OLAP over Imprecise Data with Domain Constraints

Doug Burdick
University of Wisconsin – Madison

Joint work with AnHai Doan (UW-Madison), Raghu Ramakrishnan (Yahoo! Research), Shivakumar Vaithyanathan (IBM Research at Almaden)
Traditional OLAP: Data Model

![Diagram of OLAP data model]

**FactID** | **Auto** | **Loc** | **Repair**  
---|---|---|---  
 p1 | F150 | Mad | 100  
 p2 | Sierra | Mad | 500  
 p3 | F150 | Mil | 100  
 p4 | Sierra | Mil | 200
Traditional OLAP: Queries

Auto = Truck
Loc = Mil
SUM(Repair) = ?
Answer: 300

<table>
<thead>
<tr>
<th>FactID</th>
<th>Auto</th>
<th>Loc</th>
<th>Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>F150</td>
<td>Mad</td>
<td>100</td>
</tr>
<tr>
<td>p2</td>
<td>Sierra</td>
<td>Mad</td>
<td>500</td>
</tr>
<tr>
<td>p3</td>
<td>F150</td>
<td>Mil</td>
<td>100</td>
</tr>
<tr>
<td>p4</td>
<td>Sierra</td>
<td>Mil</td>
<td>200</td>
</tr>
</tbody>
</table>
Querying Information Extracted from Text

<table>
<thead>
<tr>
<th>ID</th>
<th>Review Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>I love the reliability of my F150 from Zimbrick Ford in <strong>Milwaukee</strong>. Much better than my Sierra. Paid $30000 for a 4WD.</td>
</tr>
<tr>
<td>p2</td>
<td>My 5-speed Subaru Outback handles well in <strong>Wisconsin</strong> winters. Great value at $25000</td>
</tr>
<tr>
<td>p3</td>
<td>After my old car was totaled in the <strong>Madison</strong> flood, I bought a BMW 330. It’s at the mechanic’s all the time.</td>
</tr>
</tbody>
</table>

For each location, what is the average price for different cars?

<table>
<thead>
<tr>
<th>ID</th>
<th>Location</th>
<th>Model</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>Milwaukee</td>
<td>{F150, Sierra}</td>
<td>30000</td>
</tr>
<tr>
<td>p2</td>
<td>Wisconsin</td>
<td>Subaru Outback</td>
<td>25000</td>
</tr>
<tr>
<td>p3</td>
<td>Madison</td>
<td>BMW 330</td>
<td>330</td>
</tr>
</tbody>
</table>

In a dataset from a real-world application at IBM Almaden with 800,000 facts, 30% were imprecise
[VLDB 05] Proposed Solution: Allow Imprecise Facts

Table:

<table>
<thead>
<tr>
<th>FactID</th>
<th>Auto</th>
<th>Loc</th>
<th>Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>F150</td>
<td>Mad</td>
<td>100</td>
</tr>
<tr>
<td>p2</td>
<td>Sierra</td>
<td>Mad</td>
<td>500</td>
</tr>
<tr>
<td>p3</td>
<td>F150</td>
<td>Mil</td>
<td>100</td>
</tr>
<tr>
<td>p4</td>
<td>Sierra</td>
<td>Mil</td>
<td>200</td>
</tr>
<tr>
<td>p5</td>
<td>Truck</td>
<td>Mil</td>
<td>100</td>
</tr>
</tbody>
</table>
[VLDB 05] Problem: How to Query Imprecise Facts

Auto = F150
Loc = Mil
SUM(Repair) = ?

Answer: ?

<table>
<thead>
<tr>
<th>FactID</th>
<th>Auto</th>
<th>Loc</th>
<th>Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>F150</td>
<td>Mad</td>
<td>100</td>
</tr>
<tr>
<td>p2</td>
<td>Sierra</td>
<td>Mad</td>
<td>500</td>
</tr>
<tr>
<td>p3</td>
<td>F150</td>
<td>Mil</td>
<td>100</td>
</tr>
<tr>
<td>p4</td>
<td>Sierra</td>
<td>Mil</td>
<td>200</td>
</tr>
<tr>
<td>p5</td>
<td>Truck</td>
<td>Mil</td>
<td>100</td>
</tr>
</tbody>
</table>
[VLDB 05] Solution: Use possible worlds

Imprecise fact table D 

EDB D' 

Possible worlds 

$w_1$  

$w_2$  

$w_3$  

$w_4$  

Allocation

Query answer is **expected value** over possible worlds
[VLDB 05] Example

Imprecise Fact Table D

<table>
<thead>
<tr>
<th>FactID</th>
<th>Auto</th>
<th>Loc</th>
<th>Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>F150</td>
<td>Mad</td>
<td>100</td>
</tr>
<tr>
<td>p2</td>
<td>Sierra</td>
<td>Mad</td>
<td>500</td>
</tr>
<tr>
<td>p3</td>
<td>F150</td>
<td>Mil</td>
<td>100</td>
</tr>
<tr>
<td>p4</td>
<td>Sierra</td>
<td>Mil</td>
<td>200</td>
</tr>
<tr>
<td>p5</td>
<td>Truck</td>
<td>Mil</td>
<td>100</td>
</tr>
</tbody>
</table>

Extended Database D’

<table>
<thead>
<tr>
<th>ID</th>
<th>FactID</th>
<th>Auto</th>
<th>Loc</th>
<th>Repair</th>
<th>Alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p1</td>
<td>F150</td>
<td>Mad</td>
<td>100</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>p2</td>
<td>Sierra</td>
<td>Mad</td>
<td>500</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>p3</td>
<td>F150</td>
<td>Mil</td>
<td>100</td>
<td>1.0</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
<td>Sierra</td>
<td>Mil</td>
<td>200</td>
<td>1.0</td>
</tr>
<tr>
<td>5</td>
<td>p5</td>
<td>F150</td>
<td>Mil</td>
<td>100</td>
<td>0.6</td>
</tr>
<tr>
<td>6</td>
<td>p5</td>
<td>Sierra</td>
<td>Mil</td>
<td>100</td>
<td>0.4</td>
</tr>
</tbody>
</table>
[VLDB 05] Example

\[
P(w_1) = 0.6 \\
P(w_2) = 0.4
\]
Contributions [VLDB 05, VLDB 06]

- Formalize entire process
- Develop several allocation policies
- Show how to execute allocation efficiently
- Demonstrate how to answer queries efficiently

Assumes all imprecise facts are independent
Challenge: Incorporate Domain Constraints

<table>
<thead>
<tr>
<th>ID</th>
<th>Repair Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>F150, oil change, $100, WI, John Smith</td>
</tr>
<tr>
<td>r2</td>
<td>customer John Smith brought F150 to garage engine noise, WI, $250</td>
</tr>
<tr>
<td>r3</td>
<td>Madison, Honda, broken ex. pipe, Dells &amp; I-90, towed 25 miles, $130</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FactID</th>
<th>Loc</th>
<th>Auto</th>
<th>Name</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>Wisconsin</td>
<td>F150</td>
<td>John Smith</td>
<td>100</td>
</tr>
<tr>
<td>p2</td>
<td>Wisconsin</td>
<td>F150</td>
<td>John Smith</td>
<td>250</td>
</tr>
<tr>
<td>p3</td>
<td>Madison</td>
<td>Honda</td>
<td>Dells</td>
<td>130</td>
</tr>
<tr>
<td>p4</td>
<td>Dells</td>
<td>Honda</td>
<td>Madison</td>
<td>130</td>
</tr>
</tbody>
</table>

"Two facts with same person name and model must have same city"

"Exactly one of facts p3 or p4 exists"
Summary of Contributions

- Present constraint language $L$
  - Define both syntax of $L$ and semantics of answering queries with constraints defined in $L$

- Efficiently answer queries with constraints using a marginal database $D^*$

- Present algorithms to efficiently construct marginal database $D^*$
Constraint Language: Examples

- “Two facts with same person name and model must have same location”
  - $(r.\text{Name} = r'.\text{Name}) \land (r.\text{Auto} = r'.\text{Auto}) \rightarrow (r.\text{Loc} = r'.\text{Loc})$

- “Exactly one of facts p3 or p4 exists”
  - $\exists p3 \rightarrow \neg \exists p4$
  - $\exists p4 \rightarrow \neg \exists p3$

- “If the location for p1 is Madison, then p3 must exist (and p4 cannot exist)”
  - $(p1.\text{Loc} = \text{“Madison”}) \rightarrow \exists p3 \land \neg \exists p4$
Constraint Language: Syntax

- A constraint has form $A \Rightarrow B$ where $A, B$ are conjunctions of atoms.
- Atoms have form $[r.A \Theta c]$ or $[r.A \Theta r'.A]$ or $\exists(r), \neg\exists(r)$ where
  - $r, r'$ are either
    - specific fact IDs themselves
    - variables that bind to fact IDs in $D$
  - $r.A$ is the value of attribute $A$ of fact $r$.
  - $\Theta \in \{=, \neq, \leq, <, \geq, >\}$ is a comparison operator over the appropriate domain.
  - $c$ is a constant from $\text{dom}(A)$, and
  - $\exists(r)$ ($\neg\exists(r)$) is a predicate that holds if fact $r$ exists (cannot exist).
A possible world satisfying all constraints is **valid**

Query answer is expected value over **valid** possible worlds
Efficient Query Answering

- Can compute expected value over valid possible worlds in *single scan* of Marginal Database (MDB) D*

**Diagram:**
- Imprecise fact table D
  - Allocation
  - EDB D'
    - Possible worlds: W₁, W₂, W₃, W₄
  - Constraints C
    - W₂
    - Q
    - A

- Imprecise fact table D
  - Allocation
  - EDB D'
    - Allocation
    - Marginalization
    - MDB D*
      - Q
      - A
Constraint: \((r.\text{Model} = r'.\text{Model}) \rightarrow (r.\text{Loc} = r'.\text{Loc})\)

**EDB D’**

<table>
<thead>
<tr>
<th>FactID</th>
<th>Model</th>
<th>Loc</th>
<th>Cost</th>
<th>Alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>Cam</td>
<td>Mad</td>
<td>100</td>
<td>0.7</td>
</tr>
<tr>
<td>r1</td>
<td>Cam</td>
<td>Dells</td>
<td>100</td>
<td>0.3</td>
</tr>
<tr>
<td>r2</td>
<td>Cam</td>
<td>Mad</td>
<td>400</td>
<td>0.8</td>
</tr>
<tr>
<td>r2</td>
<td>Cam</td>
<td>Dells</td>
<td>400</td>
<td>0.2</td>
</tr>
</tbody>
</table>

**MDB D’**

<table>
<thead>
<tr>
<th>FactID</th>
<th>Model</th>
<th>Loc</th>
<th>Cost</th>
<th>Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>Cam</td>
<td>Mad</td>
<td>100</td>
<td>0.9</td>
</tr>
<tr>
<td>r1</td>
<td>Cam</td>
<td>Dells</td>
<td>100</td>
<td>0.1</td>
</tr>
<tr>
<td>r2</td>
<td>Cam</td>
<td>Mad</td>
<td>400</td>
<td>0.9</td>
</tr>
<tr>
<td>r2</td>
<td>Cam</td>
<td>Dells</td>
<td>400</td>
<td>0.1</td>
</tr>
</tbody>
</table>

\[ P(w1) = 0.56 \]
\[ P(w2) = 0.24 \]
\[ P(w3) = 0.56 \]
\[ P(w4) = 0.06 \]

\[ PN(w1) = 0.90 \]
\[ PN(w2) = 0 \]
\[ PN(w3) = 0 \]
\[ PN(w4) = 0.10 \]
Marginal Database (MDB) D*

- Let $D'$ be EDB obtained from imprecise fact table $D$
- Each claim in $D'$ has tuple $f_t$ with allocation weight $w_t$
- Let $W$ be set of valid possible worlds satisfying a given set of constraints $C$
- Let $m_t$ be the total probability of worlds in $W$ where $f_t$ is true.

- We refer to $m_t$ as the \textit{marginal} probability of $f_t$ and $(f_t, m_t)$ is a marginal tuple.

- Store all marginal tuples in \textit{marginal database} (MDB) $D^*$
Marginalization Algorithms

- Can process connected component in constraint hypergraph **independently**
Constraint Hypergraph: Example

Constraint:

\[(r.\text{Model} = r'.\text{Model}) \rightarrow (r.\text{Loc} = r'.\text{Loc})\]
Constraint Hypergraph: $G=(V,H)$

- **Nodes $V$:** For each fact $r$ in given imprecise database $D$, introduce a node to $V$

- **Hyperedges $H$:** For each minimal set of facts with a combination of completions violating a constraint, introduce a hyperedge to $H$
Experimental Setup

- Algorithms evaluated on several datasets
  - Real-world dataset: 798K facts, 4 dimensions
  - Used several synthetic datasets
    - Scalability (up to 3.2 million tuples)

- Constraint sets
  - Randomly generated several constraint sets of varying “complexity”
  - Develop suitable complexity metric
Performance

800K Facts

- Total Time
- GenerateComponents
- ProcessComponents

Best Fit (Total Time)
- Best Fit (GenComps)
- Best Fit (ProcComps)

Time (sec)

Total Bindings

0.0E+00  1.0E+07  2.0E+07
Performance

3200K Facts

- Total Time
- GenerateComponents
- ProcessComponents
- Best Fit (Total Time)
- Best Fit (GenComps)
- Best Fit (ProcComps)
Component Sizes

- 2--5
- 6--10
- 11--20

800K (Automotive)
Related work

- Imprecise data with constraints
  - MayBMS [Antova et al. 07]
  - Representing and Querying Correlated Tuples in Probabilistic Databases [Sen, Deshpande 07]
  - ConQuer [Fuxman et al 05]

- Probabilistic databases
  - Probabilistic Databases [Dalvi et al. 04]
  - TRIO system for uncertain data [Widom et al.05]

- OLAP
  - Constraints in OLAP [Hurtado et. al 02]
  - OLAP over Incomplete Data [Dyreson 96]
Summary

- We extend our framework for OLAP over imprecise data to support domain information.

- Eliminate the strong independence assumptions required earlier
  - Often violated in many applications (e.g., IE from text)

- First work we are aware of to consider OLAP aggregation queries over imprecise data in the presence of constraints
Discussion

- Pretty brute-force
- Fact Table => EDB, how?
- Other Queries: AVG, MIN, MAX
  - How to generate MDB?
- Expressiveness of Constraints
  - A => B (0.4) or C (0.6)
  - More complex distributional constraints on data