R's schema augmented with *aid*'s (globally unique alternative identifiers) and *xid*'s (x-tuple identifiers that encode which alternatives belong to the same x-tuple). When a TriQL query creates a derived x-relation S, in addition to creating a table for S as described above, an additional table '`lin:S`' is created to store the lineage of data in S. Tuples in the lineage table contain three attributes: the *aid* for an alternative in S, and the *tablename* and *aid* for an alternative in S's lineage. Trio also maintains a catalog containing Trio-specific metadata, such as schema-level lineage information and which x-relations contain confidence values.

TriQL queries are rewritten automatically into SQL commands over the representation just described. Query processing proceeds in two phases. In the first phase, a single SQL query generates a table (call it T) containing all information needed to construct the query result. In the second phase, table T is post-processed to correctly group alternatives into the x-tuples in the result, and to construct the lineage table for the result. This second phase offers two options: It can produce the final result as a stored x-relation, or it can expose a cursor, in which case result x-tuples and their lineages are assembled as the application iterates through the cursor. Result confidence values are either computed *immediately* as part of the second phase, or *on-demand* when requested by a user or by a subsequent TriQL query with a `conf()` predicate.

4 Demonstration Highlights

Two data sets are used for the demonstration. One is a synthetic crime-solver database similar to the one used for examples in this description, but with additional tables and much more data. The second is a large product data set provided to us by Yahoo! Shopping, on which we have performed *deduplication*. Deduplication produces uncertainty (product matches with less than full confidence) and lineage (connecting merged product records to the originals).

4.1 User Interface

ULDBs and TriQL queries are demonstrated through *TrioExplorer*, a generic graphical user interface built on top of Trio's API. TrioExplorer lets the user connect to a ULDB, browse its schema and schema-level lineage structure, browse data in x-relations, ask TriQL queries and visualize their results, and navigate lineages of stored x-relations and query results.

4.2 Demonstration Walk-Through

The demonstration consists of three parts:

1. **Basic functionality.** We begin by introducing the ULDB model by visualizing the contents of base x-relations in the crime-solver application. A few simple TriQL queries are run to introduce their intuitive semantics over ULDBs. A selected TriQL query is then used to demonstrate result x-relations, how result data is connected to input data via lineage, and how lineage coordinates uncertainty between derived and base x-relations. Further queries demonstrate layers of derived x-relations and lineage, and the `lineage()` and `lineage*()` predicates in TriQL that enable lineage to be queried declaratively. Lastly, confidence values are introduced, and we show how queries can reference them.

2. **Second application.** Next we demonstrate our second application: deduplicated product data. This part illustrates scalability on a very large data set and shows a few additional queries over a different type of application.

3. **Under-the-hood.** Finally, for interested participants, we provide a glimpse into Trio's architecture and implementation. We return to our crime-solver database and show how its x-relations are mapped to the underlying relational representation, and how TriQL queries are translated into SQL commands over the representation. Time permitting we also demonstrate how Trio stores information on lineage structure and uses it to implement the `lineage*()` predicate, and how confidence computations are performed on-demand by traversing lineage.

References

[1] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *Proc. of Intl. Conference on Very Large Databases (VLDB)*, September 2006.

[2] O. Benjelloun, A. Das Sarma, C. Hayworth, and J. Widom. An introduction to ULDBs and the Trio system. *IEEE Data Engineering Bulletin, Special Issue on Probabilistic Databases*, 29(1):5–16, March 2006.

[3] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *Proc. of Intl. Conference on Data Engineering (ICDE)*, April 2006.

[4] A. Das Sarma, S.U. Nabar, and J. Widom. Representing uncertain data: Uniqueness, equivalence, minimization, and approximation. Technical report, Stanford InfoLab, December 2005. Available at http://dbpubs.stanford.edu/pub/2005-38.

[5] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. of Conference on Innovative Data Systems Research (CIDR)*, 2005.