CS 595 - Hot topics in database systems: **Data Provenance**

I. Database Provenance

I.1 Provenance Models and Systems

Boris Glavic

September 22, 2012

Outline

1. Where-Provenance and DBNotes
   - Introduction
   - Where-Provenance Model
   - Insensitive Where-Provenance
   - Insensitive Where-Provenance for Queries With Union
   - DBNotes
   - Recap

Where-Provenance

Motivation

- Annotation Propagation
  - Given database, annotations on attribute values
  - How to annotate query results?
  - Alterative Goal: study Where values (data) are copied from

Rationale

- From which attribute values in the input is a target attribute value $t.A$ in the result of q copied from.
- Annotated DBs: add annotations based on annotations in Where-provenance

Provenance Representation

- Set of input attribute values (locations)

Annotated Databases

Perquisites

- Relational Model
- Cell (R, t, a): Relation R at tuple t and attribute a

Annotated Database

- Each cell (R, t, a) is associated with a set of annotations $\mathcal{A}(R, t, a)$ (Strings)
- For query results: $\mathcal{A}(Q(I), t, a)$

Example

Excursion - Query Containment

Query Equivalence

- Recall: $q \equiv q' \Leftrightarrow \forall I : Q(I) = Q'(I)$
- Queries have same results over same input

Query Containment

- $q \subseteq q'$: Query q contained in q'
- $q \subseteq q' \Leftrightarrow \forall I : Q(I) \subseteq Q(I')$
- $\Rightarrow$The result of q is contained in the result of q'
- Relationship to equivalence: $q \subseteq q' \wedge q' \subseteq q \Leftrightarrow q \equiv q'$

ILLINOIS INSTITUTE OF TECHNOLOGY

Annotation Containment

Annotation Containment

- $q \subseteq_{\mathcal{A}} q'$: Query q annotation contained in q' iff
  1. $q \subseteq q'$: q contained in q'
  2. $\forall I, t, A : \mathcal{A}(Q(I), t, A) \subseteq \mathcal{A}(Q'(I), t, A)$
- $\Rightarrow q$ only has tuples from q' with the same or some of the annotations

Annotation Equivalence

- $q =_{\mathcal{A}} q' : q \subseteq_{\mathcal{A}} q' \wedge q' \subseteq_{\mathcal{A}} q$
- Direct definition $\forall I, t, a$:
  1. $Q(I) = Q'(I)$
  2. $\mathcal{A}(Q(I), t, a) = \mathcal{A}(Q'(I), t, a)$

Annotation Example

EnzymeProduce

Enzyme	Gene
EC 1.1.1.1$^{\{a_1\}}$	4q11-q13$^{\{a_4\}}$
EC 1.97.1.6$^{\{a_3\}}$	4q11-q13$^{\{a_4\}}$

```
CREATE VIEW EnzymeProduce AS
SELECT Enzyme, Id AS Gene
FROM Gene G, Enzyme E
WHERE G.Id = E.ProducedBy
PROPAGATE G.Id TO Gene, E.Enzyme TO Enzyme
```

Gene

Id	Name
4q11-q13$^{\{a_4\}}$	ALB
18q21.3	BCL2

Enzyme

Enzyme	Weight	ProducedBy
EC 1.1.1.1$^{\{a_1\}}$	45	4q11-q13$^{\{a_2\}}$
EC 1.97.1.6$^{\{a_3\}}$	12	4q11-q13$^{\{a_2\}}$

Relationship Annotation Propagation and Where-Provenance

- Where-provenance one possible way to define annotation propagation

Relationship Annotation Propagation and Where-Provenance

- Where-provenance one possible way to define annotation propagation
- ⇒Output attribute value carries annotations from all input values in its provenance

Relationship Annotation Propagation and Where-Provenance

- Where-provenance one possible way to define annotation propagation
- ⇒Output attribute value carries annotations from all input values in its provenance
- ⇒Copying values ⇒ copying annotations

Relationship Annotation Propagation and Where-Provenance

- Where-provenance one possible way to define annotation propagation
- $\Rightarrow$Output attribute value carries annotations from all input values in its provenance
- $\Rightarrow$Copying values $\Rightarrow$ copying annotations
- What about functions or aggregation?
  - E.g., $sum(a)$
  - E.g., $f(a_1, a_2)$
- Not always what user wants
  - Allow user to specify how to propagate?

Outline

Where-Provenance

Properties

- **Granularity**: Attribute values
- **Representation**: Set of attribute locations: (R, t, a)
- **Definition**: Syntactic, recursive definition
  - Hard to come up with declarative definition!
- Sensitive to query rewrite
- $\Rightarrow$Insensitive declarative version

ILLINOIS INSTITUTE OF TECHNOLOGY

Definition (Where-Provenance)

1. $Where(R, t, a) = \mathcal{A}(R, t, a)$

Definition (Where-Provenance)

1. $Where(R, t, a) = \mathcal{A}(R, t, a)$
2. $Where(\sigma_C(q), t, a) = Where(q, t, a)$

Definition (Where-Provenance)

1. $Where(R, t, a) = \mathcal{A}(R, t, a)$
2. $Where(\sigma_C(q), t, a) = Where(q, t, a)$
3. $Where(\pi_A(q), t, b) = \bigcup_{u.A=t} Where(q, u, a)$
   - Assuming that a is projected on b: $(a \rightarrow b) \in A$

ILLINOIS INSTITUTE OF TECHNOLOGY

Definition (Where-Provenance)

1. $Where(R, t, a) = \mathcal{A}(R, t, a)$
2. $Where(\sigma_C(q), t, a) = Where(q, t, a)$
3. $Where(\pi_A(q), t, b) = \bigcup_{u.A=t} Where(q, u, a)$
   - Assuming that a is projected on b: $(a \rightarrow b) \in A$
4. $Where(q_1 \bowtie_C q_2, t, a) =$
   $Where(q_1, t.Q_1, a) \cup Where(q_2, t.Q_2, a)$

ILLINOIS INSTITUTE OF TECHNOLOGY

Definition (Where-Provenance)

1. $Where(R, t, a) = \mathcal{A}(R, t, a)$
2. $Where(\sigma_C(q), t, a) = Where(q, t, a)$
3. $Where(\pi_A(q), t, b) = \bigcup_{u.A=t} Where(q, u, a)$
   - Assuming that a is projected on b: $(a \rightarrow b) \in A$
4. $Where(q_1 \bowtie_C q_2, t, a) =$
   $Where(q_1, t.Q_1, a) \cup Where(q_2, t.Q_2, a)$
5. $Where(q_1 \cup q_2, t, a) = Where(q_1, t, a) \cup Where(q_2, t, a)$

Comments

- $Where(q, t, a) = \emptyset$ if q has no result attribute a
- Projection only on attribute + renaming
- Tuple t here refers to the values not the identity!
  - E.g., $(Peter, Chicago)$ and not t_2

Where-Provenance Example

Example

result

	name	street	city
t_1	Bob	10 W 31st	Chicago
t_2	Alice	75 Cermak	Chicago
t_3	Gertrud	15 Ellis	Berlin

```sql
SELECT P.name, A.street, A.city
FROM person P, address A
WHERE P.addr = A.street
```

person

	name	addr
p_1	Bob	10 W 31st
p_2	Alice	75 Cermak
p_2	Gertrud	15 Ellis

address

	street	city
a_1	10 W 31st	Chicago
a_2	75 Cermak	Chicago
a_3	15 Ellis	Berlin

Where-Provenance Example

Example

result

	name	street	city
t_1	Bob	10 W 31st	Chicago
t_2	Alice	75 Cermak	Chicago
t_3	Gertrud	15 Ellis	Berlin

$$q = \pi_{name,A.street,A.city}(q_1)$$

$$q_1 = person \bowtie_{addr=street} address$$

person

	name	addr
p_1	Bob	10 W 31st
p_2	Alice	75 Cermak
p_2	Gertrud	15 Ellis

address

	street	city
a_1	10 W 31st	Chicago
a_2	75 Cermak	Chicago
a_3	15 Ellis	Berlin

Where-Provenance Example

Example

$$Where(q, t_2, name) = Where(q_1, i_2, name)$$
$$Where(q_1, i_2, name) = Where(person, p_2, name) \cup Where(address, a_2, name)$$
$$Where(person, p_2, name) = \{a_2, a_3\}$$
$$Where(address, a_2, name) = \{\}$$

person

	name	addr
p_1	Bob	10 W 31st$^{\{a_1\}}$
p_2	Alice$^{\{a_2,a_3\}}$	75 Cermak
p_2	Gertrud	15 Ellis

address

	street	city
a_1	10 W 31st	Chicago$^{\{a_4\}}$
a_2	75 Cermak$^{\{a_5\}}$	Chicago
a_3	15 Ellis	Berlin

a_1 ="since 2005" a_2 ="artist" a_3 ="single" a_4 ="south side"
a_5 ="construction"

Where-Provenance Example

Example

result

	name	street	city
t_1	Bob	10 W 31st	Chicago$^{\{a_4\}}$
t_2	Alice$^{\{a_2,a_3\}}$	75 Cermak$^{\{a_5\}}$	Chicago
t_3	Gertrud	15 Ellis	Berlin

person

	name	addr
p_1	Bob	10 W 31st$^{\{a_1\}}$
p_2	Alice$^{\{a_2,a_3\}}$	75 Cermak
p_2	Gertrud	15 Ellis

address

	street	city
a_1	10 W 31st	Chicago$^{\{a_4\}}$
a_2	75 Cermak$^{\{a_5\}}$	Chicago
a_3	15 Ellis	Berlin

$a_1 =$"since 2005" $a_2 =$"artist" $a_3 =$"single" $a_4 =$"south side"
$a_5 =$"construction"

Sensitivity to Query Rewrite

- Consider equivalent variation of example query

Example

```
SELECT P.name, A.street, A.city
FROM person P, address A
WHERE P.addr = A.street
```

result

	name	street	city
t_1	Bob	10 W 31st	Chicago$^{\{a_4\}}$
t_2	Alice$^{\{a_2,a_3\}}$	75 Cermak$^{\{a_5\}}$	Chicago
t_3	Gertrud	15 Ellis	Berlin

Sensitivity to Query Rewrite

- Consider equivalent variation of example query

Example

```sql
SELECT P.name, P.addr AS street, A.city
FROM person P, address A
WHERE P.addr = A.street
```

result

	name	street	city
t_1	Bob	10 W 31st$^{\{a_1\}}$	Chicago$^{\{a_4\}}$
t_2	Alice$^{\{a_2,a_3\}}$	75 Cermak$^{\{a_5\}}$	Chicago
t_3	Gertrud	15 Ellis	Berlin

Outline

ILLINOIS INSTITUTE OF TECHNOLOGY

Insensitive Where-Provenance

Idea

- For query q
- Consider all queries equivalent to q
- Annotate output with the union of the annotations produced for each of these queries
- $\Rightarrow$Gather all annotations that could have been copied

Insensitive Where-Provenance

Definition (Insensitive Where-Provenance)

$$IWhere(q, t, a) = \bigcup_{q' \equiv q} Where(q', t, a)$$

ILLINOIS INSTITUTE OF TECHNOLOGY

Insensitive Where-Provenance

Definition (Insensitive Where-Provenance)

$IWhere(q, t, a) = \bigcup_{q' \equiv q} Where(q', t, a)$

- Consider definition of union: Union annotations
- $\Rightarrow IWhere(q, t, a) = Where(\bigcup_{q' \equiv q} q', t, a)$

Insensitive Where-Provenance

Definition (Insensitive Where-Provenance)

$IWhere(q, t, a) = \bigcup_{q' \equiv q} Where(q', t, a)$

- Consider definition of union: Union annotations
- $\Rightarrow IWhere(q, t, a) = Where(\bigcup_{q' \equiv q} q', t, a)$

Problem

- There are infinitely many equivalent queries
- How to practically compute this?

ILLINOIS INSTITUTE OF TECHNOLOGY

Equivalent Queries

- Even though the number of equivalent queries is infinite

OF TECHNOLOGY

Equivalent Queries

- Even though the number of equivalent queries is infinite
- The number of queries with different annotation propagation behaviour is finite
  - The number of cells in input/output is finite
  - For each output cells - either input cells annotations are added or not
  - $\Rightarrow 2^{IC*OC}$ potential annotation behaviours
    - IC number of input cells
    - OC number of output cells

OF TECHNOLOGY

Equivalent Queries

- Even though the number of equivalent queries is infinite
- The number of queries with different annotation propagation behaviour is finite
  - The number of cells in input/output is finite
  - For each output cells - either input cells annotations are added or not
  - $\Rightarrow 2^{IC * OC}$ potential annotation behaviours
    - IC number of input cells
    - OC number of output cells
  - $\Rightarrow$ large, but finite

OF TECHNOLOGY

Equivalent Queries

- Even though the number of equivalent queries is infinite
- The number of queries with different annotation propagation behaviour is finite
  - The number of cells in input/output is finite
  - For each output cells - either input cells annotations are added or not
  - $\Rightarrow 2^{IC*OC}$ potential annotation behaviours
    - *IC* number of input cells
    - *OC* number of output cells
  - $\Rightarrow$large, but finite
  - $\Rightarrow$unlikely that equivalent queries produce all these combinations!

OF TECHNOLOGY

Query Basis

- $\mathcal{E}(q) = \{q' \mid q' \equiv q\}$
- Query Basis $\mathcal{B}(q)$ for q:
  - $\bigcup_{q' \in \mathcal{B}(q)} q' =_{\mathcal{A}} \bigcup_{q' \in \mathcal{E}(q)} q'$
  - A Query basis has same annotation behaviour as all equivalent queries!

Query Basis

- $\mathcal{E}(q) = \{q' \mid q' \equiv q\}$
- Query Basis $\mathcal{B}(q)$ for q:
  - $\bigcup_{q' \in \mathcal{B}(q)} q' =_{\mathcal{A}} \bigcup_{q' \in \mathcal{E}(q)} q'$
  - A Query basis has same annotation behaviour as all equivalent queries!
  - How to find a query basis?

Query Basis For SPJ Queries

Types of equivalent SPJ queries

ILLINOIS INSTITUTE OF TECHNOLOGY

Query Basis For SPJ Queries

Types of equivalent SPJ queries

1. Through attributes that are equated
    - Can project on either of them with different propagation behaviour
    - See gene example
    - E.g., $\pi_a(\sigma_{a=b}(R)) \equiv \pi_b(\sigma_{b=a}(R))$

ILLINOIS INSTITUTE OF TECHNOLOGY

Query Basis For SPJ Queries

Types of equivalent SPJ queries

1. Through attributes that are equated
   - Can project on either of them with different propagation behaviour
   - See gene example
   - E.g., $\pi_a(\sigma_{a=b}(R)) \equiv \pi_b(\sigma_{b=a}(R))$
2. Through redundant joins
   - $\pi_A(R) \equiv \pi_A(R \bowtie_{A=A'} \pi_{A \rightarrow A'}(R))$
   - Each tuple finds at least one join partner (itself)
   - Additional join partners can carry additional annotations

ILLINOIS INSTITUTE OF TECHNOLOGY

Example Redundant Joins

Example

```
SELECT P.name, A.street, A.city
FROM person P, address A
WHERE P.addr = A.street
```

```
SELECT P.name, A.street, R.city
FROM person P, address A, address R
WHERE P.addr = A.street AND A.city = R.city
```

ILLINOIS INSTITUTE OF TECHNOLOGY

Algorithm - Generate Query Basis

1. Initialize $\mathcal{B}(q) = \emptyset$
2. For each $a = a'$ in q with a in result schema
   - Add variant of q where projection on a is replaced with projection on a'
3. For each query in $\mathcal{B}(q)$ and attribute a from relation R in projection
   - Add variant of q where an addition join $\bowtie_{a=a'} \pi_{a \rightarrow a'}(R)$ is added and replace final projection of a with a'

ILLINOIS INSTITUTE OF TECHNOLOGY

Example - Query Basis

Example

Step 2

$$q = \pi_{name,street,city}(person \bowtie_{addr=street} address)$$
$$q' = \pi_{name,addr,city}(person \bowtie_{addr=street} address)$$

Example - Query Basis

Example

Step 3

$$q'' = \pi_{name',street,city}(person \bowtie_{addr=street} address \bowtie_{name=name'} \pi_{name \rightarrow name'}(person))$$

$$q''' = \pi_{name,street,city'}(person \bowtie_{addr=street} address \bowtie_{city=city'} \pi_{city \rightarrow city'}(address))$$

$$q'''' = \pi_{name,addr',city}(person \bowtie_{addr=street} address \bowtie_{addr=addr'} \pi_{addr \rightarrow addr'}(person))$$

$$q''''' = \pi_{name,street',city}(person \bowtie_{addr=street} address \bowtie_{street=street'} \pi_{street \rightarrow street'}(address))$$

Example - Query Basis

Example

result

	name	street	city
t_1	Bob	10 W 31st$^{\{a_1\}}$	Chicago$^{\{a_4\}}$
t_2	Alice$^{\{a_2,a_3\}}$	75 Cermak$^{\{a_5\}}$	Chicago$^{\{a_4\}}$
t_3	Gertrud	15 Ellis	Berlin

person

	name	addr
p_1	Bob	10 W 31st$^{\{a_1\}}$
p_2	Alice$^{\{a_2,a_3\}}$	75 Cermak
p_2	Gertrud	15 Ellis

address

	street	city
a_1	10 W 31st	Chicago$^{\{a_4\}}$
a_2	75 Cermak$^{\{a_5\}}$	Chicago
a_3	15 Ellis	Berlin

Discussion

Effect of Insensitive Where-Provenance

- Annotations from equated attributes are propagated too
  - Reasonable, can be seen as type of copying

ILLINOIS INSTITUTE OF TECHNOLOGY

Discussion

Effect of Insensitive Where-Provenance

- Annotations from equated attributes are propagated too
  - Reasonable, can be seen as type of copying
- Annotations in attribute *a* from other tuples in the same relation are propagated if the tuple has the same value in attribute *a*
  - E.g., `city` attribute in running example
  - debatable whether correct semantics for every use case!

ILLINOIS INSTITUTE OF TECHNOLOGY

Outline

1. Where-Provenance and DBNotes
   - Introduction
   - Where-Provenance Model
   - Insensitive Where-Provenance
   - Insensitive Where-Provenance for Queries With Union
   - DBNotes
   - Recap

From SPJ to Union queries

Observations

- Adding Union adds new types of equivalent queries
- ⇒every query that is contained in a query can be union-ed to the query without changing the result
- ⇒this is independent of whether original query uses union!

ILLINOIS INSTITUTE OF TECHNOLOGY

Query Containment with Union

- Relation $R(a, b)$ and $S(c)$
- $q = \pi_a(R)$

ILLINOIS INSTITUTE OF TECHNOLOGY

Query Containment with Union

- Relation $R(a, b)$ and $S(c)$
- $q = \pi_a(R)$
- $q' = (\pi_a(R) \cup \pi_c(R \bowtie_{a=c} S))$

ILLINOIS INSTITUTE OF TECHNOLOGY

Query Containment with Union

- Relation $R(a, b)$ and $S(c)$
- $q = \pi_a(R)$
- $q' = (\pi_a(R) \cup \pi_c(R \bowtie_{a=c} S))$
- $q \equiv q'$

ILLINOIS INSTITUTE OF TECHNOLOGY

Generating Query Basis - Union

Step 4

- For each attribute a in result of q and relation R with attribute b in database
  - Add query $q \cup q'$ to $\mathcal{B}(q)$ with
  - q' is q with additional join with R on $a = b$

Example Query Basis

Example

Step 2

$$q = \pi_{name,street,city}(person \bowtie_{addr=street} address)$$
$$q' = \pi_{name,addr,city}(person \bowtie_{addr=street} address)$$

Example Query Basis

Example

Step 3

$$q_3 = \pi_{name',street,city}(person \bowtie_{addr=street} address \bowtie_{name=name'} \pi_{name \rightarrow name'}(person))$$

$$q_3' = \pi_{name,street,city'}(person \bowtie_{addr=street} address \bowtie_{city=city'} \pi_{city \rightarrow city'}(address))$$

$$q_3'' = \pi_{name,addr',city}(person \bowtie_{addr=street} address \bowtie_{addr=addr'} \pi_{addr \rightarrow addr'}(person))$$

$$q_3''' = \pi_{name,street',city}(person \bowtie_{addr=street} address \bowtie_{street=street'} \pi_{street \rightarrow street'}(address))$$

Example Query Basis

Example

Step 4

Joins on `person.name`

$$q_4 = q \cup \pi_{addr',street,city}(person \bowtie_{addr=street} address \bowtie_{name=addr'} \pi_{addr \rightarrow addr'}(person))$$

$$q'_4 = q \cup \pi_{street',street,city}(person \bowtie_{addr=street} address \bowtie_{name=street'} \pi_{street \rightarrow street'}(address))$$

$$q''_4 = q \cup \pi_{city',street,city}(person \bowtie_{addr=street} address \bowtie_{name=city'} \pi_{city \rightarrow city'}(address))$$

Example Query Basis

Example

Step 4

Joins on `address.street`

$$q_4''' = q \cup \pi_{name,name',city}(person \bowtie_{addr=street} address \bowtie_{street=name'} \pi_{name \rightarrow name'}(person))$$

...

Joins on `address.city`

...

Example Query Basis

Example

result

	name	street	city
t_1	Bob	10 W 31st$^{\{a_1\}}$	Chicago$^{\{a_4\}}$
t_2	Alice$^{\{a_2,a_3\}}$	75 Cermak$^{\{a_5\}}$	Chicago$^{\{a_4\}}$
t_3	Gertrud	15 Ellis	Berlin

person

	name	addr
p_1	Bob	10 W 31st$^{\{a_1\}}$
p_2	Alice$^{\{a_2,a_3\}}$	75 Cermak
p_2	Gertrud	15 Ellis

address

	street	city
a_1	10 W 31st	Chicago$^{\{a_4\}}$
a_2	75 Cermak$^{\{a_5\}}$	Chicago
a_3	15 Ellis	Berlin

Discussion

- An output cell (q, t, a) will carry all annotations from cells in the database that have the same attribute value

Discussion

- An output cell (q, t, a) will carry all annotations from cells in the database that have the same attribute value
- **Rationale**: annotations are bound to values not cells

ILLINOIS INSTITUTE OF TECHNOLOGY

Discussion

- An output cell (q, t, a) will carry all annotations from cells in the database that have the same attribute value
- **Rationale**: annotations are bound to values not cells
- Still strange in a lot of cases.
  - E.g., numeric attributes age in `person` and `street number` in `company`
  - Query `SELECT * FROM person`
  - ⇒Annotations on company street numbers should carry over to the age of a person?

ILLINOIS INSTITUTE OF TECHNOLOGY

Outline

DBNotes

Approach

- Add annotations to DBMS
- User queries DB with pSQL:
  - USPJ SQL queries
  - plus language constructs to define how annotations are propagated
- Implement as middleware layer over standard DBMS
- Store annotations + data in standard DBMS
- Middleware translates pSQL into SQL queries DB
- DB results are transformed into annotated relations

ILLINOIS INSTITUTE OF TECHNOLOGY

pSQL

pSQL query block

```
SELECT DISTINCT selectlist
FROM            fromlist
WHERE           wherelist
PROPAGATE       (DEFAULT
                 | DEFAULT-ALL
                 | R1.a1 TO b1, ..., Rn.an TO bn)
```

- pSQL query is union of query blocks
- *selectlist*: Select attributes and rename ($R_1.a_1 \rightarrow b_1$)
- *fromlist*: List of relations with potential aliases
- *wherelist*: conjunction of attribute-attribute or attribute-constant equalities

pSQL - the PROPAGATE clause

- Defines how annotations are copied from input to output
- Three different options:
  - Custom
  - `DEFAULT`
  - `DEFAULT-ALL`

ILLINOIS INSTITUTE OF TECHNOLOGY

pSQL - the PROPAGATE clause

- Defines how annotations are copied from input to output
- Three different options:
  - Custom
  - `DEFAULT`
  - `DEFAULT-ALL`

Custom

- User defines annotation propagation at the query level
- $R_1.a_1 \texttt{TO} b_1$ = add annotations from input attribute $R_1.a_1$ to result attribute b_1.

ILLINOIS INSTITUTE OF TECHNOLOGY

pSQL - the PROPAGATE clause

- Defines how annotations are copied from input to output
- Three different options:
  - Custom
  - `DEFAULT`
  - `DEFAULT-ALL`

DEFAULT

- Propagate annotations from all cells in the Where-provenance

ILLINOIS INSTITUTE OF TECHNOLOGY

pSQL - the PROPAGATE clause

- Defines how annotations are copied from input to output
- Three different options:
  - Custom
  - `DEFAULT`
  - `DEFAULT-ALL`

DEFAULT-ALL

- Propagate annotations from all cells in the insensitive Where-provenance

pSQL example query

Example

```
SELECT P.name, A.street, A.city
FROM person P, address A
WHERE P.addr = A.street
PROPAGATE P.name TO name,
          P.addr TO street,
          P.addr TO city
```

person

	name	addr
p_1	Bob	10 W 31st$^{\{a_1\}}$
p_2	Alice$^{\{a_2,a_3\}}$	75 Cermak
p_2	Gertrud	15 Ellis

address

	street	city
a_1	10 W 31st	Chicago$^{\{a_4\}}$
a_2	75 Cermak$^{\{a_5\}}$	Chicago
a_3	15 Ellis	Berlin

pSQL example query

Example

result

	name	street	city
t_1	Bob	10 W 31st$^{\{a_1\}}$	Chicago$^{\{a_1\}}$
t_2	Alice$^{\{a_2,a_3\}}$	75 Cermak	Chicago
t_3	Gertrud	15 Ellis	Berlin

person

	name	addr
p_1	Bob	10 W 31st$^{\{a_1\}}$
p_2	Alice$^{\{a_2,a_3\}}$	75 Cermak
p_2	Gertrud	15 Ellis

address

	street	city
a_1	10 W 31st	Chicago$^{\{a_4\}}$
a_2	75 Cermak$^{\{a_5\}}$	Chicago
a_3	15 Ellis	Berlin

Storage Scheme

- Schema
  - For each attribute A in relation R
  - Add attribute A_a that stores annotations on A
- Instance
  - Each tuple can store one annotation on each attribute
  - Duplicate tuples to fit in more annotations

Example

- $R(a, b)$ will be $R(a, a_a, b, b_a)$

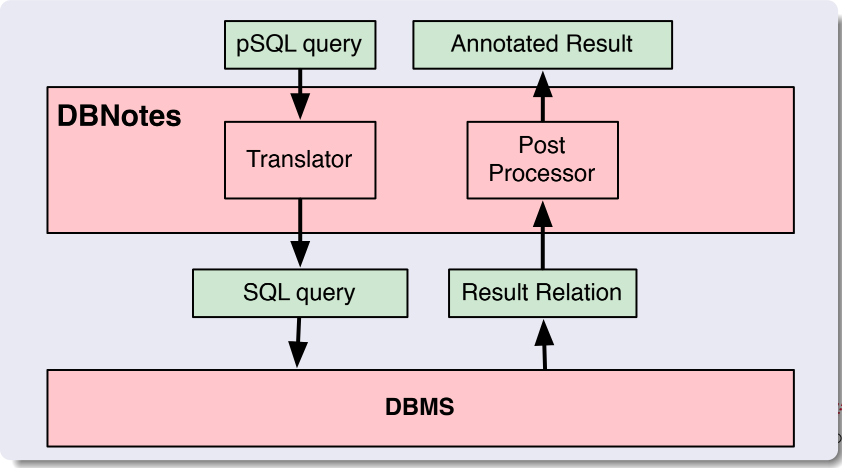
R

	a	b
r_1	$1^{a_1,a_2}$	2^{a_3}
r_2	2^{a_4}	3

R'

	a	a_a	b	b_a
r_1	1	a_1	2	a_3
r_1	1	a_2	2	-
r_1	2	a_4	3	-

DBNotes - Approach



Translator

1. Transform propagate clauses into *Custom* form
2. Build bins for each output attribute b
   - Add $R.a$ to bin if $R.a$ TO b in propagate clause

ILLINOIS INSTITUTE OF TECHNOLOGY

Translator

1. Transform propagate clauses into *Custom* form
2. Build bins for each output attribute b
   - Add $R.a$ to bin if $R.a$ `TO` b in propagate clause
3. Generate intermediate queries Q_1 to Q_n
   - While at least on bin not empty
   - Take one attribute a from each bin
   - Generate query that projects each a_a to b_a
   - Use `NULL TO` b_a if bin is empty

Translator

1. Transform propagate clauses into *Custom* form
2. Build bins for each output attribute b
   - Add $R.a$ to bin if $R.a$ `TO` b in propagate clause
3. Generate intermediate queries Q_1 to Q_n
   - While at least on bin not empty
   - Take one attribute a from each bin
   - Generate query that projects each a_a to b_a
   - Use `NULL TO` b_a if bin is empty
4. Generate wrapper query
   - *orderlist* all attribute of pSQL query

```
SELECT DISTINCT *
FROM (Q1 UNION ... UNION Qn)
ORDER BY orderlist
```

Post-Processor

- Gather all annotations for each result tuple
- Works like aggregation
  1. Initialize each attribute annotation sets to empty set
  2. For each tuple add annotations to sets
  3. If tuple has different attribute value
     - Output annotated result tuple
     - Start over at (1)

ILLINOIS INSTITUTE OF TECHNOLOGY

Example Processing

```
SELECT P.name, A.street, A.city
FROM person P, address A
WHERE P.addr = A.street
PROPAGATE P.name TO name,
          P.addr TO street,
          P.addr TO city
```

person

	name	addr
p_1	Bob	10 W 31st$^{\{a_1\}}$
p_2	Alice$^{\{a_2,a_3\}}$	75 Cermak
p_2	Gertrud	15 Ellis

address

	street	city
a_1	10 W 31st	Chicago$^{\{a_4\}}$
a_2	75 Cermak$^{\{a_5\}}$	Chicago
a_3	15 Ellis	Berlin

ILLINOIS INSTITUTE OF TECHNOLOGY

Example Processing - Storage

person′

	name	$name_a$	addr	$addr_a$
p_1	Bob		10 W 31st	a_1
p_2	Alice	a_2	75 Cermak	
p_2	Alice	a_3	75 Cermak	
p_2	Gertrud		15 Ellis	

address′

	street	$street_a$	city	$city_a$
a_1	10 W 31st		Chicago	a_4
a_2	75 Cermak	a_5	Chicago	
a_3	15 Ellis		Berlin	

Example Processing - Intermediate Queries

- **Bin** name: {P.name}
- **Bin** street: {P.addr}
- **Bin** city: {P.addr}

```
SELECT P.name, P.name_a,
       A.street, P.addr_a AS street_a,
       A.city, P.addr_a AS city_a
FROM person' P, address' A
WHERE P.addr = A.street
```

ILLINOIS INSTITUTE OF TECHNOLOGY

Example Processing - Wrapper Query

```
SELECT DISTINCT *
FROM (SELECT P.name, P.name_a,
        A.street, P.addr_a AS street_a,
        A.city, P.addr_a AS city_a
     FROM person' P, address' A
     WHERE P.addr = A.street) AS i
ORDER BY name, street, city
```

ILLINOIS INSTITUTE OF TECHNOLOGY

Example Processing - Post-processing

result

	name	$name_a$	street	$street_a$	city	$city_a$
t_1	Bob		10 W 31st	a_1	Chicago	a_1
t_2	Alice	a_2	75 Cermak		Chicago	
t_2	Alice	a_3	75 Cermak		Chicago	
t_3	Gertrud		15 Ellis		Berlin	

Example Processing - Post-processing

result

	name	street	city
t_1	Bob	10 W 31st$^{\{a_1\}}$	Chicago$^{\{a_1\}}$
t_2	Alice$^{\{a_2,a_3\}}$	75 Cermak	Chicago
t_3	Gertrud	15 Ellis	Berlin

result

	name	name_a	street	street_a	city	city_a
t_1	Bob		10 W 31st	a_1	Chicago	a_1
t_2	Alice	a_2	75 Cermak		Chicago	
t_2	Alice	a_3	75 Cermak		Chicago	
t_3	Gertrud		15 Ellis		Berlin	

Discussion

Types of Annotations

- On tuples
- On rectangular regions in relations
- Queries define what to annotate

Discussion

Annotation Propagation

- DBNotes propagation is schema based!
- allow conditions
  - over data: if $a > 5$ then annotate with annotations from attribute b
  - over annotations: propagate if annotation contains word “important”
- manipulation of annotations
  - Extract pattern
  - Concatenate annotations
  - ...

ILLINOIS INSTITUTE OF TECHNOLOGY

Outline

Recap

Where-Provenance

- **Rationale**: Models copy behaviour
- **Representation**: Set of cells
  - Cell is R, t, a
- **Syntactic Definition**:
  - For USPJ queries
- **Variants**:
  - Insensitive: Union for of provenance for equivalent queries

ILLINOIS INSTITUTE OF TECHNOLOGY

Recap

Annotation Propagation

- One semantics is based on Where-provenance
- User defined also possible

ILLINOIS INSTITUTE OF TECHNOLOGY

Recap

DBNotes

- Middleware implementations
- pSQL language
- Translated into SQL over relational storage schema
- Postprocessing

ILLINOIS INSTITUTE OF TECHNOLOGY

Provenance Model Comparison

Property	Why	Lin	PI-CS	Where
Representation	Set of Set of Tuples	List of Set of Tuples	Set/Bag of List of Tuples	Sets of Attribute Value Positions
Granularity	Tuple	Tuple	Tuple	Attribute Value
Language Support	USPJ	ASPJ-Set	ASPJ-Set + Nested subqueries	U-SPJ
Semantics	Set	Set + Bag*	Bag	Set
Variants	Wit, Why, IWhy	Set/Bag	Influence + Copy	SPJ + Insensitive + Insensitive Union
Definition	Decl. - Synt. - Decl./Synt.	Decl. + Synt.	Decl. + Synt.	Synt.
Design Principles	Sufficiency - No false positives	Sufficiency + No false negatives + no false positives	Sufficiency + No false negatives + No false positives	Copying
Systems	-	WHIPS	Perm	DBNotes
Insensitivity	Yes - No - Yes	No	No	No - Yes - Yes

Literature

James Cheney, Laura Chiticariu, and Wang-Chiew Tan.
Provenance in Databases: Why, How, and Where.
Foundations and Trends in Databases, 1(4):379–474, 2009.

Laura Chiticariu, Wang-Chiew Tan, and Gaurav Vijayvargiya.
DBNotes: a Post-it System for Relational Databases based on Provenance.
In SIGMOD '05: Proceedings of the 31th SIGMOD International Conference on Management of Data, 942–944, 2005.

Deepavali Bhagwat, Laura Chiticariu, Wang-Chiew Tan, and Gaurav Vijayvargiya.
An Annotation Management System for Relational Databases.
The VLDB Journal, 14(4):373–396, 2005.

Deepavali Bhagwat, Laura Chiticariu, Wang-Chiew Tan, and Gaurav Vijayvargiya.
An Annotation Management System for Relational Databases.
In VLDB '04: Proceedings of the 30th International Conference on Very Large Data Bases, 900–911, 2004.

Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan.
Why and Where: A Characterization of Data Provenance.
In ICDT '01: Proceedings of the 8th International Conference on Database Theory, 316–330, 2001.