CS 595 - Hot topics in database systems:

Data Provenance

I. Database Provenance
I.1 Provenance Models and Systems

Boris Glavic

September 5, 2012
Outline

1. Provenance Models Primer
2. Why-Provenance
Provenance Model

Data Provenance: Which input data contributed to which output data?
Provenance Model

- Data Provenance: Which input data contributed to which output data?
- but ... what does “contributed to” really mean?
- Provenance model models “contribution”
Why more than one Provenance Model?

- Evolution - Improving on previous models?
Why more than one Provenance Model?

- **Evolution** - Improving on previous models?
- **Granularities** - Provenance for different granularities is different
Why more than one Provenance Model?

- **Evolution** - Improving on previous models?
- **Granularities** - Provenance for different granularities is different
- **Transformation Language** - E.g., subsets of SQL
Why more than one Provenance Model?

- **Evolution** - Improving on previous models?
- **Granularities** - Provenance for different granularities is different
- **Transformation Language** - E.g., subsets of SQL
- **Set/Bag semantics**
Why more than one Provenance Model?

- **Evolution** - Improving on previous models?
- **Granularities** - Provenance for different granularities is different
- **Transformation Language** - E.g., subsets of SQL
- **Set/Bag semantics**
- **Computation/Size**
  - How large is Provenance?
  - How hard to compute?
Outline

1. Provenance Models Primer

2. Why-Provenance
   - Preliminaries
   - Witnesses for Tuples
   - Proof-Witnesses and Why-Provenance
   - Minimal Why-Provenance
   - Deletion Propagation Revisited
   - Recap
Provenance Models Primer

Preliminaries

Database Schema

- **Database schema** $S$: Set of relation schemata
  - $S = \{ R_1, \ldots, R_n \}$

- **Relation schema** $R$: Name + list of attribute name - domain pairs
  - $R(A_1 : D_1, \ldots, A_n : D_n)$
  - $R = \text{Relation name}$
  - $A_i = \text{Attribute name}$
  - $D_i = \text{Domain name}$
Database Schema

- **Database schema** $S$: Set of relation schemata
  - $S = \{R_1, \ldots, R_n\}$

- **Relation schema** $R$: Name + list of attribute name - domain pairs
  - $R(A_1 : D_1, \ldots, A_n : D_n)$
  - $R = \text{Relation name}$
  - $A_i = \text{Attribute name}$
  - $D_i = \text{Domain name}$

**Example**

PersonDB = \{Person, Address\}

Person(Name : String, AddrId : Int)

Address(Id : Int, City : String, Street : String)
**Database Instance**

- **Database Instance** $I$: Set of relations confirming to schema $S$
  - One relation instance for each relation schema in $S$
- **Relation Instance** $R$: Set of tuples confirming to relation schema $R$
- **Tuple** $t$: List of values
  - $(d_1, \ldots, d_n)$ with $\forall i \in \{1, \ldots, n\}: d_i \in D_i$
  - $\Rightarrow d_i$ is an element from domain $D_i$

**Example**

<table>
<thead>
<tr>
<th>Person</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>AddrId</td>
</tr>
<tr>
<td>$p_1$</td>
<td>Peter</td>
</tr>
<tr>
<td>$p_2$</td>
<td>Alice</td>
</tr>
<tr>
<td>$p_3$</td>
<td>Heinz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>City</th>
<th>Street</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>Chicago</td>
<td>51st</td>
</tr>
<tr>
<td>a2</td>
<td>Evanston</td>
<td>10th</td>
</tr>
</tbody>
</table>
Query Evaluation

Query

- Expression $q$ in some query language (e.g., SQL)
  - Relation schemata $\rightarrow$ (result) relation schema
- Defined for one or more schemata:
  - Only accesses relations, attributes from schema

Query Evaluation

- $Q(I)$:
  - Evaluating query $q$ for schema $S$
  - Over some instance $I$ for schema $S$
  - use $Q$ if $I$ clear
Query Evaluation

Example

Query $q$: $\pi_{\text{name}}(\text{Person})$

<table>
<thead>
<tr>
<th>Person</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>AddrId</td>
<td></td>
</tr>
<tr>
<td>$p_1$</td>
<td>Peter</td>
<td>1</td>
</tr>
<tr>
<td>$p_2$</td>
<td>Alice</td>
<td>1</td>
</tr>
<tr>
<td>$p_3$</td>
<td>Heinz</td>
<td>2</td>
</tr>
</tbody>
</table>

$Q(I)$

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
</tr>
<tr>
<td>$t_2$</td>
</tr>
<tr>
<td>$t_3$</td>
</tr>
</tbody>
</table>
Witnesses for Tuples

Why-Provenance

Rationale

- Models which input tuples are sufficient to derive an output tuple $t$ of query $Q$
Why-Provenance

Rationale
- Models which input tuples are sufficient to derive an output tuple $t$ of query $Q$

Provenance Representation
- A set of witnesses
- A witness $w$ is set of tuples
- $\Rightarrow$ Set-semantics
Witness

- **Witness** $w$ for a tuple $t$ in query result $Q(I)$
  - Intuitively: Sufficient set of tuples to derive $t$ using $q$

**Definition (Witness)**

$w$ is a witness for a tuple $t$ in a query result $Q(I)$ iff:

1. $w \subseteq I$: Subset of all tuples in $I$
2. $t \in Q(w)$: Tuple in result of evaluating query over $w$
Witness Example

Example

```sql
SELECT shop, price
FROM sales, items
WHERE itemId = id
```

<table>
<thead>
<tr>
<th>sales</th>
<th>items</th>
<th>Q(I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>shop</td>
<td>itemID</td>
<td>id</td>
</tr>
<tr>
<td>Migros</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Migros</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Coop</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Witness Example

**Example**

- Witnesses for $t_1$
- $w_1 = \{s_1, i_1\}$: Is witness: $Q(w_1) = \{t_1\}$
- $w_2 = I$: Is witness: $Q(w_2) = Q(I)$
- $w_3 = \{s_1, i_1, i_3\}$: Is witness: $Q(w_3) = \{t_1\}$
- $w_4 = \{s_2, i_1, i_3\}$: No witness: $Q(w_3) = \{t_2\}$
Set of Witnesses

Given a tuple \( t \) in result of a query \( q \) over instance \( I \)

Set of witnesses for \( t \): \( \text{Wit}(q, t, I) \)

Use \( \text{Wit}(q, t) \) or \( \text{Wit}(t) \) if other parameters fixed

Definition (Witness set)

\[ \text{Wit}(q, t, I) = \{ w \mid w \text{ is witness of } t \text{ for } Q(I) \} \]
**Set of Witnesses**

**Example**

\[
\text{Why}(q, t_1, I) = \{w_1, \ldots, w_n\}
\]

\[
w_1 = \{s_1, i_1\}
\]

\[
w_2 = \{s_1, s_2, i_1\}
\]

\[
w_3 = \{s_1, i_1, i_2\}
\]

\[
w_4 = \{s_1, i_1, i_3\}
\]

\[
w_n = I
\]

<table>
<thead>
<tr>
<th>sales</th>
<th>items</th>
<th>(Q(I))</th>
</tr>
</thead>
<tbody>
<tr>
<td>shop</td>
<td>itemId</td>
<td>shop</td>
</tr>
<tr>
<td>s_1</td>
<td>Migros</td>
<td>Migros</td>
</tr>
<tr>
<td>s_2</td>
<td>Migros</td>
<td>Migros</td>
</tr>
<tr>
<td>s_3</td>
<td>Coop</td>
<td>Coop</td>
</tr>
<tr>
<td>id</td>
<td>price</td>
<td>t_1</td>
</tr>
<tr>
<td>i_1</td>
<td>1</td>
<td>t_2</td>
</tr>
<tr>
<td>i_2</td>
<td>2</td>
<td>t_3</td>
</tr>
<tr>
<td>i_3</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Witnesses for Tuples

Properties

1. Instance is trivial witness
   - $I \in Wit(q, t, I)$ for any $q$ and $t$

2. Superset of witness is also witness
   - $w \in Wit(q, t, I) \Rightarrow \forall I' \subset I : (w \cup I') \in Wit(q, t, I)$
   - Only for positive operators!
   - $\Rightarrow$ Irrelevant tuples

3. Independent of query language (query = black box)
Provenance Models Primer

Witnesses for Tuples

Space Complexity

- For positive operators:
  - Given a witness \( w \)
  - Can create new witnesses by deciding for each tuple in \( I - w \) whether it should be in the witness
  - \( \Rightarrow 2^{|I - w|} - 1 \) additional witnesses of size up to \( |I| \)
  - \( \Rightarrow O(2^{|I|}) \) witnesses of size up
Witnesses for Tuples

Space Complexity

Example

Instance $I = \{ t_1, t_2, t_3, t_4 \}$
Witness $w = \{ t_1, t_2 \}$

$I - w = \{ t_3, t_4 \}$ with $\| I - w \| = 2$

Either include $t_3$ or $t_4$ or both

$3 = 2\| I - w \| - 1$ choices

$w' = \{ t_1, t_2, t_3 \}$  \hspace{1cm} w'' = \{ t_1, t_2, t_4 \}$  \hspace{1cm} w''' = \{ t_1, t_2, t_3, t_4 \}$
Why

Witnesses for Tuples

Time Complexity

How to compute?

- Formalism gives no hint
- Straight forward approach:
  - For each subset $I'$ of $I$
  - Compute $Q(I')$
  - Includes $t \Rightarrow I'$ is witness
Time Complexity

How to compute?

- Formalism gives no hint
- Straight forward approach:
  - For each subset $I'$ of $I$
  - Compute $Q(I')$
  - Includes $t \Rightarrow I'$ is witness

Complexity

- Number of subsets of $I$ times compute $q$
  - $2^{|I'|}$ subsets
  - Complexity of $Q(I')$: polynomial in $I$
Witnesses for Tuples

Time Complexity

How to compute?
- Formalism gives no hint
- Straight forward approach:
  - For each subset $I'$ of $I$
  - Compute $Q(I')$
  - Includes $t \Rightarrow I'$ is witness

Complexity
- Number of subsets of $I$ times compute $q$
  - $2^\|I\|)$ subsets
  - Complexity of $Q(I')$: polynomial in $I$
- $O(2^\|I\|)$
Time Complexity

How to compute?
- Formalism gives no hint
- Straight forward approach:
  - For each subset $I'$ of $I$
  - Compute $Q(I')$
  - Includes $t \Rightarrow I'$ is witness
- If positive ops $\Rightarrow$ avoid computing $Q(I')$

Complexity
- Number of subsets of $I$ times compute $q$
  - $2^{\|I'\|}$ subsets
  - Complexity of $Q(I')$: polynomial in $I$
- $\Rightarrow O(2^{\|I'\|})$
Proof-Witness

Rationale

- $Wit(q, t, I)$ contains witnesses with irrelevant tuples
- Define witnesses based on query expression
- $\Rightarrow$ Hope to avoid irrelevant tuples
- Include one tuple from each leaf of the alegra tree of $q$

Example

<table>
<thead>
<tr>
<th>shop</th>
<th>itemId</th>
</tr>
</thead>
<tbody>
<tr>
<td>s₁</td>
<td>Migros</td>
</tr>
<tr>
<td>s₂</td>
<td>Migros</td>
</tr>
<tr>
<td>s₃</td>
<td>Coop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>i₁</td>
<td>100</td>
</tr>
<tr>
<td>i₂</td>
<td>10</td>
</tr>
<tr>
<td>i₃</td>
<td>25</td>
</tr>
</tbody>
</table>
Proof-Witness

- A proof-witness
  - contains one tuple from each algebra tree leaf of $q$
  - definition dependent on query expression (syntax)
Why-Provenance

- Why\((q, t, I)\) is set of all proof-witnesses for \(t\)
- Recursive compositional definition for algebra operators:
  - Accessing an relation \(R\) ⇒ Base case
  - Selection \(\sigma_C\)
  - Projection \(\pi_A\)
  - Join \(\bowtie_C\)
  - Union \(\cup\)
Why-Provenance

- $\text{Why}(q, t, I)$ is set of all proof-witnesses for $t$
- Recursive compositional definition for algebra operators:

**Definition**

\[
\begin{align*}
\text{Why}(R, t, I) &= \{\{t\}\} \\
\text{Why}(\sigma_C(q), t, I) &= \text{Why}(q, t, I) \\
\text{Why}(\pi_A(q), t, I) &= \bigcup_{u \in Q(I) : u.A=t} \text{Why}(q, u, I) \\
\text{Why}(q_1 \bowtie_C q_2, t, I) &= \{(w_1 \cup w_2) \mid w_1 \in \text{Why}(q_1, t_1, I) \land t_1 = t.Q_1 \land w_2 \in \text{Why}(q_2, t_2, I) \land t_2 = t.Q_2\} \\
\text{Why}(q_1 \cup q_2, t, I) &= \text{Why}(q_1, t, I) \cup \text{Why}(q_2, t, I)
\end{align*}
\]
Example Why-provenance computation

\[ q = \pi_{\text{shop, price}}(sales \bowtie_{\text{itemId} = \text{id}} items) \]

**Example**

```
SELECT shop, price
FROM sales, items
WHERE itemId = id
```

<table>
<thead>
<tr>
<th>sales</th>
<th>items</th>
</tr>
</thead>
<tbody>
<tr>
<td>shop</td>
<td>itemId</td>
</tr>
<tr>
<td>s₁</td>
<td>Migros 1</td>
</tr>
<tr>
<td>s₂</td>
<td>Migros 3</td>
</tr>
<tr>
<td>s₃</td>
<td>Coop 3</td>
</tr>
<tr>
<td>id</td>
<td>price</td>
</tr>
<tr>
<td>i₁</td>
<td>100</td>
</tr>
<tr>
<td>i₂</td>
<td>10</td>
</tr>
<tr>
<td>i₃</td>
<td>25</td>
</tr>
</tbody>
</table>
Example Why-provenance computation

\[ q = \pi_{\text{shop}, \text{price}}(q_1) \]

\[ q_1 = \text{sales} \bowtie_{\text{itemId} = \text{id}} \text{items} \]

<table>
<thead>
<tr>
<th>shop</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>Migros</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>Migros</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>Coop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>shop</th>
<th>itemId</th>
<th>id</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j_1 )</td>
<td>Migros</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>( j_2 )</td>
<td>Migros</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>( j_3 )</td>
<td>Coop</td>
<td>3</td>
<td>25</td>
</tr>
</tbody>
</table>
Example Why-provenance computation

**Compute** $Why(q, t_1, l)$: 1) Substitute definitions

$Why(q, t) = \bigcup_{u \in Q_1(l) : u.(\text{shop}, \text{price}) = t} Why(q_1, u)$

$Why(q_1, t) = \{(w_1 \cup w_2) \mid w_1 \in Why(sales, t_1) \wedge t_1 = t.sales \wedge w_2 \in Why(items, t_2) \wedge t_2 = t.items\}$

$Why(sales, t) = \{\{t\}\}$

$Why(sales, t) = \{\{t\}\}$
Example Why-provenance computation

**Example**

**Compute** $Why(q, t_1, l)$: 2) Evaluate for tuple top-down - 1

$$\Rightarrow Why(q, t_1) = Why(q_1, j_1)$$

$$Why(q_1, t) = \{ (w_1 \cup w_2) | w_1 \in Why(sales, t_1)$$

$$\land t_1 = t.sales \land w_2 \in Why(items, t_2)$$

$$\land t_2 = t.items \}$$

$$Why(sales, t) = \{ \{ t \} \}$$

$$Why(sales, t) = \{ \{ t \} \}$$
Example Why-provenance computation

Compute \( \text{Why}(q, t_1, I) \):

3) Evaluate for tuple top-down - 2

\[
\text{Why}(q, t_1) = \text{Why}(q_1, j_1)
\]

\[
\Rightarrow \text{Why}(q_1, j_1) = \{(w_1 \cup w_2) \mid w_1 \in \text{Why}(sales, s_1) \wedge w_2 \in \text{Why}(items, i_1)\}
\]

\[
\text{Why}(sales, t) = \{\{t\}\}
\]

\[
\text{Why}(sales, t) = \{\{t\}\}
\]
Example Why-provenance computation

**Compute** $Why(q, t_1, l)$:

1. $Why(q, t_1) = Why(q_1, j_1)$
2. $Why(q_1, j_1) = \{(w_1 \cup w_2) | w_1 \in Why(sales, s_1) \land w_2 \in Why(items, i_1)\}$
3. $\Rightarrow Why(items, i_1) = \{\{i_1\}\}$
4. $\Rightarrow Why(sales, s_1) = \{\{s_1\}\}$
Example Why-provenance computation

**Compute** \(\text{Why}(q, t_1, I)\):

5) Substitute results bottom-up - 1

\[
\text{Why}(q, t_1) = \text{Why}(q_1, j_1)
\]

\[
\Rightarrow \text{Why}(q_1, j_1) = \{\{s_1, i_1\}\}
\]

\[
\text{Why}(\text{items}, i_1) = \{\{i_1\}\}
\]

\[
\text{Why}(\text{sales}, s_1) = \{\{s_1\}\}
\]
Example Why-provenance computation

Compute $Why(q, t_1, l)$:

6) Substitute results bottom-up - 2

$\Rightarrow Why(q, t_1) = \{s_1, i_1\}$

$Why(q_1, j_1) = \{s_1, i_1\}$

$Why(items, i_1) = \{i_1\}$

$Why(sales, s_1) = \{s_1\}$
Excursion: Query Equivalence

- Queries are equivalent if
  - Produce same result for all possible instances

**Definition**

Query equivalence Two queries $q_1$ and $q_2$ are equivalent ($q_1 \equiv q_2$) iff

- $\forall I : Q_1(I) = Q_2(I)$

**Example**

- $q_1 = R \cup S$
- $q_2 = S \cup R$
Insensitivity to Query Rewrite

What is that?

- Equivalent queries have same black-box behaviour
- Should have same provenance?

Definition (Insensitivity to Query Rewrite)

A provenance model $\mathcal{P}$ is called \textit{insensitive to query rewrite} iff

- $q_1 \equiv q_2 \Rightarrow \mathcal{P}(q_1, t) = \mathcal{P}(q_2, t)$
Insensitivity for $\text{Wit}$ and $\text{Why}$

$\text{Wit}(q, t)$
- Is insensitive
- Follows from the definition

$\text{Why}(q, t)$
Not insensitive:
- Counterexample
Insensitivity for *Wit* and *Why*

**Example**

\[
q_1 = \pi_a(R) \\
\text{Why}(q_1, t_1) = \{\{r_1\}\} \\
\text{Why}(q_1, t_2) = \{\{r_2\}\}
\]

\[
q_2 = \pi_a(R \bowtie_{b=c} \pi_{b\rightarrow c}(R)) \\
\text{Why}(q_2, t_1) = \{\{r_1\}, \{r_1, r_2\}\} \\
\text{Why}(q_2, t_2) = \{\{r_2\}, \{r_1, r_2\}\}
\]
Properties

- A proof-witness is also a witness
  - \( \Rightarrow \) \( \text{Why}(q, t, I) \subseteq \text{Wit}(q, t, I) \)
  - **Proof**: Induction over the structure of a query

- Only for operators we defined a rule!

- Less redundancy

- Sensitive to query rewrite

- Computation efficient: Brute force approach
  - Store all intermediate results
  - Recursive implementation of rules
  - Trace back one step at a time
Space Complexity

- Limited by the number of algebra tree leafs (relations)
- $R_1, \ldots, R_n$ relations
- $\Rightarrow R_1 \times \ldots \times R_n$
- In practise usually much smaller!
  - E.g., join on foreign key $\Rightarrow$ one witness list in provenance
Time Complexity

Straight forward implementation

- Compute all intermediate results
  - $\times$ operators in query
  - Each operator polynomial in instance size
- Each tracing step can be expressed as query over intermediate result
  - $\Rightarrow$ polynomial in instance size
- $\Rightarrow$ polynomial in instance size
Mineral Why-Provenance

Rationale

- Make Why-provenance insensitive to query rewrite
- Without losing positive properties
  - Computable (within reasonable time bounds)
  - Size
Minimal Why-Provenance

Minimal Elements of a Set

Definition

Minimal Elements

- Given set of sets $S$
- Minimal element: does not contain in other elements
- Element $e \in S$ is minimal iff $\nexists e' \in S : e' \subset e$

Example

- $S = \{\{a, b, c\}, \{a, b\}, \{a, c\}\}$
- $\{a, b, c\}$ is not minimal, e.g., $\{a, b\}$ contained
- $\{a, b\}$ is minimal, no other elements contained in $\{a, b\}$
Minimal Why-Provenance

- Minimal elements of Why-Provenance
- \( \Rightarrow \) removes redundancy

**Definition (Minimal Why-Provenance)**

\[
M_{Why}(q, t, I) = \{ w | w \in Why(q, t, I) \\
\land \nexists w' \in Why(q, t, I) : w' \subset w \}
\]
Minimal Why-provenance example

Example

\[ q_1 = \pi_a(R) \]
\[ Why(q_1, t_1) = \{\{r_1\}\} \]
\[ MWwhy(q_1, t_1) = \{\{r_1\}\} \]

\[ q_2 = \pi_a(R \bowt_{b=c} \pi_{b\rightarrow c}(R)) \]
\[ Why(q_2, t_1) = \{\{r_1\}, \{r_1, r_2\}\} \]
\[ MWwhy(q_2, t_1) = \{\{r_1\}\} \]
Minimal Why-Provenance

Alternative Definition

- Use $Wit(q, t, I)$ instead of $Why(q, t, I)$
- $\Rightarrow$ Same results?

Definition ($MWit(q, t, I)$)

$$MWit(q, t, I) = \{ w \mid w \in Wit(q, t, I) \land \not\exists w' \in Wit(q, t, I) : w' \subset w \}$$
Equivalence of $M\text{Why}(q, t, I) = M\text{Wit}(q, t, I)$

$M\text{Wit}(q, t, I) \subseteq M\text{Why}(q, t, I)$

1. $w \in \text{Wit}(q, t, I) \Rightarrow \exists w' \subseteq w : w' \in M\text{Why}(q, t, I)$
   - **Proof**: Induction over operators in $q$

2. $M\text{Why}(q, t, I) \subseteq \text{Wit}(q, t, I)$
   - $w \in M\text{Wit}(q, t, I)$
   - $\Rightarrow \exists w' \subseteq w : w' \in M\text{Wit}(q, t, I)$ (using 1)
   - $\Rightarrow w' \in \text{Wit}(q, t, I)$ (using 2)
   - $\Rightarrow w' = w$ (because $w$ is minimal)
The equivalence of $MWhy(q, t, I) = MWit(q, t, I)$ is proven as follows:

$MWit(q, t, I) \subseteq MWhy(q, t, I)$

- Every witness in $MWit(q, t, I)$ is also in $MWhy(q, t, I)$
- Take one witness $w \in MWit(q, t, I)$
  - $\Rightarrow w \in Why(q, t, I)$ ($MWit(q, t, I) \subseteq Why(q, t, I)$)
  - $\Rightarrow w \in MWhy(q, t, I)$
  - Assume: $w \notin MWhy(q, t, I)$
  - $\Rightarrow \exists w' \subset w : w' \in MWhy(q, t, I)$
  - $\Rightarrow w' \in Wit(q, t, I)$ ($Why(q, t, I) \subseteq Wit(q, t, I)$)
  - $\Rightarrow w$ not minimal Contradiction!
Provenance Models Primer

Minimal Why-Provenance

Equivalence of $\text{MWhy}(q, t, I) = \text{MWit}(q, t, I)$

$\text{MWhy}(q, t, I) \subseteq \text{MWit}(q, t, I)$

- $w \in \text{MWhy}(q, t, I)$
- $\Rightarrow w \in \text{Why}(q, t, I) \Rightarrow w \in \text{Wit}(q, t, I)$
- Suppose $\exists w' \subseteq w : w' \in \text{Wit}(q, t, I)$
- $\Rightarrow \exists w'' \subseteq w' : w'' \in \text{Why}(q, t, I)$
- $\Rightarrow w = w' = w''$ (w is minimal in $\text{Why}(q, t, I)$)
Equivalence of $MWhy(q, t, I) = MWit(q, t, I)$

Holds because we have shown that

- $MWhy(q, t, I) \subseteq MWit(q, t, I)$
- $MWit(q, t, I) \subseteq MWhy(q, t, I)$
Insensitivity to Query Rewrite

- $MWit(q, t, I)$ is insensitive
  - Same argument as for $Wit(q, t, I)$
  - $\Rightarrow$ Condition on black-box behaviour
- $MWhy(q, t, I) = MWit(q, t, I)$
- $\Rightarrow MWhy(q, t, I)$ is insensitive
Provenance Models Primer

Minimal Why-Provenance

Properties

- Insensitive to query rewrite
  - Side-effect: Remove redundancy
- Small size
- $MWhy$
  - Computation efficient: Compute $Why +$ containment checks
  - Limited to defined rules
- $MWit$
  - Independent of query language
  - Computation not straight-forwards
Space Complexity

- At most $Why(q, t, I)$
- $\Rightarrow$ polynomial in $\|I\|$
Time Complexity

- **Approach**
  - Compute $\text{Why}(q, t, I)$
  - Pairwise containment checks to find minimal elements
- Polynomial in size of instance $I$
Deletion Propagation

- Given a view
- How to update the view when input tuples are deleted
Deletion Propagation

- Given a view
- How to update the view when input tuples are deleted
  - ⇒ Use Why-provenance
Deletion Propagation using Why

**Approach**

- Store Why-provenance for each tuple in view $v$
- Upon deletion:
  - Adapt provenance
  - Delete tuples from view with empty provenance
- Set of tuples $D$ that got deleted
- For each tuple $t$ in view
  - Remove witnesses from $Why(t, v, I)$ that contain tuples from $D$
Deletion Propagation Example

CREATE VIEW ActiveCS AS
SELECT DISTINCT E.Name AS Emp
FROM Employee E, Project P, Assigned A
WHERE E.Id = A.Emp AND P.Name = A.Project
    AND Dep = CS

<table>
<thead>
<tr>
<th>Employee</th>
<th>Project</th>
<th>Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Name</td>
<td>Project</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>1</td>
<td>Peter</td>
<td>Server</td>
</tr>
<tr>
<td>2</td>
<td>Gertrud</td>
<td>Webpage</td>
</tr>
<tr>
<td>3</td>
<td>Michael</td>
<td>Fire CS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Dep</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Server</td>
<td>CS</td>
</tr>
<tr>
<td>2</td>
<td>Webpage</td>
<td>CS</td>
</tr>
<tr>
<td>3</td>
<td>Fire CS</td>
<td>HR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project</th>
<th>Emp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>1</td>
</tr>
<tr>
<td>Server</td>
<td>2</td>
</tr>
<tr>
<td>Webpage</td>
<td>2</td>
</tr>
<tr>
<td>Fire CS</td>
<td>3</td>
</tr>
</tbody>
</table>
Deletion Propagation Example

**Example**

<table>
<thead>
<tr>
<th>Employee</th>
<th>Id</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>1</td>
<td>Peter</td>
</tr>
<tr>
<td>e2</td>
<td>2</td>
<td>Gertrud</td>
</tr>
<tr>
<td>e2</td>
<td>3</td>
<td>Michael</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project</th>
<th>Name</th>
<th>Dep</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>Server</td>
<td>CS</td>
</tr>
<tr>
<td>p2</td>
<td>Webpage</td>
<td>CS</td>
</tr>
<tr>
<td>p3</td>
<td>Fire CS</td>
<td>HR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assigned</th>
<th>Project</th>
<th>Emp</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>Server</td>
<td>1</td>
</tr>
<tr>
<td>a2</td>
<td>Server</td>
<td>2</td>
</tr>
<tr>
<td>a3</td>
<td>Webpage</td>
<td>2</td>
</tr>
<tr>
<td>a4</td>
<td>Fire CS</td>
<td>3</td>
</tr>
</tbody>
</table>

**ActiveCS**

<table>
<thead>
<tr>
<th>Emp</th>
<th>t1</th>
<th>Peter</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>Gertrud</td>
<td></td>
</tr>
</tbody>
</table>
Deletion Propagation Example

- Delete tuple from Projects

<table>
<thead>
<tr>
<th>Employee</th>
<th>Project</th>
<th>Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Name</td>
<td>Project</td>
</tr>
<tr>
<td>e₁</td>
<td>1</td>
<td>Peter</td>
</tr>
<tr>
<td>e₂</td>
<td>2</td>
<td>Gertrud</td>
</tr>
<tr>
<td>e₂</td>
<td>3</td>
<td>Michael</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project</th>
<th>Dep</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁</td>
<td>Server</td>
</tr>
<tr>
<td>p₂</td>
<td>Webpage</td>
</tr>
<tr>
<td>p₃</td>
<td>Fire CS</td>
</tr>
<tr>
<td>a₁</td>
<td>1</td>
</tr>
<tr>
<td>a₂</td>
<td>2</td>
</tr>
<tr>
<td>a₃</td>
<td>2</td>
</tr>
<tr>
<td>a₄</td>
<td>3</td>
</tr>
</tbody>
</table>
Deletion Propagation Example

Example

- What would be the effect on the view?

ActiveCS

<table>
<thead>
<tr>
<th>Emp</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
</tr>
<tr>
<td>$t_2$</td>
</tr>
</tbody>
</table>

Employee

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>1 Peter</td>
</tr>
<tr>
<td>$e_2$</td>
<td>2 Gertrud</td>
</tr>
<tr>
<td>$e_2$</td>
<td>3 Michael</td>
</tr>
</tbody>
</table>

Project

<table>
<thead>
<tr>
<th>Name</th>
<th>Dep</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>Server CS</td>
</tr>
<tr>
<td>$p_2$</td>
<td>Webpage CS</td>
</tr>
<tr>
<td>$p_3$</td>
<td>Fire CS HR</td>
</tr>
</tbody>
</table>

Assigned

<table>
<thead>
<tr>
<th>Project</th>
<th>Emp</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>Server 1</td>
</tr>
<tr>
<td>$a_2$</td>
<td>Server 2</td>
</tr>
<tr>
<td>$a_3$</td>
<td>Webpage 2</td>
</tr>
<tr>
<td>$a_4$</td>
<td>Fire CS 3</td>
</tr>
</tbody>
</table>
Deletion Propagation - Approach

Assumption

- Assume we have Why-provenance for each tuple
  - $\text{Why}(t_1) = \{\{e_1, p_1, a_1\}\}$
  - $\text{Why}(t_2) = \{\{e_2, p_1, a_2\}, \{e_2, p_2, a_3\}\}$
- Set of deleted tuples ($D = \{p_1\}$)
Deletion Propagation Example

- \( Why(t_1) = \{e_1, p_1, a_1\} \rightarrow \{\} \)
- \( Why(t_2) = \{\{e_2, p_1, a_2\}, \{e_2, p_2, a_3\}\} \rightarrow \{\{e_2, p_2, a_3\}\} \)
Recap

**Flavours of Why-Provenance**
- Set of Witnesses
- Why-Provenance
- Minimal Why-Provenance

**Concepts**
- Insensitvity to Query Rewrite
- Query Equivalence
- Query language Independence
- Sufficiency
Recap

Literature