Trio: A System for Data, Uncertainty, and Lineage

Jennifer Widom
Stanford University
Two faculty independently proclaim uncertainty as the next major theme in Computer Science

- One old-timer, one youngster
- Proclamations not motivated by their own (or our) research
Uncertainty in Databases

- Not a new idea — proposed 20+ years ago
- Most initial (18+ years) work largely theoretical; not much systems-building until recently
- But applications weren’t ready anyway
  ➔ Are they now?
The Trio database system, developed by Professor Jennifer Widom and her research team, can account for the uncertainty of data and its sourcing.
The “Trio” in Trio

1. Data
   Student #123 is majoring in Econ: \((123, \text{Econ}) \in \text{Major}\)

2. Uncertainty
   Student #123 is majoring in Econ or CS:
   \((123, \text{Econ} \parallel \text{CS}) \in \text{Major}\)
   With confidence 60% student #456 is a CS major:
   \((456, \text{CS 0.6}) \in \text{Major}\)

3. Lineage
   456 \in \text{HardWorker} derived from:
   \((456, \text{CS}) \in \text{Major}\)
   “CS is hard” \in \text{some web page}
The Trio database system, developed by Professor Jennifer Widom and her research team, can account for the uncertainty of data and its sourcing.
New Application Domains

• Many involve data that is uncertain (approximate, probabilistic, inexact, incomplete, imprecise, fuzzy, inaccurate,...)

• Many of the same ones need to track the lineage (provenance) of their data

Coincidence or Fate?
Original Motivation for the Project

New Application Domains

- Many involve data that is uncertain (approximate, probabilistic, inexact, incomplete, imprecise, fuzzy, inaccurate,...)
- Many of the same ones need to track the lineage (provenance) of their data

Neither uncertainty nor lineage is supported in current database systems
Sample Applications

Information extraction
- Find & label entities in unstructured text
- Often probabilistic

Information integration
- Combine data from multiple sources
- Inconsistencies

Scientific experiments
- Inexact/incomplete data
- Many levels of “derived data products”
Sample Applications

Sensor data management
• Approximate readings
• Missing readings
• Levels of data aggregation

Deduplication ("data cleaning")
• Object linkage, entity resolution
• Often heuristic/probabilistic

Approximate query processing
• Fast but inexact answers
Our Claim

The connection between uncertainty and lineage goes deeper than just a shared need by several applications.
Substantiation of Claim

[technical] Lineage:

1. Enables simple and consistent representation of uncertain data
2. Correlates uncertainty in query results with uncertainty in the input data
3. Can make computation over uncertain data more efficient

[fluffy] Applications use lineage to reduce or resolve uncertainty
Our Goal

Develop a new kind of database management system (DBMS) in which:

1. Data
2. Uncertainty
3. Lineage

are all first-class interrelated concepts

⇒ With all the “usual” DBMS features
Pop Quiz

Why should every slide be making you squirm in your seat?
Our Goal

Develop a new kind of database management system (DBMS) in which:

1. Data
2. Uncertainty
3. Lineage

are all first-class interrelated concepts

⇒ With all the “usual” DBMS features
The “Usual” DBMS Features

(From first lecture of my Intro. to Databases class)

1. Efficient,
2. Convenient,
3. Safe,
4. Multi-User storage of and access to
5. Massive amounts of
6. Persistent data
Standard Relational DBMSs

Persistent; Convenient
- All data stored in simple tables ("relations")
- Queries and updates via simple but powerful declarative language (SQL)

Multi-User; Safe
- Transactions

Massive; Efficient
- Storage and indexing structures
- Query optimization
Trio: What Changes

Persistent; Convenient

- All data stored in simple tables ("relations")
- Queries and updates via simple but powerful declarative language (SQL)

Multi-User; Safe

- Transactions

Massive; Efficient

- Storage and indexing structures
- Query optimization
Trio: What Changes

Persistent; Convenient
- All data stored in simple tables ("relations")
- Queries and updates via simple but powerful declarative language (SQL)

Multi-User; Safe - standard DBMS underneath
- Transactions

Massive; Efficient - standard DBMS underneath
- Storage and indexing structures
- Query optimization
Another “Trio” in Trio

1. Data Model
   Simplest extension to relational model that’s sufficiently expressive

2. Query Language
   Simple extension to SQL with well-defined semantics and intuitive behavior

3. System
   A complete open-source DBMS that people want to use
Another “Trio” in Trio

1. Data Model
   Uncertainty-Lineage Databases (ULDBs)

2. Query Language
   TriQL

3. System
   Trio-One — built on top of standard DBMS
1. Data Model
   *Uncertainty-Lineage Databases (ULDBs)*

2. Query Language
   *TriQL*

3. System
   *Trio-One* — built on top of standard DBMS

4. Demo
First a Disclaimer

We are not about machine learning or probabilistic reasoning!

We are about efficient and convenient storage, manipulation, and retrieval of large data sets (with uncertainty and lineage in them)
Running Example: Crime-Solving

\[
\text{Saw (witness, color, car)} \quad // \quad \text{may be uncertain}
\]
\[
\text{Drives (person, color, car)} \quad // \quad \text{may be uncertain}
\]
\[
\text{Suspects (person)} = \Pi_{\text{person}}(\text{Saw} \bowtie \text{Drives})
\]
In Standard Relational DBMS

<table>
<thead>
<tr>
<th>Saw (witness, color, car)</th>
<th>Drives (person, color, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>red</td>
</tr>
<tr>
<td>Betty</td>
<td>blue</td>
</tr>
<tr>
<td>Carol</td>
<td>green</td>
</tr>
<tr>
<td>Jimmy</td>
<td>red</td>
</tr>
<tr>
<td>Billy</td>
<td>blue</td>
</tr>
<tr>
<td>Frank</td>
<td>red</td>
</tr>
<tr>
<td>Frank</td>
<td>green</td>
</tr>
</tbody>
</table>

Create Table Suspects as
Select person
From Saw, Drives
Where Saw.color = Drives.color
And Saw.car = Drives.car

Suspects
Frank
Billy
An uncertain database represents a set of possible instances

- Amy saw either a Honda or a Toyota
- Jimmy drives a Toyota, a Mazda, or both
- Betty saw an Acura with confidence 0.5 or a Toyota with confidence 0.3
- Hank is a suspect with confidence 0.7
Our Model for Uncertainty

1. Alternatives
2. ‘?’ (Maybe) Annotations
3. Confidences
Our Model for Uncertainty

1. **Alternatives:** uncertainty about value
2. ‘?’ (Maybe) Annotations
3. Confidences

<table>
<thead>
<tr>
<th>Saw (witness, color, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
</tr>
</tbody>
</table>

Three possible instances
Our Model for Uncertainty

1. Alternatives
2. ‘?’ (Maybe): uncertainty about presence
3. Confidences

<table>
<thead>
<tr>
<th>Saw (witness, color, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
</tr>
<tr>
<td>Betty</td>
</tr>
</tbody>
</table>

Six possible instances
Our Model for Uncertainty

1. Alternatives

2. ‘?’ (Maybe): uncertainty about presence

3. Confidences

<table>
<thead>
<tr>
<th>Saw (witness, color, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amy</strong></td>
</tr>
<tr>
<td><em>red, Honda</em></td>
</tr>
<tr>
<td><strong>Betty</strong></td>
</tr>
<tr>
<td><em>blue, Acura</em></td>
</tr>
<tr>
<td><strong>Betty</strong></td>
</tr>
<tr>
<td><em>blue, Acura</em></td>
</tr>
</tbody>
</table>

*absent ≠ unknown*
Our Model for Uncertainty

1. Alternatives

2. ‘?’ (Maybe) Annotations

3. Confidences: weighted uncertainty

<table>
<thead>
<tr>
<th>Saw (witness, color, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
</tr>
<tr>
<td>Betty</td>
</tr>
</tbody>
</table>

Six possible instances, each with a probability
Models for Uncertainty

Our model (so far) is not especially new

We spent some time exploring the space of models for uncertainty

Tension between understandability and expressiveness

– Our model is understandable
– But it is not complete, or even closed under common operations
Closure and Completeness

Completeness
Can represent all sets of possible instances

 Closure
Can represent results of operations

Note: Completeness $\Rightarrow$ Closure
Our Model is Not Closed

<table>
<thead>
<tr>
<th>Saw (witness, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cathy</td>
</tr>
<tr>
<td>Honda</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drives (person, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jimmy, Toyota</td>
</tr>
<tr>
<td>Billy, Honda</td>
</tr>
<tr>
<td>Hank, Honda</td>
</tr>
</tbody>
</table>

Suspects = π_{\text{person}}(\text{Saw} \bowtie \text{Drives})

CANNOT correctly capture possible instances in the result
Lineage (provenance): “where data came from”

- Internal lineage
- External lineage

In Trio: A function $\lambda$ from data elements to other data elements (or external sources)
Example with Lineage

<table>
<thead>
<tr>
<th>ID</th>
<th>Saw (witness, car)</th>
<th>Drives (person, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Cathy Honda</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Billy, Honda</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Hank, Honda</td>
<td></td>
</tr>
</tbody>
</table>

Suspects = \( \Pi_{\text{person}}(\text{Saw} \bowtie \text{Drives}) \)

<table>
<thead>
<tr>
<th>ID</th>
<th>Suspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Jimmy</td>
</tr>
<tr>
<td>32</td>
<td>Billy</td>
</tr>
<tr>
<td>33</td>
<td>Hank</td>
</tr>
</tbody>
</table>

\( \lambda(31) = (11,2),(21,2) \)
\( \lambda(32,1) = (11,1),(22,1); \quad \lambda(32,2) = (11,1),(22,2) \)
\( \lambda(33) = (11,1), 23 \)

Correctly captures possible instances in the result
Example with Lineage

<table>
<thead>
<tr>
<th>ID</th>
<th>Saw (witness, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Cathy</td>
</tr>
<tr>
<td></td>
<td><strong>Honda</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Drives (person, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Jimmy, Toyota</td>
</tr>
<tr>
<td>22</td>
<td>Billy, Honda</td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>

Suspects = \( \pi_{person}(Saw \bowtie Drives) \)

<table>
<thead>
<tr>
<th>ID</th>
<th>Suspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Jimmy</td>
</tr>
<tr>
<td>32</td>
<td>Billy</td>
</tr>
<tr>
<td>33</td>
<td>Hank</td>
</tr>
</tbody>
</table>
Trio Data Model

Uncertainty-Lineage Databases (ULDBs)

1. Alternatives
2. ‘?’ (Maybe) Annotations
3. Confidences
4. Lineage

ULDBs are closed and complete
ULDBs: Lineage

Conjunctive lineage sufficient for most operations

- Disjunctive lineage for duplicate-elimination
- Negative lineage for difference

General case after several queries:

- Boolean formula
ULDBs: Minimality

A ULDB relation $R$ represents a set of possible instances

- Does every tuple in $R$ appear in some possible instance? (no extraneous tuples)

  Data-minimality

- Does every *maybe*-tuple in $R$ not appear in some possible instance? (no extraneous ‘?’s)

- Also Lineage-minimality
Data Minimality Examples

Extraneous ‘?’

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Billy</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Billy, Honda</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Minimality Examples

Extraneous tuple

Extraneous

Diane

Diane | Mazda

Diane

Diane | Acura

Diane | Mazda || Acura
ULDBs: Membership Questions

- Does a given tuple $t$ appear in some (all) possible instance(s) of $R$?
  Polynomial algorithms based on data-minimization

- Is a given table $T$ one of (all of) the possible instances of $R$?
  NP-Hard
Querying ULDBs  TriQL

Simple extension to SQL
Formal semantics, intuitive meaning
Query uncertainty, confidences, and lineage
Simple TriQL Example

Create Table Suspects as
Select person
From Saw, Drives
Where Saw.car = Drives.car

ID | Saw (witness, car) | Drives (person, car) |
---|---------------------|----------------------|
11 | Cathy, Honda || Mazda | Jimmy, Toyota || Jimmy, Mazda |
21 | Billy, Honda || Frank, Honda |
22 | Hank, Honda |

| ID | Suspects |
---|---------|
31 | Jimmy |
32 | Billy, Frank |
33 | Hank |

? \( \lambda(31)=(11,2),(21,2) \)
? \( \lambda(32,1)=(11,1),(22,1); \lambda(32,2)=(11,1),(22,2) \)
? \( \lambda(33)=(11,1),23 \)
Formal Semantics

Relational (SQL) query $Q$ on ULDB $D$

![Diagram showing the relationship between input data $D$, the query $Q$ on each instance $D_1, D_2, ..., D_n$, and the result $Q(D_1), Q(D_2), ..., Q(D_n)$]
TriQL: Querying Confidences

**Built-in function:** Conf()

SELECT person  
FROM Saw, Drives  
WHERE Saw.car = Drives.car  
AND Conf(Saw) > 0.5 AND Conf(Drives) > 0.8
TriQL: Querying Lineage

Built-in join predicate: Lineage()

SELECT Suspects.person
FROM Suspects, Saw
WHERE Lineage(Suspects,Saw)
AND Saw.witness = ‘Amy’

X ==> Y shorthand for Lineage(X,Y)
Operational Semantics

```
SELECT attr-list
FROM X1, X2, ..., Xn
WHERE predicate
```

Over standard relational database:

For each tuple in cross-product of $X_1, X_2, ..., X_n$

1. Evaluate the predicate
2. If true, project attr-list to create result tuple
Over ULDB:

For each tuple in cross-product of $X_1, X_2, \ldots, X_n$

1. Create “super tuple” $T$ from all combinations of alternatives
2. Evaluate predicate on each alternative in $T$; keep only the true ones
3. Project attr-list on each alternative to create result tuple
4. Details: ‘?’, lineage, confidences

```
SELECT attr-list
FROM X_1, X_2, \ldots, X_n
WHERE predicate
```
Operational Semantics: Example

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

<table>
<thead>
<tr>
<th>Saw (witness, car)</th>
<th>Drives (person, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cathy</td>
<td>Mazda</td>
</tr>
<tr>
<td>Honda</td>
<td></td>
</tr>
<tr>
<td>Jim</td>
<td></td>
</tr>
<tr>
<td>Mazda</td>
<td></td>
</tr>
<tr>
<td>Hank</td>
<td>Honda</td>
</tr>
</tbody>
</table>

Honda || Mazda
SELECT person
FROM Saw, Drives
WHERE Saw.car = Drives.car

<table>
<thead>
<tr>
<th>Saw (witness, car)</th>
<th>Drives (person, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cathy Honda</td>
<td></td>
</tr>
<tr>
<td>Cathy Mazda</td>
<td>Hank Honda</td>
</tr>
</tbody>
</table>

(Cathy,Honda,Jim,Mazda) || (Cathy,Honda,Bill,Mazda) || (Cathy,Mazda,Jim,Mazda) || (Cathy,Mazda,Bill,Mazda)
### Operational Semantics: Example

```sql
SELECT person
FROM Saw, Drives
WHERE Saw.car = Drives.car
```

<table>
<thead>
<tr>
<th>Saw (witness, car)</th>
<th>Drives (person, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cathy</td>
<td>Jim</td>
</tr>
<tr>
<td>Honda</td>
<td>Mazda</td>
</tr>
<tr>
<td>Mazda</td>
<td>Hank</td>
</tr>
<tr>
<td></td>
<td>Honda</td>
</tr>
</tbody>
</table>

(Cathy,Honda,Jim,Mazda) || (Cathy,Honda,Bill,Mazda) || (Cathy,Mazda,Jim,Mazda) || (Cathy,Mazda,Bill,Mazda)
Operational Semantics: Example

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

<table>
<thead>
<tr>
<th>Saw (witness, car)</th>
<th>Drives (person, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cathy</td>
<td>Jim</td>
</tr>
<tr>
<td>Honda, Mazda</td>
<td>Bill</td>
</tr>
<tr>
<td>Cathy</td>
<td>Mazda</td>
</tr>
<tr>
<td></td>
<td>Hank</td>
</tr>
<tr>
<td></td>
<td>Honda</td>
</tr>
</tbody>
</table>

(Cathy,Honda,Jim,Mazda) || (Cathy,Honda,Bill,Mazda) || (Cathy,Mazda,Jim,Mazda) || (Cathy,Mazda,Bill,Mazda)
Operational Semantics: Example

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

<table>
<thead>
<tr>
<th>Saw (witness, car)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cathy</td>
<td>Honda</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drives (person, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jim</td>
</tr>
<tr>
<td>Mazda</td>
</tr>
<tr>
<td>Hank</td>
</tr>
<tr>
<td>Honda</td>
</tr>
</tbody>
</table>

(Cathy,Honda,Jim,Mazda) || (Cathy,Honda,Bill,Mazda) || (Cathy,Mazda,Jim,Mazda) || (Cathy,Mazda,Bill,Mazda)

(Cathy,Honda,Hank,Honda) || (Cathy,Mazda,Hank,Honda)
Operational Semantics: Example

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

```
<table>
<thead>
<tr>
<th>Saw (witness, car)</th>
<th>Drives (person, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cathy</td>
<td>Honda</td>
</tr>
<tr>
<td>Jim</td>
<td>Bill</td>
</tr>
<tr>
<td>Hank</td>
<td>Honda</td>
</tr>
</tbody>
</table>

(Cathy,Honda,Jim,Mazda) || (Cathy,Honda,Bill,Mazda) || (Cathy,Mazda,Jim,Mazda) || (Cathy,Mazda,Bill,Mazda)

(Cathy,Honda,Hank,Honda) || (Cathy,Mazda,Hank,Honda)
Operational Semantics: Example

\[\text{SELECT person} \]
\[\text{FROM Saw, Drives} \]
\[\text{WHERE Saw.car} = \text{Drives.car} \]

<table>
<thead>
<tr>
<th>Saw (witness, car)</th>
<th>Drives (person, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cathy</td>
<td>Honda (\parallel) Mazda</td>
</tr>
<tr>
<td></td>
<td>Jim (\parallel) Bill Mazda</td>
</tr>
<tr>
<td></td>
<td>Hank Mazda Honda</td>
</tr>
</tbody>
</table>

(Cathy,Honda,Jim,Mazda) \(\parallel\) (Cathy,Honda,Bill,Mazda) \(\parallel\) (Cathy,Mazda,Jim,Mazda) \(\parallel\) (Cathy,Mazda,Bill,Mazda)

(Cathy,Honda,Hank,Honda) \(\parallel\) (Cathy,Mazda,Hank,Honda)
CREATE TABLE Suspects AS
SELECT Drives.person
FROM Saw, Drives
WHERE Saw.car = Drives.car

<table>
<thead>
<tr>
<th>Saw (witness, car)</th>
<th>Drives (person, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cathy</td>
<td>Honda</td>
</tr>
<tr>
<td></td>
<td>Jim</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Suspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jim</td>
</tr>
<tr>
<td>Hank</td>
</tr>
</tbody>
</table>

? λ( ) = …
? λ( ) = …
Confidences supplied with base data

Trio computes confidences on query results
- Default probabilistic interpretation
- Can choose to plug in different arithmetic

<table>
<thead>
<tr>
<th>Saw (witness, car)</th>
<th>Drives (person, car)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cathy</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Honda</strong> 0.6</td>
</tr>
<tr>
<td></td>
<td><strong>Mazda</strong> 0.4</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Jim</strong> 0.3</td>
</tr>
<tr>
<td></td>
<td><strong>Bill</strong> 0.6</td>
</tr>
<tr>
<td></td>
<td><strong>Mazda</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Hank</strong> 1.0</td>
</tr>
<tr>
<td></td>
<td><strong>Honda</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Suspects</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Jim</strong> 0.3</td>
</tr>
<tr>
<td></td>
<td><strong>Bill</strong> 0.4</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Hank</strong> 0.6</td>
</tr>
</tbody>
</table>
Additional TriQL Constructs

- “Horizontal subqueries”
  Refer to tuple alternatives as a relation
- Aggregations: low, high, expected
- Unmerged (horizontal duplicates)
- Flatten, GroupAlts
- NoLineage, NoConf, NoMaybe
- Query-computed confidences
- Data modification statements
Trio-Specific Additional Features

- Lineage tracing
- On-demand confidence computation
- Coexistence checks
- Extraneous data removal

→ Interrelated algorithms
One More Example Query

<table>
<thead>
<tr>
<th>PrimeSuspect (crime#, suspect, accuser)</th>
<th>Credibility (person, score)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Jimmy, Amy</td>
<td></td>
</tr>
<tr>
<td>2 Frank, Cathy</td>
<td></td>
</tr>
</tbody>
</table>

List suspects with conf values based on accuser credibility

<table>
<thead>
<tr>
<th>Suspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jimmy 0.33</td>
</tr>
<tr>
<td>Frank 0.25</td>
</tr>
</tbody>
</table>
### One More Example Query

#### PrimeSuspect (crime#, suspect, accuser)

|   | Jimmy, Amy || Billy, Betty || Hank, Cathy |
|---|-------------|----------------|--------------|
| 1 |             |                |              |
| 2 | Frank, Cathy || Freddy, Betty |

#### Credibility (person, score)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Betty</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Cathy</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{SELECT suspect, score}/[\text{sum(score)}] \text{ as conf} \\
\text{FROM (SELECT suspect,} \\
\text{(SELECT score FROM Credibility C} \\
\text{WHERE C.person = P.accuser) \\
\text{FROM PrimeSuspect P})}
\]

### Suspects

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Jimmy</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>Frank</td>
<td>0.25</td>
<td></td>
</tr>
</tbody>
</table>
The Trio System

Version 1 (“Trio-One”)

On top of standard DBMS

Surprisingly easy and complete, reasonably efficient
Trio-One Overview

- Standard relational DBMS
- Trio API and translator (Python)
- Command-line client
- TrioExplorer (GUI client)

**Trio API and translator (Python)**

- Standard SQL

**Standard relational DBMS**

- Encoded Data Tables
- Trio Metadata
- Lineage Tables
- Trio Stored Procedures

**TrioExplorer (GUI client)**

- DDL commands
- TriQL queries
- Schema browsing
- Table browsing
- Explore lineage
- On-demand confidence computation

**Command-line client**

- Partition and “verticalize”
- Shared IDs for alternatives
- Columns for confidence, “?”

**Metadata**

- One per result table
- Uses unique IDs
- Encodes formulas

**Lineage**

- Table types
- Schema-level lineage structure
- Conf()
- Lineage()
Example: Data Encoding

**Saw (witness, color, car)**

| Amy   | red, Honda 0.3 || red, Toyota 0.4 || orange, Mazda 0.3 |
|-------|-----------------|
| Betty | blue, Acura 0.8 |

**View:** \( \text{Saw} = \text{Saw-C} \bowtie \text{Saw-U} \)

**Saw-C**

<table>
<thead>
<tr>
<th>xid</th>
<th>witness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Amy</td>
</tr>
<tr>
<td>2</td>
<td>Betty</td>
</tr>
</tbody>
</table>

**Saw-U**

<table>
<thead>
<tr>
<th>xid</th>
<th>aid</th>
<th>conf</th>
<th>color</th>
<th>car</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>0.3</td>
<td>red</td>
<td>Honda</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>0.4</td>
<td>red</td>
<td>Toyota</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>0.3</td>
<td>orange</td>
<td>Mazda</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>0.8</td>
<td>blue</td>
<td>Acura</td>
</tr>
</tbody>
</table>
Query $Q$ into result table $R$

1. Run query $Q'$ to produce “super-result” $R-U$
   $Q' \approx Q$ but adds ID’s of source tuples, joins lineage tables when lineage() predicates, other tricks
2. Group $R-U$ into alternatives, generate xid’s
3. Move certain attrs. to $R-C$, lineage data to $R-Lin$
4. Compute confidences? (next slide)
5. Add metadata: view defn. for $R$, schemas, confidence info., lineage structure
6. Transient results: stop at 2, return cursor
Previous approach (probabilistic databases)

- Each operator computes confidences during query execution
- *Restricts allowable query execution strategies*

In Trio

Confidence of data element $d$ is function of confidences in $\lambda^*(d)$
Our approach

- Use any query execution strategy
- Compute confidences on-demand based on lineage
- Some optimizations
  - “Independent lineage subtrees”
  - Memoization
  - Batch computation
More forms of uncertainty
- Continuous uncertainty (intervals, Gaussians)
- Correlated uncertainty
- Incomplete relations

More forms of lineage
- External lineage
- Update lineage

Confidence-based queries
- Threshold; “Top-K”
Current Topics (sample)

Design theory
- Dependencies, normal forms, decomposition
- New definitions, twists, and challenges

System
- Full query language
- Performance experiments
- Demo applications
- Version 2: Go native?
  - Storage and indexing structures
  - Statistics
  - Query optimization
Search “stanford trio”

Trio contributors, past and present

Parag Agrawal, Omar Benjelloun, Ashok Chandra, Anish Das Sarma, Alon Halevy, Chris Hayworth, Ander de Keijzer, Raghotham Murthy, Michi Mutsuzaki, Shubha Nabar, Tomoe Sugihara, Martin Theobald