Trio: A System for Integrated Management of Data, Accuracy, and Lineage

Jennifer Widom

Stanford University
widom@stanford.edu

Abstract

Trio is a new database system that manages not only *data*, but also the *accuracy* and *lineage* of the data. Inexact (uncertain, probabilistic, fuzzy, approximate, incomplete, and imprecise!) databases have been proposed in the past, and the lineage problem also has been studied. The goals of the Trio project are to combine and distill previous work into a simple and usable model, design a query language as an understandable extension to SQL, and most importantly build a working system—a system that augments conventional data management with both accuracy and lineage as an integral part of the data. This paper provides numerous motivating applications for Trio and lays out preliminary plans for the data model, query language, and prototype system.

1 Introduction

In traditional database management systems (DBMSs), every data item is either in the database or it isn't, the exact value of every data item is known, and how a data item came into existence is an auxiliary fact if recorded at all. Many database applications inherently require more flexibility and accountability in their data (see Section 2 for numerous examples): A data item may belong in the database with some amount of confidence, or its value may be approximate. Furthermore, how a data item came to exist—particularly if it was derived using other (possibly inexact) data at some point in time—can be an important fact, sometimes as important as the data item itself. Currently, applications with inexact data, or that rely on data derivation information (hereafter referred to as *lineage*), typically support these features outside the processing of a conventional DBMS.

In the new *Trio* project at Stanford we are developing a prototype database management system whose objective is to address the shortcomings of conventional DBMSs outlined above. Specifically, Trio provides management and querying of three interrelated components—*data*, *accuracy*, and *lineage*—in a simple, efficient, and fully integrated fashion. Salient features of Trio are:

1. Data values may be inexact—they may be approximate, uncertain, or incomplete. For example, an attribute value might be specified as a range known to contain the exact value (or some other type of approximation), or a record may include some confidence that it actually belongs in the database, or a relation may be estimated to miss some fraction of its records.

2. Queries operate over inexact data by returning answers that themselves may be inexact.

3. Lineage is an integral part of the data model: If a record r was derived by query Q over version V of data D at time T, this fact is associated with r. Lineage also captures updates, program-based derivations, bulk data loads, and import of data from outside sources.

4. Accuracy may be queried. For example, "*find all values whose approximation is within 1%*," or "*find all records with* $\geq 98\%$ *chance of belonging in the database*."

5. Lineage may be queried. For example, "*find all records whose derivation includes data from relation R,*" or "*determine whether a particular record r was derived from any data imported on 4/1/04.*"

6. Lineage and accuracy may be combined in queries. For example, "*find all records derived solely from high-confidence data.*"

7. Lineage can be used to enhance data modifications. For example, changes to a record r may invalidate other data derived using r, or may propagate in the style of materialized views. Of special interest is changes in the accuracy of existing data: when data D becomes more (or less) accurate, the effect on data derived from D may be computed and propagated automatically.

Providing all of these features for fully integrated management of data, accuracy, and lineage requires reconsidering many aspects of a DBMS. In the Trio project we plan to address at least the following areas:

- Theoretical foundations and data model
- Query language, including semantics, optimization, and execution strategies
- New access methods as needed for efficiency
- User and application interfaces that incorporate accuracy and lineage
- System architecture and implementation

In this paper we present a preliminary version of the *Trio Data Model* (*TDM*) that captures data, accuracy, and lineage (Section 4). We also discuss *TriQL* (pronounced "treacle"), a query language that extends SQL to incorporate the capabilities related to accuracy and lineage discussed above (Section 5). We cover implementation issues briefly (Section 6), but in this paper we do not delve into query optimization or execution, access methods, or interfaces.

Before turning to the data model, query language, and system, in the remainder of this section we discuss the scope of Trio and its objectives in the context of previous work. Section 2 presents a significant number of motivating applications, and Section 3 introduces a specific running example used thereafter in the paper.

1.1 What Trio is Not

Trio offers a platform for data management that extends a traditional DBMS in several ways, as discussed above. Possibilities are numerous for making data more flexible and accountable, so it is equally important to understand what capabilities are not an objective for Trio:

- *Trio is not a comprehensive temporal DBMS.* Time is an important component of data lineage: applications often need to keep track of when data items were derived, not just how. Furthermore, lineage capabilities may work together with versioning features in the Trio system (see Section 4.3). Nevertheless we do not plan to include rich fine-grained temporal data modeling or query constructs as part of Trio. Full temporal support could certainly be useful for some of our motivating applications (Section 2), but we prefer to separate that issue and focus initially on adding accuracy and lineage to conventional nontemporal data.

- *Trio is not a DBMS for semistructured data.* As with temporal data, semistructured data is clearly important in some of our motivating applications. However, we prefer to keep our base data model simple (i.e., relational) so we can focus on issues arising from the need to manage and query accuracy and lineage, not from lack of structure in the data itself.

- *Trio is not the last word in managing inexact data.* A tremendous amount of work has already been done in this area, as discussed in Section 1.2 below. Our goals are to distill from previous work a data model and query language incorporating accuracy that is simple enough to be adopted easily, yet expressive enough to capture a variety of applications; to integrate accuracy with data lineage; and to make the entire ensemble work in a running prototype.

- *Trio is not the last word in managing data lineage.* Similar to accuracy, data lineage has seen considerable previous work, discussed in Section 1.2. Here too our primary goals are usability, full integration with other aspects of the system, and a complete working prototype.

- *Trio is not a federated or distributed system.* Data values in Trio may be identifiers for external files or other outside sources of data, as in, e.g., [HN00]. However, we are not planning to build special treatment of these values into the first-version Trio query processor, even though it might be useful for some of our target applications. Trio still provides some useful features for outside data: When data is imported from possibly unreliable outside sources (e.g., in the scientific data application domain described Section 2.1), accuracy for that data can be set accordingly, and can be adjusted later via lineage. Furthermore, Trio data values that identify outside data sources could encode expected reliability or quality of the sources as an accuracy measure.

1.2 Contribution over Previous Work

There has been a significant amount of work in areas variously known as *uncertain*, *probabilistic*, *fuzzy*, *approximate*, *incomplete*, and *imprecise* data management. Examples of this work can be found in [BGMP92, BP82, BP93, CKP03, DS04, Fag02, Fuh90, IL84, KOR96, Lee92, LLRS97, LM04, LS90, Mot94, OO93, OW00, Sad91, SM01, YLW+95, Zim97], and even this list is woefully incomplete. One of our initial objectives is simply to sort out and understand the fundamental differences among these closely related lines of research. We will identify which ideas can and should be incorporated into Trio, bearing in mind our motivating applications (Section 2), and our goals of usability and rapid prototype deployment.

There has been a much smaller but nonetheless steady stream of recent work in data lineage (sometimes referred to as *provenance*), e.g., [BB99, BCTV04, BKT01, BKTT04, CW03, CWW00, FB01, FVWZ02, WM90, WS97]. In this area too we will incorporate previous ideas into our work, identifying the techniques that are useful for our motivating applications, feasible from a systems perspective, and consistent with our overall approach.

Considering previous work in accuracy and lineage, the main objectives of Trio are to:

1. Distill a simple and usable data model incorporating both accuracy and lineage

2. Introduce a query language that extends SQL to handle data, accuracy, and lineage in an integrated fashion, including queries and modifications

3. Deploy a working system that is sufficiently easy to adopt and efficient enough that it actually gets used

Two areas of recent work, *superimposed information* [DMB+01, MD99] and *annotation management* [BCTV04], propose models less specific than ours, but with the same overall goal of enhancing data with additional information that may be as important as the data itself.

Superimposed information targets any application that benefits from a second level of information layered over the base data. Superimposed information is added to existing information sources to "help organize, access, connect, and reuse information elements in those sources" [MD99]. Certainly lineage can

be considered a type of superimposed information, and accuracy could fall into that category too. However, work on superimposed information considers a completely separate layer, rather than integrating the additional information as in our approach.

Unlike superimposed information, annotations couple additional information directly with the base data. The annotation management system described in [BCTV04] considers the problem of propagating annotations through query operators in a subset of SQL, addressing in particular the issue that equivalent query formulations may not produce equivalent result annotations. Lineage (called provenance in [BCTV04]) plays a role in how annotations are propagated. Although not featured in [BCTV04], accuracy could be treated as a type of annotation, so the approach in [BCTV04] may be useful as we formulate our query semantics (Section 5.1).

2 Motivating Applications

This section motivates Trio with a wide variety of application domains. All of these applications can exploit some of Trio's unique features to best manage their data, and some applications can exploit all of the features. This suite of suitable application domains is certainly not exhaustive, but it is more than sufficient to motivate the need for a new system such as Trio that manages data, accuracy, and lineage in an integrated fashion.

Of course with such a plethora of motivating application domains it will be important to narrow down and prototype one or two specific applications in detail initially, in order to focus and validate our work. One candidate is presented in Section 3 and used for examples later in the paper.

2.1 Scientific Data Management

Consider an experimental scientist, e.g., a biologist, chemist, astronomer, earth scientist, etc. Scientists typically conduct many experiments that produce raw data—sometimes vast amounts of data—that must be saved and analyzed. Data-generating experiments may be performed in a laboratory, in the field, with pencil-and-paper analysis, via computer simulation, or some combination of these methods. Regardless of the method, in many cases the data values to be stored may be inexact, or may have a confidence value associated with them [KOR96].

From raw data values, aggregate or other combined or higher-level values may be derived and also stored in the database [FB01]. Subsequent experiments may generate new raw values which, in addition to creating new derived data, may alter the confidence or approximation of previous values, both raw and derived [GST+02]. In addition, scientists frequently incorporate or link data from outside sources into their own databases, both as raw values and to create new derived data. The accuracy and reliability of data obtained from these outside sources must be captured along with the data, and can also change over time.

This application domain can exploit all of Trio's features. Furthermore it motivates one of our main goals, which is keeping the system simple, efficient, and usable. Although scientists are clamoring for more suitable data management tools, they will not consider adopting a new software system if it is overly complex, slow, or has a high barrier to entry.

2.2 Sensor Data Management

Consider numerous sensors collecting data at regular intervals and transmitting their readings to a centralized system for processing. (Of course there are other, more distributed, approaches to sensor data management [Kum03], but centralized collection is a viable approach for a class of applications.) Often sensors may be unreliable: readings may be missed at some intervals, or transmitted values may be erroneous or imprecise. A platform for managing and querying inaccurate or incomplete data relieves the sensor processing system from implementing this functionality itself, outside of its data management system [Kum03].

Lineage also plays an important role in this setting. Typically, raw sensor readings are heavily aggregated and summarized, but it is useful to keep track of the original sources of the derived higher-level data—to facilitate detailed analysis but also to manage inaccuracy: For example, if a sensor is discovered to be particularly faulty, accuracy of its recent readings can be downgraded, with the effects propagated automatically to the accuracy of summary data computed using these readings. This capability is one of Trio's important features.

2.3 Data Deduplication

Deduplication is one common component of the *data cleaning* process, e.g., [CGG+03, GFS+01, HS98, Mon97, RH01, Sar00]. Some approaches to data cleaning exploit lineage, e.g., [GFS+01], and deduplication is especially important when integrating data from multiple sources, e.g, [SB02]. Deduplication algorithms identify and merge multiple data items that are likely to represent the same real-world entity. (In the closely related area of *record linkage*, e.g., [EEV02, JLM03], matching data items are identified but not merged.) In deduplication applications, some data items may include uncertainty (especially when merging data from multiple sources of varying quality), and the decision to merge two items may yield an uncertain composite item. Furthermore, maintaining the history of merges that create a composite item is necessary, both for propagating potential changes in confidence, as well as for unmerging merged items if warranted by additional information.

Like the scientific (Section 2.1) and sensor (Section 2.2) domains, deduplication is an application that can exploit all of Trio's novel features.

2.4 Profile Assembly

Profile assembly is closely related to deduplication. It consists of collecting, correlating, and combining possibly small pieces of information about individuals, with the goal of developing a comprehensive profile. Assembled profiles can be used for targeted advertising, assessing credit risk, intelligence-related purposes, and other applications in this genre. Very much like deduplication (Section 2.3), developing a profile over time may require querying, matching, merging, and possibly unmerging data items of varying accuracy and reliability—capabilities that clearly benefit from integrated management of accuracy and lineage.

2.5 Privacy Preservation

In the converse to profile assembly (Section 2.4), one class of techniques to preserve privacy of individual records in a database, while still enabling certain types of queries, is to perturb values, change exact values to approximate values, or generate and store statistics from a set of values, e.g., [AKSX02, SS98]. Once specific individual values are anonymized in this fashion, queries produce approximate results without revealing individual data items. Storage management and query processing over approximate or statistical values is a requirement in this setting (except when atomic values are simply transformed to other atomic values).

Lineage also is useful in this domain, because applications may need the capability to identify the original exact values from which a specific approximate value was derived—for users with special privileges, or to "unanonymize" certain data. For example, an application may wish to restore original (or less approximate) values when anonymity requirements are relaxed, or when additional data dilutes the database and relaxes anonymity requirements for certain individual items.

2.6 Approximate Query Processing

Sometimes query processing efficiency is of more importance than absolute result precision. That is, an application may choose to sacrifice accuracy to obtain faster answers or use fewer resources, particularly if the degree of approximation can be controlled. One prevalent approach to this tradeoff is to store exact values in the database, then process queries using sampling or other statistical techniques to produce approximate answers, e.g., [AGPR99, BDF+97, GG01]. Another approach—the one we are interested in—is for the database itself to store approximate values, which can reduce communication and/or update costs depending on the exact application, e.g., [LM04, OW00].

A system like Trio that can manage simple approximate values and execute queries over them efficiently is more appropriate than a conventional DBMS in this setting. Furthermore, Trio's lineage capabilities can be used to associate approximate values with exact values when data approximation is used to reduce communication costs in a distributed system, e.g., [OLW01], facilitating the propagation of updates and exact values when resources are available.

2.7 Hypothetical Reasoning

A hypothetical reasoning system allows users to explore the effects of hypothetical situations, such as hypothetical changes or derivations in a database, e.g., [GH97, WS83]. Hypotheses are likely to have some uncertainty associated with them, and hypotheses may build on other hypotheses, which build on yet other hypotheses, and so on. Trio can be used to manage the creation and revocation of hypothetical database changes, to associate confidence with hypothetical data and derivations, and to query hypothetical data.

Recalling what Trio is not (Section 1.1), certainly Trio is not designed to be the ultimate system for performing complex hypothetical analyses, or for managing numerous alternative databases. However, for applications requiring relatively contained "what-if" capabilities that may include confidence, Trio provides convenient features not offered in a conventional DBMS.

2.8 Online Query Processing

Online query processing is a technique for providing approximate answers to users while more refined answers are still being computed [HH01]. Furthermore, users may provide feedback during the refinement phase to influence processing. For simple queries, the accuracy features of Trio may be of some use during online query processing, however the true benefit is obtained in complex Online Analytical Processing (OLAP) applications. OLAP queries typically rely on many layers of subqueries, views, and/or summarization. During online query processing in this setting, higher-level approximate answers are derived from lower-level approximations, so as lower-level values are refined the effects must propagate upwards. Exploiting derivation information to propagate accuracy modifications automatically is one of Trio's important features.

3 Running Example: Christmas Bird Count

We introduce a specific real application as a running example for the remainder of the paper: data management for the *Christmas Bird Count* (CBC). During the bird count "tens of thousands of observers count millions of birds in thousands of locations," followed by "in-depth post-count analysis" [CBC]. In this paper we considerably simplify, abstract, and hypothesize some of the CBC data and functionality, to keep the examples short and compelling, but we do plan to ultimately model (if not implement) the application in full.

During the bird count, which has occurred annually for over 100 years, volunteers and professionals worldwide observe birds for a fixed period of time, recording their observations. The data is used to understand trends in bird populations (both numbers and locations), and to correlate bird-life with short-term and long-term environmental conditions. The following features make this application an excellent candidate for Trio:

- Individual bird sightings are not always precise in terms of species, location, or time.
- Some participants (e.g., professional ornithologists) may provide higher-confidence, higher-quality data than others (e.g., bird hobbyists).
- Many different views of the data are required, with many levels of transformation and aggregation. Nevertheless all raw data is maintained, and "drilling down" from aggregate to raw data is common.
- Outside data sources are incorporated, e.g., environmental and geologic data, land-use data, population figures, and so forth. Furthermore packaged transformations are used (e.g., mapping latitude-longitude to known regions; temporal clustering of sightings).
- Data is added continuously, and data may occasionally be corrected retroactively.

Specific examples using this application domain are sprinkled throughout the remainder of the paper.

4 The Trio Data Model

The *Trio Data Model* (*TDM*) formalizes and integrates the three building blocks of our system: data, accuracy, and lineage. Our goal is to identify a relatively simple core model motivated largely by the application domains described in Section 2. We may increase the complexity of the model over time if specific additional expressiveness is demanded by a large number of applications. However, since the Trio system is likely to support user-defined data types and functions (see Section 6), capabilities not provided within TDM can be "plugged in" as needed by specific applications. One of our main challenges is to identify how much to build into TDM's core data model, at the expense of complexity, and how much is left to individual applications, at the expense of inconvenience and possibly lower performance.

We emphasize that the model presented here is preliminary and subject to change as the Trio project unfolds.

4.1 TDM – Data

The basic data in TDM follows the standard relational model: A *database* is a set of uniquely-named *relations*. The *schema* of each relation is a set of typed *attributes*, uniquely named within the relation. An *instance* of a relation is a set of *tuples*, each containing a value (or NULL) for each of the attributes. An instance of a database consists of an instance for each relation in the database. TDM places no restrictions on the types from which attribute values are drawn: they may be typical atomic types (integer, float, string, date, enumeration, etc.), or they may be large binary objects, programs, file identifiers, URLs, and so on. Of course some data types are more amenable to approximate values than others.

4.2 TDM – Accuracy

Inaccuracy of the data in a Trio database instance may occur at the attribute level, tuple level, and/or relation level. In our initial model we take a fairly limited approach to each of these components, although already

we encounter some subtle interactions among them, as discussed under "*Approximation versus Confidence*" below.

We had some difficulty selecting terms to use for the different components of accuracy in TDM, not to mention the term "accuracy" itself. Previous work in this general area has used varied and inconsistent terminology; see Section 1.2. In addition to the lack of past consensus, another challenge was to avoid terms that connote specific semantic interpretations, since TDM's accuracy model is flexible enough for different applications to use its components in different ways.

TDM's accuracy model is comprised of the following three basic components (and corresponding names for them), elaborated in the next three subsections.

1. Atomic values: An attribute value may be an **approximation** of some (unknown) exact value.
2. Tuples: A tuple may have an associated **confidence**, typically indicating the likelihood the tuple is actually in the relation.
3. Missing Data: A relation may have an associated **coverage**, typically indicating how much of the correct relation is likely to actually be present.

Approximation

Individual attribute values in TDM may be approximations. Broadly, a Trio approximate value is comprised of a (possibly infinite) set of *possible values*, along with a *probability distribution* over that set. Initially we limit the sets and distributions in TDM to the ones enumerated below. As discussed earlier, we may choose to extend the model as demanded by applications, and we expect the Trio system will support user-defined types for more complex or application-specific approximations.

Each attribute value in TDM is in exactly one of the following four categories:

(A1) An exact value.

(A2) A set of possible values, each with an associated *probability* in the range $[0, 1]$ such that the probabilities sum to 1. (See additional discussion below.)

(A3) Endpoints for a range of possible values, when values are drawn from an ordered and possibly continuous domain (e.g., integer, float, date). The basic Trio model assumes a uniform probability distribution across values in a *minimum*/*maximum* range.

(A4) A Gaussian distribution over a range of possible values (again assuming an ordered domain), denoted by a *mean*/*standard-deviation* pair [FGB02].

For approximation type A2—a set of possible values—we do permit probabilities whose sum $p < 1$, by stipulating an additional special value $\perp$ ("unknown") in the set with probability $1 - p$. Explicit probabilities may be omitted, in which case $\perp$ may or may not be specified as a member of the set, and a uniform probability distribution over the set members is assumed.

We realize there are numerous other options for approximate data, as discussed in Section 1.2. In keeping with our goal of deploying a simple and usable first-version prototype system, our tentative plan is to limit TDM to these types of approximation. A further simplification is that we assume independence of approximate attribute values in the same tuple. For example, in a bird sighting the exact value of an approximate `color` is probably correlated with the exact value of an approximate `species`, but TDM treats the approximations as independent.

By default, attribute values are expected to be exact (category A1). Note that some special cases of approximate values also can be treated as exact:

- A category-A2 singleton set whose one element has *probability* $= 1$
- A category-A3 range containing a single value
- A category-A4 Gaussian with *standard-deviation* $= 0$

Also note that a category-A2 singleton set $\{\perp\}$ may be considered equivalent to NULL, unless the application intends to interpret NULL as something different from "unknown."

Confidence

Tuple-level accuracy in the Trio Data Model is encoded in *confidence values*. Each tuple is accompanied by a *confidence* in the range $[0, 1]$, denoting the likelihood that the tuple correctly belongs in its relation. By default each tuple has *confidence* $= 1$. As a shortcut we also permit relation-level confidence: A relation with *confidence* $= c$ is equivalent to a relation whose tuples all have *confidence* $= c$.[1]

Coverage

The third and final type of inaccuracy captured in TDM is missing records. Each Trio relation has an associated *coverage* value in the range $[0, 1]$, denoting the estimated portion of the intended complete relation that is actually present. (Missed sensor readings as described in Section 2.2 are an example for *coverage* < 1, and a CBC example is given below.) By default, all relations have *coverage* $= 1$.

Accuracy in the CBC Application

Consider the Christmas Bird Count application introduced in Section 3. Suppose each participant P records their raw observations in their own private relation Obs^P. All of the observation relations use the same schema (simplified considerably from the actual CBC schema for illustration):

```
Obs(time, latitude, longitude, species)
```

All four categories of approximation can occur in attribute values. Values for attribute `time` may be exact (category A1), or may specify a range (category A3) or Gaussian (category A4); similarly for `latitude` and `longitude`. Attribute `species` is particularly interesting: An observer may identify a single species with complete confidence (category A1: an exact value), with less than full confidence (category A2: one value with *probability* < 1), or may be certain the observed bird was one of a set of possible species (category A2: multiple values).

Both confidence and coverage may be relevant as well. Relation-level confidence offers a convenient mechanism for encoding the expected overall accuracy of the observations in a given Obs^P, perhaps based on the experience level of participant P,[2] or the conditions under which P was working. Furthermore, ideally relation Obs^P records all birds visiting the observed area during the observation time period. Coverage can be used to encode the estimated fraction of those visits actually recorded in Obs^P.

[1] We may find that relation-level confidence is far more common than tuple-level confidence. If so, we may decide to eliminate tuple-level confidence from TDM in the interest, as always, of keeping the model simple.

[2] Recall that participants in the bird count range from seasoned ornithologists to novice bird-watchers, so confidence may vary considerably.

Approximation versus Confidence

There are some subtle differences between approximation and confidence in TDM. For example, consider the following two database instances for a Trio relation $R(A)$:

1. R contains a single tuple whose A value is a category-A2 set $\{a, b, c, d\}$. With the default uniform distribution, each value has *probability* $= 0.25$.

2. R contains four tuples, (a), (b), (c), and (d), each with *confidence* $= 0.25$.

In the first case, one value belongs in the database: either a, b, c, or d, with equal probability. In the second case, between 0 and 4 values belong in the database, where each value has an independent 25% chance of belonging.

Consider a second pair of instances for the same relation $R(A)$:

1. R contains a single tuple whose A value is a category-A2 singleton set $\{\langle c, 0.5\rangle\}$. (As an aside, notice this set-approximation is equivalent to $\{c, \perp\}$.)

2. R contains a single tuple (c) with *confidence* $= 0.5$.

In the first case, one value belongs in the database, with a 50% chance of it being the value c, and 50% an unknown other value. In the second case, either 0 or 1 values belong in the database with equal probability, and if there is a value then it is c.

Of course we may combine approximation and confidence. For example, suppose R contains a single tuple whose A value is category-A2 set $\{\langle c, 0.6\rangle\}$, and the tuple itself has *confidence* $= 0.8$. Then we have an 80% chance of a tuple belonging in R, and if it does, with 60% chance the value of A in that tuple is 'c'.

These examples highlight that even with a fairly simple model of accuracy, subtleties arise quickly, and important modeling decisions must be made when encoding an application's data in TDM. Clearly a more expressive model would lead to even more decisions, and more complexity and confusion when processing the data—for humans, applications, and for the Trio system itself.

4.3 TDM – Lineage

Formalizing the lineage component of the Trio Data Model is even more open-ended than formalizing accuracy. In general, the *lineage* of data describes how the data came into existence and how it has evolved over time [FB01, FVWZ02, WS97]. In our initial model we focus on lineage at the tuple level, although we may later expand to include attribute-level and/or relation-level lineage.

Before discussing lineage, let us digress for a moment and consider how modifications and deletions are handled in Trio. Our current inclination is that Trio data is never updated in place, and never deleted. Rather, when updates occur, new data values are inserted in the database while old values are "expired" but remain accessible in the system. Similarly, deleted data is expired but not actually expunged. This overall approach is similar to *no-overwrite storage* as introduced by Postgres [Sto87]. It also connotes full *temporal databases* [SA86], but as discussed in Section 1.1 we do not plan to include expressive temporal operators or query constructs as part of Trio.

We obtain at least three advantages by using a no-overwrite approach in Trio:

1. **Historical lineage:** If a data item I is derived from data D at time T, and D is subsequently updated, we can still obtain I's original lineage data from the expired portion of the database.

2. **Phantom lineage:** As part of lineage we may be interested in explaining why certain data is *not* in the database. With the no-overwrite approach we can support this capability in a limited form (for deleted data), although the general problem of phantom lineage remains an interesting challenge for the future.

3. **Versioning:** The no-overwrite approach enables Trio to support at least some level of *versioning*, which may be demanded by several of the applications we are targeting (most notably scientific data management [GST+02], but also sensor data management, deduplication, and others). It is our hope that simple versioning features in Trio will go hand-in-hand with lineage: that basic lineage capabilities will help support versioning, and vice-versa.

Let us begin with a general description of the lineage information we wish to capture, then specify more precisely the lineage component of TDM. Given a tuple t, there are three main aspects to t's lineage:[3]

- **When** t was derived
- **How** t was derived
- **What** data was used to derive t

We keep the **when** component fairly simple: Tuple t was either derived *now* because t is the result of a function defined over other data in the database (e.g., t is part of a view), or t was derived at a given time in the past, which we refer to as a *snapshot* derivation.

For the **how** component, we separate the following five categories based on type of derivation.

- **Query-based:** t was derived by a TriQL (or SQL) query. The query may define a view, or it may be a query that was executed in the past and the results were added to the database.

- **Program-based:** t was derived as the result of running a program, which may have accessed the database, and whose results were added to the database. We assume programs are "black boxes" in the sense that we cannot analyze their specific behavior. However, if we know which relations a program accesses, that information is part of t's lineage. For programs we consider only snapshot derivations, i.e., we do not cover up-to-date ("now" derivation) views defined by black-box programs.

- **Update-based:** t was derived as the result of modifying a previous database tuple t'. The modification may have been precipitated by a TriQL (or SQL) update statement, or by a program in which case we have the same considerations as program-based derivations for inserted data.

- **Load-based:** t was inserted as part of a bulk data load.

- **Import-based:** t was added to the database as part of an import process from one or more outside data sources. Data import may incorporate packaged transformations with known properties, as in *Extract-Transform-Load* (ETL) scenarios [CW03, FVWZ02], but for now we treat the import process as a black-box program for which we know only the sources that are accessed. As with program-based derivations, we assume snapshot only.

Note two important cases that fall into this categorization but were not mentioned explicitly: (1) If a SQL `insert-values` statement is used to insert a tuple t, then t has a very simple snapshot query-based

[3] A possible fourth aspect is **who** caused t to be derived, which we could add to our model easily if needed.

lineage. (2) Suppose a database trigger is invoked, and its action inserts new data. Since SQL trigger actions are either queries or programs, we can treat these actions as separate operations for the purposes of lineage. Unfortunately, we lose the relationship between the trigger and the inserted data (not to mention what caused the trigger to be invoked in the first place), a simplification we may wish to revisit in the future.

The third lineage component, **what** data was used to derive t, is potentially the trickiest. Previous work applies in certain cases: For query-based "now" derivations, there are known algorithms for identifying the "exact" data producing t based on the structure of the query.[4] This lineage data can be explicit, as in [BCTV04], or it can be implicit through a *lineage query* applied to the current database, as in [CWW00]. These techniques also can be applied to snapshot query-based derivations: We can record the identified lineage data at derivation time, or apply a lineage query to the state of the database at the time of derivation by possibly accessing "expired" data as suggested at the beginning of this section.

Two important notes regarding lineage data and queries as discussed in the previous paragraph:

1. Some of the existing algorithms explicitly capture recursively-defined lineage, e.g., for views layered on other views [BCTV04, CWW00]. We assume one level of lineage in our model, however we propose traversing lineage relationships recursively as part of the TriQL query language; see Section 5.1.

2. Occasionally the lineage of a tuple t may be based on the absence rather than presence of certain data, e.g., if t is based on a relational `minus` operation. This issue is not specific to Trio—it pervades work in relational data lineage (see, e.g., [CWW00]), and Trio may adopt any of the proposed solutions.

Now consider the remaining four types of derivations, and what data we wish to capture in the lineage for each type. Often we have two choices: *schema-based* lineage, which tends to be compact and coarse-grained, or *instance-based* lineage, which without a derivation query to guide us may amount to the entire database state at a given point in time, or the entire contents of a load file. (Note this distinction is not new: both schema-based and instance-based lineage have been considered in previous work.) Given the trends in low-cost massive storage devices, we do not rule out instance-based lineage even when the data may be very large. Thus we offer the following options:

- For program-based derivations, the lineage data may be unknown, it may be a list of relations accessed by the program (possibly zero-length), it may be the contents of those relations at the time of derivation, or it may be the entire database at the time of derivation if which relations were accessed is not known.

- For update-based derivations, the lineage obviously includes the previous value of the updated tuple. In addition, if the update was precipitated by a TriQL update statement or a program, the lineage may include that of a query-based or program-based derivation.

- For load-based derivations, the lineage data may be unknown, or it may be the contents of one or more load files at the time of derivation. (Note that if a tuple t was derived from specific loaded data, that case is better captured as a query-based derivation.)

- For import-based derivations, the lineage data may be unknown, or it may be the entire set of imported data, as in load-based derivations.

Lineage in TDM

Now we formalize lineage in the Trio Data Model. Logically (and possibly physically), every Trio database instance contains a special *lineage relation*. We assume database tuples have unique identifiers—a realistic

[4] In [BCTV04, BKT01], an interesting distinction is drawn between *where lineage* and *why lineage* in the definition for the "exact lineage" of a view tuple t. In Trio it is possible we will need to support both types.

Derivation type	Attribute `how-derived`	Attribute `lineage-data`
Query-based	query text	exact lineage data, or lineage query over current or expired data
Program-based	pointer to program, or program code; parameter values used; program version info (if any)	UNKNOWN, or list of relations, or contents of relations, or entire database
Update-based	update statement text, or program invocation info (see program-based)	ID of previous tuple t'; see query-based or program-based
Load-based	name of load file(s)	UNKNOWN, or load file contents
Import-based	source descriptor(s); program invocation info (see program-based)	UNKNOWN, or entire imported data set

Table 1: Contents of attributes `how-derived` and `lineage-data` in special relation `Lineage`.

assumption in most DBMSs—and that tuple IDs are not reused over time (less realistic, but still plausible). The lineage relation has the following schema:

```
Lineage(tupleID, derivation-type, time, how-derived, lineage-data)
```

Assume for now that all attribute values in the lineage relation are *exact* with respect to TDM's accuracy model from Section 4.2; we discuss the possibility of inexact lineage below. Note that with the no-overwrite strategy discussed at the beginning of this section, `tupleID` is a key for the `Lineage` relation.

The content of a `Lineage` tuple t_L corresponding to a database tuple t is defined based on t's derivation type. In all cases, attribute `time` is either a timestamp or the special symbol NOW. The contents of the remaining two attributes, `how-derived` and `lineage-data`, are described in Table 1. Obviously many of these attribute values can have very complex types, and possibly extremely large values—perhaps even identifying a substantial fraction of some database state. Here we are concerned primarily with the data model itself. A number of encoding strategies for these attribute values are possible, depending on Trio's data storage mechanisms as well as overall system architecture, discussed briefly in Section 6.

TDM Lineage Considerations

Here are some additional considerations for our proposed formalization of lineage in the Trio Data Model.

- An obvious alternative model would be to associate lineage information with tuples directly, instead of defining lineage as a separate relation, especially given the one-to-one mapping between data tuples and lineage tuples. We believe that separating lineage information into its own relation has some advantages, both conceptually and from an implementation perspective. For example, as discussed in the third point, we can model inexact lineage using TDM's accuracy model, and we can selectively record lineage for some data tuples but not others.

- Derivations or lineage information may sometimes be coarse enough that an entire relation has the same lineage. With import-based or load-based derivations, for example, the entire contents of a

relation R may be generated by a bulk load, or by an import from an outside source. We can model this scenario as identical `Lineage` tuples for all tuples $t \in R$, but obviously it makes more sense to encode the entire relation's lineage in one lineage tuple, replacing `tupleID` with an identifier for R.

- Since our lineage information is modeled as a relation, we could make it a Trio relation and incorporate inaccuracy. One obvious candidate is attribute `time`: Trio may not be able to identify the exact derivation time for some data, in which case *probability* < 1 could be associated with the specified time (approximation category A2 from Section 4.2), or more likely time could be specified as a range known to contain the correct derivation time (category A3) or a Gaussian distribution (category A4). Attributes `how-derived` and `lineage-data` are less obviously amenable to approximation, although set-valued approximations (category A2) could be appropriate depending on the circumstances. Currently we don't see a clear use for < 1 confidences or < 1 coverage (see Section 4.2) in the `Lineage` relation for this application, but we don't rule out the possibility.

- Although storage is not necessarily a significant constraint in modern database systems, the capture and management of lineage may be a complex and potentially expensive task. Thus, by default data in TDM does not include lineage—more formally, by default the lineage tuple t_L for a given database tuple t is not present in relation `Lineage`. Instead, applications are expected to activate lineage capabilities explicitly as needed. For now we assume lineage is activated at relation-definition time, and activation applies to entire relations and not individual tuples.

Lineage in the CBC Application

Our running example application from Section 3 has numerous requirements that can exploit lineage capabilities. We give but a few examples here, since the main contribution of Trio is not to introduce or justify lineage management on its own, which has been done before, but rather to integrate lineage with accuracy, and support both in a usable working prototype.

In the CBC application, snapshot query-based derivations (recall Section 4.3) may be the most prevalent. Each January the participants' raw bird observation data for that year is merged into a global data set suitable for queries and further processing. In addition, annual data is combined with data from previous years to compute new statistics and trends.

The CBC application also includes a significant number of import-based derivations, since a major goal of the bird count is to correlate bird data with environmental, geologic, and population data, and this data must be imported from outside sources.[5] Update-based derivations are needed when data is corrected. Finally, query-based "now" derivations are needed for the many levels of views used by different scientists, while load-based derivations apply at least to the uploaded raw participant data.

5 TriQL: The Trio Query Language

The Trio Query Language, *TriQL* (pronounced "treacle"), is designed for querying and modifying data in the Trio Data Model. We do not propose a specific syntax for TriQL in this paper; rather, we present informally our initial ideas for core functionality in the language. Like TDM, our intention is to keep TriQL relatively simple initially, so we can build incrementally and introduce functionality and complexity only when we are certain we need it. Here too, a primary challenge is to understand which extended functionality is fundamental enough that it should be built into Trio, and which should be left to application-specific user-defined functions.

[5] Alternatively these data sources may be accessed during query processing, in a federated style that for now is not given special treatment in Trio; recall Section 1.1.

5.1 Queries in TriQL

TriQL extends the SQL query language: it extends the semantics of SQL queries so they can be issued against data that includes accuracy and lineage, and it adds new features to SQL for queries involving accuracy and lineage explicitly. We base TriQL on SQL largely because SQL is most familiar to our potential users, and it includes a number of constructs necessary for our motivating applications: grouping and aggregation, `like` predicates, expression evaluation, and so on.

In developing TriQL we plan to first define an underlying formal algebra—most likely an extension of relational algebra that incorporates accuracy and lineage. Regardless of the formal model, one absolute premise of TriQL is *closure*: the result of a TriQL query (or subquery) over a Trio database is itself in TDM, i.e., TriQL query results incorporate the same model of accuracy and lineage as Trio stored data.

Syntactically Unextended SQL

The first step in designing TriQL is to specify the semantics of "regular" SQL queries applied to Trio data. When we execute a TriQL query, lineage for tuples in the query result can be computed largely following previous work on lineage in SQL, e.g., [BCTV04, CWW00]. Thus we focus here on the accuracy components of TDM, as specified in Section 4.2. First note that we do not build in result ranking based on accuracy. While ranked results certainly have motivation and merit, and there has been some interesting work in this area, e.g., [ACDG03, DS04, Fuh90], we maintain the unordered relational model in both data and query results. Satisfactory result ranking may often be achievable simply by ordering result tuples on their *confidence* values, which TriQL will support.

Even within the scope of TDM and unordered results, developing a complete semantic specification is no simple task—it requires considering every operator in SQL and specifying how the operator behaves over data that may include approximate attribute values, tuples with < 1 confidence, and relations with < 1 coverage. Here are a few illustrations of the semantic decisions that must be made:

- When we join two tuples each with *confidence* < 1, what is the confidence of the result tuple?
- When we combine two relations with *coverage* < 1, what is the coverage of the result relation for the different combination operators (e.g., join, union, intersection)?
- When we aggregate values in tuples with *confidence* < 1, what is the form of our result? Drawing from our CBC application (Section 3), suppose a participant P reports five sightings of a bird species B, and P's observations all have *confidence* = 0.9. How many sightings do we report for B? It probably makes the most sense to return an approximate count value using our accuracy model, in a tuple with *confidence* $= 1$ (since we are confident some count does exist).
- How do we aggregate values from multiple approximation categories? In our CBC example, suppose in addition to the five B sightings with confidence 0.9, two other participants report seeing either species B (*probability* $= 0.35$) or species C (*probability* $= 0.65$) with full confidence. Now how many sightings do we report for B? And what if our sightings relation has *coverage* < 1?

The large body of previous work in this general area (recall Section 1.2) should guide us in identifying and choosing among the alternatives when working out the detailed semantic specification, and we hope to benefit from our decision to keep the TDM accuracy model relatively contained.

Additions to SQL

Now let us consider additions to SQL that go beyond a new semantic interpretation for "regular" queries.

Accuracy Predicates. TriQL should build in a set of common predicates for filtering based on accuracy. For example, using the CBC application:

- *Return only sightings whose* `species` *attribute is exact.*
- *Return only sightings whose* `time` *range approximation is within 10 minutes.*
- *Return only sightings with* $\geq 98\%$ *chance of belonging in the database*.

We expect an appropriate set of predicates should be relatively easy to identify and define based on our suite of motivating applications (Section 2) and previous work. Note that these predicates should be applicable to subquery results as well as to stored data. Fortunately the semantics for this case falls out directly: Subqueries return results in TDM based on the new semantics for general queries discussed in the previous subsection. Then accuracy predicates are applied to the subquery results just as they would be applied to tuples in a Trio stored relation.

Lineage Queries. TriQL should permit queries over lineage, as well as queries that traverse lineage relationships recursively. We could simply permit the `Lineage` relation to be referenced in queries, and that alone does enable some useful capabilities. However, we believe special-purpose constructs are needed, particularly when exploiting information buried inside the attribute values modeled in the `Lineage` relation (recall Table 1). Here are some examples, both simple and more complex, using the CBC application:

- *Identify the program and its parameters used to derive the* `region` *attribute in a* `summary-by-region` *table.*
- *Retrieve all temperature data imported from climate database* C *on 4/1/04.*
- *Retrieve all tuples whose derivation includes data from relation* Obs^P *for a specific participant* P.
- *Retrieve all tuples whose derivation could have included a specific sighting tuple* t.
- *Given a sighting tuple* t *(or a relation* Obs^P*), find all data derived from it directly or indirectly.*

Integrating Accuracy and Lineage. TriQL queries should be able to integrate accuracy and lineage. It is possible that sufficient integration capabilities will fall out naturally by combining accuracy predicates and lineage query constructs, and this is the ideal situation in terms of orthogonality and implementation. The following examples illustrate the types of queries that should be expressible:

- *Retrieve only data derived entirely from sightings in observation relations with confidence* $\geq 98\%$. (Note the retrieved data may or may not have *confidence* ≥ 0.98, depending how it was derived.)
- *Identify relations with coverage* ≤ 0.95 *derived from climate database* C *more than one week ago.*
- Perhaps even: *Retrieve all summary data whose derivation query filters out approximate times with a greater than 10-minute approximation.* (Note the retrieved data may or may not have times within a 10-minute approximation, depending how it was derived.)

5.2 Data Modifications in TriQL

The data modification component of TriQL is particularly important because modifying accuracy may take on special meaning.[6] Modifying the accuracy of a tuple t (or relation R) should have the option of propagating automatically based on lineage to possibly modify the accuracy of data derived using t (or R), which may then propagate to accuracy of other data, and so on. Here accuracy can include attribute-value approximations, tuple-level confidence, and relation coverage (Section 4.2). This accuracy update-propagation feature is most applicable for query-based and program-based derivations (Section 4.3); note that in some scenarios a large recomputation may be needed. A related interesting case mentioned earlier is when we modify the perceived accuracy of an outside source: We may wish to modify the accuracy of all data imported from that source, which may propagate to other data derived from it, and so on.

The CBC application has many uses for accuracy update propagation; here are two simple examples:

- If we upgrade the confidence of an observation relation Obs^P based on new information about participant P, data derived using Obs^P should have its accuracy updated automatically.
- We may discover that a large misclassification of species occurred, requiring updates to accuracy in some summary data. These accuracy updates should propagate automatically to other data derived using the summaries.

We may wish to exploit lineage for non-accuracy-related modifications as well. If we update a tuple t,[7] and data D was derived using t at some point in the past, under some circumstances we may wish to invalidate, delete, or recompute D (which may itself propagate to data derived from D, and so on).

A further capability we are considering is "reverse" propagation of updates to data or accuracy, in which the system attempts to propagate changes made directly to derived data (or its accuracy) back to the original data from which it was derived. This capability entails solving a generalization of the *view update problem* [BS81], so there are many algorithmic and ambiguity issues to address. The lineage information maintained by Trio provides one building block for attacking these problems.

Note that some applications may be cautious—they may not want data or accuracy updates to be propagated by the system automatically. Even for these applications, Trio's features are useful: when data or accuracy is updated, the application can issue TriQL lineage queries to determine the potential effects and act accordingly.

6 System Implementation

We plan to begin system implementation as soon as initial versions of TDM and TriQL are concrete. We have several conflicting goals in the development of the Trio system:

1. **Rapid deployment.** We want to distribute a first-version prototype to pilot users as quickly as possible.
2. **Resilience.** We want the implementation to be resilient to changes and extensions to the data model and query language—some changes are inevitable once our first suite of Trio applications is prototyped.
3. **Efficiency.** We want even the first-version system to be efficient enough that users do not retreat to a conventional DBMS because of performance issues.

[6] Currently, we assume lineage is derived by the system and is not updatable, although we may revisit this assumption later.

[7] Recall from Section 4.3 that we do not plan to literally update t, but rather insert the new value and "expire" the old.

4. **Extensibility** We want the system to serve as a platform for experimenting easily with different methods of data management and query processing when we integrate data, accuracy, and lineage.

The first and primary decision is to choose one of the following three approaches:

1. *Implement TDM and TriQL on top of a conventional relational DBMS.* In this approach, every schema is extended to store accuracy and lineage in conventional relations, along with the base data. Data is mapped from TDM to conventional relations based on the schema extensions; TriQL queries are translated to conventional SQL queries over the extended schemas; query results are mapped from the data encoding back to TDM.

2. *Build a new data manager and query processor from scratch,* exploiting an existing back-end for low-level store.

3. *Implement TDM and TriQL in an extensible object-relational DBMS.* In this approach, TDM data is encoded as complex types managed by the underlying object-relational DBMS, and TriQL extensions to SQL are implemented as methods or stored procedures, or by extending the query processing engine.

Approach (1) certainly addresses our goal of rapid deployment, and our goal of resilience to changes since modifications to the model or query language manifest themselves as modifications to our data mapping algorithms and query rewrite engine. The primary potential drawbacks are in efficiency and extensibility.

Approach (2) naturally offers the converse benefits and drawbacks: Deployment would be delayed, and changes to the model or language would require significant recoding. On the other hand, we can fine-tune performance and experiment with alternate ideas in data management and query processing at all levels of the system. If we do consider this approach, we must carefully consider which aspects of a conventional DBMS are unaffected by incorporating accuracy and lineage. (Transaction management, concurrency control, and recovery are obvious examples.) If we cannot find a back-end store that captures a significant fraction of those capabilities, then approach (2) is far less attractive, since we are certainly not amenable to reimplementing significant well-understood portions of a DBMS.

Approach (3) offers an interesting middle ground between the other two approaches. In an open-source extensible object-relational DBMS such as *PostgreSQL* [Pos] or *Predator* [SP97], we can define new complex tuple types that store accuracy alongside attribute values, and that encode the potentially large and complex attribute values in our `Lineage` relation. Special query processing over these types can be implemented as methods or stored procedures (potentially inhibiting optimization), or can be incorporated into the operators of the query processing engine. Although this approach involves coding "inside" the database system unlike approach (1), it largely frees us from dealing with aspects of conventional DBMS processing that are unaffected by our extensions, one of the potential drawbacks of approach (2).

7 Conclusion

We are launching a new project, *Trio*, that extends conventional data management by incorporating *accuracy* and *lineage* as integral components of both data and queries. Individually, accuracy and lineage have received considerable attention in the past, but frequently from a theoretical perspective, and certainly not as part of a comprehensive system that incorporates both features as a fully integrated part of data management and query processing.

This paper makes a case for the Trio system by describing numerous application domains that could benefit from its capabilities, as well as a concrete running example. It presents our initial ideas for the Trio Data Model and TriQL query language, and discusses approaches for system implementation.

Acknowledgments

In reverse chronological order:

- Thanks to Omar Benjelloun, Ashok Chandra, Anish Das Sarma, Alon Halevy, Utkarsh Srivastava, and Evan Fanfan Zeng for joining up and turning Trio into a real project.
- Thanks to Jim Gray and Wei Hong for useful discussions on Trio applications.
- Thanks to David DeWitt, Jim Gray, and Dave Maier for insightful feedback and suggestions on earlier drafts of this paper.
- Thanks to Hector Garcia-Molina, Chris Olston, Jeff Ullman, and many Stanford Database Group students for helpful discussions in the formative stages.

References

[ACDG03] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *Proc. of the First Biennial Conference on Innovative Data Systems Research (CIDR '03)*, Pacific Grove, California, January 2003.

[AGPR99] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua approximate query answering system. In *Proc. of the 2001 ACM SIGMOD Intl. Conference on Management of Data*, pages 574–576, Philadelphia, Pennsylvania, June 1999.

[AKSX02] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *Proc. of the 28th Intl. Conference on Very Large Databases*, pages 143–154, Hong Kong, China, August 2002.

[BB99] P.A. Bernstein and T. Bergstraesser. Meta-data support for data transformations using Microsoft Repository. *IEEE Data Engineering Bulletin*, 22(1):9–14, March 1999.

[BCTV04] D. Bhagwat, L. Chiticariu, W.C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. In *Proc. of the 30th Intl. Conference on Very Large Databases*, Toronto, Canada, August 2004.

[BDF+97] D. Barbará, W. DuMouchel, C. Faloutsos, P.J. Haas, J.M. Hellerstein, Y.E. Ioannidis, H.V. Jagadish, T. Johnson, R.T. Ng, V. Poosala, K.A. Ross, and K.C. Sevcik. The New Jersey data reduction report. *IEEE Data Engineering Bulletin*, 20(4):3–45, December 1997.

[BGMP92] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. on Knowledge and Data Engineering*, 4(5):487–502, October 1992.

[BKT01] P. Buneman, S. Khanna, and W.C. Tan. Why and where: A characterization of data provenance. In *Proc. of the 8th Intl. Conference on Database Theory*, pages 316–330, London, England, January 2001.

[BKTT04] P. Buneman, S. Khanna, K. Tajima, and W.C. Tan. Archiving scientific data. *ACM Transactions on Database Systems*, 29(2):2–42, June 2004.

[BP82] B. Buckles and F. Petry. A fuzzy model for relational databases. *International Journal of Fuzzy Sets and Systems*, 7:213–226, 1982.

[BP93] R.S. Barga and C. Pu. Accessing imprecise data: An approach based on intervals. *IEEE Data Engineering Bulletin*, 16(2):12–15, June 1993.

[BS81] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems*, 6(4):557–575, December 1981.

[CBC] Christmas Bird Count Home Page. http://www.audubon.org/bird/cbc/.

[CGG+03] S. Chaudhuri, K. Ganjam, V. Ganti, , and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proc. of the 2003 ACM SIGMOD Intl. Conference on Management of Data*, pages 313–324, San Diego, California, June 2003.

[CKP03] R. Cheng, D.V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. of the 2003 ACM SIGMOD Intl. Conference on Management of Data*, pages 551–562, San Diego, California, June 2003.

[CW03] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *VLDB Journal*, 12(1):41–58, May 2003.

[CWW00] Y. Cui, J. Widom, and J. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems*, 25(2):179–227, June 2000.

[DMB+01] L.M.L. Delcambre, D. Maier, S. Bowers, M. Weaver, L. Deng, P. Gorman, J. Ash, M. Lavelle, and J. Lyman. Bundles in captivity: An application of superimposed information. In *Proc. of the 17th Intl. Conference on Data Engineering*, pages 111–120, Heidelberg, Germany, April 2001.

[DS04] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. of the 30th Intl. Conference on Very Large Databases*, Toronto, Canada, August 2004.

[EEV02] M.G. Elfeky, A.K. Elmagarmid, and V.S. Verykios. Tailor: A record linkage tool box. In *Proc. of the 18th Intl. Conference on Data Engineering*, pages 17–28, San Jose, California, February 2002.

[Fag02] R. Fagin. Combining fuzzy information: An overview. *ACM SIGMOD Record*, 31(2):109–118, June 2002.

[FB01] J. Frew and R. Bose. Earth System Science Workbench: A data management infrastructure for earth science products. In *Proc. of the 13th Intl. Conference on Scientific and Statistical Database Management*, pages 180–189, Fairfax, Virginia, July 2001.

[FGB02] A. Faradjian, J. Gehrke, and P. Bonnet. GADT: A probability space ADT for representing and querying the physical world. In *Proc. of the 18th Intl. Conference on Data Engineering*, pages 201–211, San Jose, California, February 2002.

[Fuh90] N. Fuhr. A probabilistic framework for vague queries and imprecise information in databases. In *Proc. of the 16th Intl. Conference on Very Large Databases*, pages 696–707, Brisbane, Australia, August 1990.

[FVWZ02] I.T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proc. of the 14th Intl. Conference on Scientific and Statistical Database Management*, pages 37–46, Edinburgh, Scotland, July 2002.

[GFS+01] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *Proc. of the 27th Intl. Conference on Very Large Data Bases*, pages 371–380, Rome, Italy, September 2001.

[GG01] M.N. Garofalakis and P.B. Gibbons. Approximate query processing: Taming the terabytes. In *Proc. of the 27th Intl. Conference on Very Large Data Bases*, Rome, Italy, September 2001.

[GH97] T. Griffin and R. Hull. A framework for implementing hypothetical queries. In *Proc. of the 1997 ACM SIGMOD Intl. Conference on Management of Data*, pages 231–242, Tucson, Arizona, May 1997.

[GST+02] J. Gray, A.S. Szalay, A.R. Thakar, C. Stoughton, and J. vandenBerg. Online scientific data curation, publication, and archiving. Technical Report MSR-TR-2002-74, Microsoft Research, July 2002.

[HH01] P.J. Haas and J.M. Hellerstein. Online query processing. In *Proc. of the 2001 ACM SIGMOD Intl. Conference on Management of Data*, Santa Barbara, California, May 2001.

[HN00] H.-I. Hsiao and I. Narang. DLFM: A transactional resource manager. In *Proc. of the 2000 ACM SIGMOD Intl. Conference on Management of Data*, pages 518–528, Dallas, Texas, May 2000.

[HS98] M.A. Hernández and S.J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, January 1998.

[IL84] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, October 1984.

[JLM03] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *Proc. of the 8th Intl. Conference on Database Systems for Advanced Applications (DASFAA '03)*, pages 137–148, Kyoto, Japan, March 2003.

[KOR96] S.K. Kwan, F. Olken, and D. Rotem. Uncertain, incomplete, and inconsistent data in scientific and statistical databases. In A. Motro and P. Smets, editors, *Uncertainty Management in Information Systems: From Needs to Solution*. Kluwer Academic Publishers, Boston, 1996.

[Kum03] V. Kumar, editor. *Special Issue on Sensor Network Technology and Sensor Data Management*, ACM SIGMOD Record 32(4), December 2003.

[Lee92] S.K. Lee. An extended relational database model for uncertain and imprecise information. In *Proc. of the 18th Intl. Conference on Very Large Databases*, pages 211–220, Vancouver, Canada, August 1992.

[LLRS97] L.V.S. Lakshmanan, N. Leone, R. Ross, and V.S. Subrahmanian. ProbView: A flexible probabilistic database system. *ACM Transactions on Database Systems*, 22(3):419–469, September 1997.

[LM04] I. Lazaridis and S. Mehrotra. Approximate selection queries over imprecise data. In *Proc. of the 20th Intl. Conference on Data Engineering*, pages 140–152, Boston, Massachusetts, March 2004.

[LS90] K.-C. Liu and R. Sunderraman. Indefinite and maybe information in relational databases. *ACM Transactions on Database Systems*, 15(1):1–39, March 1990.

[MD99] D. Maier and L. Delcambre. Superimposed information for the internet. In *Proc. of the 1999 ACM Workshop on the Web and Databases (WebDB '99)*, Philadelphia, Pennsylvania, June 1999.

[Mon97] A.E. Monge. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'97)*, Tucson, Arizona, May 1997.

[Mot94] A. Motro. Management of uncertainty in database systems. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability, and Beyond*, pages 457–476. ACM Press, New York, 1994.

[OLW01] C. Olston, B.T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *Proc. of the 2001 ACM SIGMOD Intl. Conference on Management of Data*, pages 355–366, Santa Barbara, California, May 2001.

[OO93] A. Ola and G. Özsoyoglu. Incomplete relational database models based on intervals. *IEEE Trans. on Knowledge and Data Engineering*, 5(2):293–308, April 1993.

[OW00] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proc. of the 26th Intl. Conference on Very Large Data Bases*, pages 144–155, Cairo, Egypt, September 2000.

[Pos] PostgreSQL Home Page. http://www.postgresql.org/.

[RH01] V. Raman and J.M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proc. of the 27th Intl. Conference on Very Large Data Bases*, pages 381–390, Rome, Italy, September 2001.

[SA86] R. Snodgrass and I. Ahn. Temporal databases. *IEEE Computer*, 19(9):35–42, September 1986.

[Sad91] F. Sadri. Modeling uncertainty in databases. In *Proc. of the 7th Intl. Conference on Data Engineering*, pages 122–131, Kobe, Japan, April 1991.

[Sar00] S. Sarawagi, editor. *Special Issue on Data Cleaning*, IEEE Data Engineering Bulletin 23(4), December 2000.

[SB02] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. of the 8th ACM SIGKDD Intl. Conference on Knowledge Discovery and Data Mining*, pages 269–278, Edmonton, Canada, July 2002.

[SM01] M. Shapcott S. McClean, B. Scotney. Aggregation of imprecise and uncertain information in databases. *IEEE Trans. on Knowledge and Data Engineering*, 13(6):902–912, November 2001.

[SP97] P. Seshadri and M. Paskin. PREDATOR: An OR-DBMS with enhanced data types. In *Proc. of the 1997 ACM SIGMOD Intl. Conference on Management of Data*, pages 568–571, Tucson, Arizona, May 1997.

[SS98] P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information. In *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, page 188, Seattle Washington, June 1998.

[Sto87] M. Stonebraker. The design of the POSTGRES storage system. In *Proc. of the 13th Intl. Conference on Very Large Databases*, pages 289–300, Brighton, England, September 1987.

[WM90] Y.R. Wang and S.E. Madnick. A polygen model for heterogeneous database systems: The source tagging perspective. In *Proc. of the 16th Intl. Conference on Very Large Databases*, pages 519–538, Brisbane, Australia, August 1990.

[WS83] J. Woodfill and M. Stonebraker. An implementation of hypothetical relations. In *Proc. of the 9th Intl. Conference on Very Large Databases*, pages 157–166, Florence, Italy, October 1983.

[WS97] A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Proc. of the 13th Intl. Conference on Data Engineering*, pages 91–102, Birmingham, England, April 1997.

[YLW+95] Q. Yang, C. Liu, J. Wu, C. Yu, S. Dao, H. Nakajima, and N. Rishe. Efficient processing of nested fuzzy SQL queries in fuzzy databases. In *Proc. of the 11th Intl. Conference on Data Engineering*, pages 131–138, Taipei, Taiwan, March 1995.

[Zim97] E. Zimányi. Query evaluation in probabilistic relational databases. *Theoretical Computer Science*, 171(1):179–219, January 1997.