Probabilistic/Uncertain Data Management


Slides based on the Suciu/Dalvi SIGMOD’05 tutorial

Databases Today are Deterministic

- An item either is in the database or is not
  - Database represents a “complete world”
- A tuple either is in the query answer or is not
- This applies to all variety of data models:
  - Relational, E/R, NF2, hierarchical, XML, …

What is a Probabilistic Database?

- “An item belongs to the database” is a probabilistic event
  - Tuple-existence uncertainty
  - Attribute-value uncertainty
- “A tuple is an answer to the query” is a probabilistic event
- Can be extended to all data models; we discuss only probabilistic relational data

Two Types of Probabilistic Data

- Database is deterministic
  - Query answers are probabilistic
    - E.g., IR-style/“fuzzy-match” queries
    - Approximate query answers
- Database is probabilistic
  - Query answers are probabilistic

Long History

Probabilistic relational databases have been studied from the late 80’s until today:

- Cavallo&Pitarelli:1987
- Barbara,Garcia-Molina, Porter:1992
- Lakshmanan,Leone,Ross&Subrahmanian:1997
- Fuhr&Roellke:1997
- Dalvi&Suciu:2004
- Widom:2005

So, Why Now?

Application pull:
- The need to manage imprecisions in complex data and query-processing tasks

Technology push:
- Advances in query-processing tools/techniques
Application Pull
Need to manage imprecisions in data
- Many types: non-matching data values, imprecise queries, inconsistent data, misaligned schemas, etc.

The quest to manage imprecisions = major driving force in the database community
- Ultimate driver for many research areas: data mining, semistructured data, schema matching, NN queries

*Thesis: A large class of data imprecisions can be effectively modeled with probabilities*

Technology Push
Processing probabilistic data is *fundamentally more complex* than other data models
- Some previous approaches sidestepped complexity

There exists a rich collection of powerful, non-trivial techniques and results, some old, some very recent, that could lead to practical management techniques for probabilistic databases

Managing Imprecisions:
Applications
1. Ranking query answers
2. Record linkage
3. Quality in data integration
4. Inconsistent data / Data cleaning
5. Information disclosure

1. Ranking Query Answers
Database is deterministic

The query returns a *ranked list of tuples*
- Based on some application-specific *ranking function*
- User interested in top-k answers

The Empty Answers Problem
Query is overspecified: no answers
Example: try to buy a house in SF...

```
SELECT * 
FROM Houses 
WHERE bedrooms = 3 
AND style = 'craftsman' 
AND district = 'Noe Valley' 
AND price < 400000
```

... good luck!
Beyond a single table: “Find the good deals in a neighborhood!”

Users specify similarity predicates with ~
System combines atomic similarities using probabilities

Types of Similarity Predicates

• String edit distances:
  – Levenshtein distance, Q-gram distances
• TF-IDF scores
• Ontology distance / semantic similarity:
  – Wordnet
• Phonetic similarity:
  – SOUNDEX

Summary on Ranking Query Answers

Types of imprecision addressed:
Data is precise, query answers are imprecise:
• User has limited understanding of the data
• User has limited understanding of the schema
• User has personal preferences

Probabilistic approach would...
• Principled semantics for complex queries
• Integrate well with other types of imprecision

Application: Data Cleaning, ETL

• Merge/purge for large databases, by sorting and clustering
• Use of dimensional hierarchies in data warehouses and exploit co-occurrences
• Novel similarity functions that are amenable to indexing
• Declarative language to combine cleaning tasks

2. Record Linkage

Determine if two data records describe same object

Scenarios:
• Join/merge two relations
• Remove duplicates from a single relation
• Validate incoming tuples against a “reference set”

Keyword Searches in Databases

Goal:
• Users want to search via keywords
• Do not know the schema

Techniques:
• Matching objects may be scattered across physical tables due to normalization; need on the fly joins
• Score of a tuple = number of joins, plus “prestige” based on in-degree

Example SQL Query:
```
SELECT *
FROM Houses x
WHERE x.bedrooms ~ 3 AND x.style ~ 'craftsmen' AND x.price ~ 600k
AND NOT EXISTS
(SELECT *
FROM Houses y
WHERE x.district = y.district AND x.ID = y.ID
AND y.bedrooms ~ 3 AND y.style ~ 'craftsmen' AND y.price ~ 600k)
```


**Application: Data Integration**

**WHIRL**

- All attributes in all tables are of type *text*
- Datalog queries with two kinds of predicates:
  - Relational predicates
  - Similarity predicates \( X \sim Y \)

Matches two sets on the fly, but not really a "record linkage" application.

**Example 1:**

\[
Q_2((*) \sim P(Company_1, Industry_1), Q(Company_2, Website), R(Industry_2, Analysis), Company_1 \sim Company_2, Industry_1 \sim Industry_2)
\]

Score of an answer tuple = product of similarities

**Example 2 (with projection):**

\[
Q_2(Website) \sim P(Company_1, Industry_1), Q(Company_2, Website), R(Industry_2, Analysis), Company_1 \sim Company_2, Industry_1 \sim Industry_2
\]

Support(t) = set of tuples supporting the answer tuple

score(t) = 1 - \( \prod_{s \in \text{Support}(t)} (1 - \text{score}(s)) \)

**Summary on Record Linkage**

**Types of imprecision addressed:**
- Same entity represented in different ways
- Misspellings, lack of canonical representation, etc.

**A probability model would...**
- Allow system to use the match probabilities: cheaper, on-the-fly
- But need to model complex probabilistic correlations: is one set a reference set ("high-quality" items)? how many duplicates are expected?

**Other Applications**

- Data lineage + accuracy: Trio [Widom:2005]
- Sensor data [Deshpande, Guestrin, Madden:2004]
- Personal information management
  - Semex [Dong&Halevy:2005, Dong,Halevy,Madhavan:2005]
  - Heystack [Karger et al. 2003], Magnet [Sinha&Karger:2005]
- Using statistics to answer queries [Dalvi&Suciu:2005]

**Applications: Summary**

**Common in these applications:**
- Data in database and/or in query answer is uncertain, ranked; sometimes probabilistic

**Need for common probabilistic model**
- **Main benefit: uniform, principled approach to imprecision**
- Other benefits:
  - Handle complex queries (instead of single table TF/IDF)
  - Cheaper/better solutions through improved probabilistic techniques
Probabilistic Data Semantics

- The possible worlds model
- Query semantics

Possible Worlds Semantics

Attribute domains:
- int, char(30), varchar(55), datetime

Relational schema:
- Employee(name varchar(55), dob datetime, salary int)

Database schema:
- Employee(...), Projects(...), Groups(...), WorksFor(...)

# of instances: \( N \) (= BIG but finite)

The Definition

The set of all possible database instances:

\[ \text{INST} = \{ I_1, I_2, I_3, \ldots, I_N \} \]

Definition A probabilistic database \( \mathcal{P} \) is a probability distribution on \( \text{INST} \)

\[ \Pr : \text{INST} \rightarrow [0,1] \quad \text{s.t.} \quad \sum_{i=1}^{N} \Pr(I_i) = 1 \]

Definition A possible world is \( I \) s.t. \( \Pr(I) > 0 \)

Example

\[ \mathcal{P} = \{ I_1, I_2, I_3, I_4 \} \]

Query Semantics

Given a query \( Q \) and a probabilistic database \( \mathcal{P} \), what is the meaning of \( Q(\mathcal{P}) \)?

Tuples as Events

One tuple \( t \Rightarrow \) event \( t \in I \)

\[ \Pr(t) = \sum_{i=1}^{N} \Pr(I_i) \]

Two tuples \( t_1, t_2 \Rightarrow \) event \( t_1 \in I \land t_2 \in I \)

\[ \Pr(t_1, t_2) = \sum_{i=1}^{N} \Pr(I_i) \sum_{j=1}^{N} \Pr(I_j) \]
Query Semantics

Semantics 1: Possible Answers
A probability distribution on sets of tuples

\[ \forall A. \Pr(Q = A) = \sum_{I \in \text{INST. } Q} \Pr(I) \]

Semantics 2: Possible Tuples
A probability function on tuples

\[ \forall t. \Pr(t \in Q) = \sum_{I \in \text{INST. } Q} \Pr(I) \]

Example: Query Semantics

Possible-Worlds Semantics: Summary

Very powerful model

- **Complete:** Can capture any instance distribution, any tuple correlations

  Intuitive, clean formal semantics for any SQL query

  - Translates to queries over deterministic instances

Possible Worlds Semantics: Summary (contd.)

Not very useful as a representation or implementation tool

- HUGE number of possible worlds!

Need more effective representation formalisms

- Something that users can understand/explore

  - Allow more efficient query execution

    - Avoid "possible worlds explosion"

    - Perhaps giving up completeness

Possible Worlds Semantics: Summary (contd.)

Possible answers semantics

- Precise

  - Can be used to compose queries

  - Difficult user interface

Possible tuples semantics

- Less precise, but simple; sufficient for most apps

  - Cannot be used to compose queries

  - Simple user interface