



# Data wrangling, data quality

## Provenance in Databases: Principles and Applications

Pierre Senellart



7 March 2022



## Provenance management

- Data management **all about query evaluation**



## Provenance management

- Data management **all about query evaluation**
- What if we want **something more** than the query result?
  - Where does the result come from?
  - Why was this result obtained?
  - How was the result produced?
  - What is the probability of the result?
  - How many times was the result obtained?
  - How would the result change if part of the input data was missing?
  - What is the minimal security clearance I need to see the result?
  - What is the most economical way of obtaining the result?
  - How can a result be explained in layman terms?

## Provenance management

- Data management **all about query evaluation**
- What if we want **something more** than the query result?
  - Where does the result come from?
  - Why was this result obtained?
  - How was the result produced?
  - What is the probability of the result?
  - How many times was the result obtained?
  - How would the result change if part of the input data was missing?
  - What is the minimal security clearance I need to see the result?
  - What is the most economical way of obtaining the result?
  - How can a result be explained in layman terms?
- **Provenance management**: along with query evaluation, record **additional bookkeeping information** allowing to answer the questions above





## Workflow provenance vs fine-grained provenance

### Workflow provenance

[Davidson et al., 2007]

- Uniquely identifies **datasets** used and produced
- Documents every **action** carried out (date, tool, version, parameters, inputs, outputs, etc.)
- Typically has a simple **directed graph structure**



## Workflow provenance vs fine-grained provenance

### Workflow provenance

[Davidson et al., 2007]

- Uniquely identifies **datasets** used and produced
- Documents every **action** carried out (date, tool, version, parameters, inputs, outputs, etc.)
- Typically has a simple **directed graph structure**

### Data (fine-grained) provenance

[Buneman et al., 2001]

- At the level of a **single data item** (a record, a data value, a node in a graph, etc.)
- Documents **how** this particular data item was produced
- Possibly a **rich mathematical structure**
- Support for a **limited** set of data operations



# Outline

## Provenance

### Preliminaries

Boolean provenance

Semiring provenance

And beyond...

## Applications

## Implementing Provenance Support

## Conclusion



## Data model

- **Relational data model**: data decomposed into relations, with labeled attributes. . .



## Data model

- **Relational data model:** data decomposed into relations, with labeled attributes...

name	position	city	classification
John	Director	New York	unclassified
Paul	Janitor	New York	restricted
Dave	Analyst	Paris	confidential
Ellen	Field agent	Berlin	secret
Magdalen	Double agent	Paris	top secret
Nancy	HR director	Paris	restricted
Susan	Analyst	Berlin	secret



## Data model

- **Relational data model**: data decomposed into relations, with labeled attributes...
- ... with an extra **provenance annotation** for each tuple (think of it first as a tuple id)

name	position	city	classification	prov
John	Director	New York	unclassified	$t_1$
Paul	Janitor	New York	restricted	$t_2$
Dave	Analyst	Paris	confidential	$t_3$
Ellen	Field agent	Berlin	secret	$t_4$
Magdalen	Double agent	Paris	top secret	$t_5$
Nancy	HR director	Paris	restricted	$t_6$
Susan	Analyst	Berlin	secret	$t_7$



## Relations and databases

Formally:

- A **relational schema**  $\mathcal{R}$  is a finite sequence of distinct **attribute names**; the **arity** of  $\mathcal{R}$  is  $|\mathcal{R}|$
- A **database schema** is a mapping from **relation names** to **relational schemas**, with finite support
- A **tuple** over relation schema  $\mathcal{R}$  is a mapping from  $\mathcal{R}$  to **data values**; each tuple comes with a **provenance annotation**
- A **relation instance** (or **relation**) over  $\mathcal{R}$  is a finite set of **tuples** over  $\mathcal{R}$
- A **database instance** (or **database**) over database schema  $\mathcal{D}$  is a mapping from the support of  $\mathcal{D}$  mapping each **relation name**  $R$  to a **relation instance** over  $\mathcal{D}(R)$



## Queries

- A **query** is an arbitrary **function** that maps databases over a fixed database schema  $\mathcal{D}$  to relations over some relational schema  $\mathcal{R}$
- The query does **not** consider or produce any provenance annotations; we will give semantics for the provenance annotations of the output, based on that of the input
- In practice, one often restricts to specific query languages:
  - Monadic-Second Order logic (MSO)
  - First-Order logic (FO) or the relational algebra
  - SQL with aggregate functions
  - etc.



# Outline

## Provenance

Preliminaries

**Boolean provenance**

Semiring provenance

And beyond...

## Applications

## Implementing Provenance Support

## Conclusion



## Boolean provenance [Imieliński and Lipski, 1984]

- $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$  finite set of **Boolean events**
- **Provenance annotation**: **Boolean function** over  $\mathcal{X}$ , i.e., a function of the form:  $(\mathcal{X} \rightarrow \{\perp, \top\}) \rightarrow \{\perp, \top\}$
- **Interpretation**: possible-world semantics
  - every valuation  $\nu : \mathcal{X} \rightarrow \{\perp, \top\}$  denotes a **possible world** of the database
  - the provenance of a tuple on  $\nu$  evaluates to  $\perp$  or  $\top$  depending whether this tuple **exists** in that possible world
  - for example, if every tuple of a database is annotated with the **indicator function** of a distinct Boolean event, the set of possible worlds is the set of **all subdatabases**



## Example of possible worlds

name	position	city	classification	prov
John	Director	New York	unclassified	$t_1$
Paul	Janitor	New York	restricted	$t_2$
Dave	Analyst	Paris	confidential	$t_3$
Ellen	Field agent	Berlin	secret	$t_4$
Magdalen	Double agent	Paris	top secret	$t_5$
Nancy	HR director	Paris	restricted	$t_6$
Susan	Analyst	Berlin	secret	$t_7$

$$\nu: \begin{array}{ccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \\ \top & \top & \top & \top & \top & \top & \top \end{array}$$



## Example of possible worlds

name	position	city	classification	prov
John	Director	New York	unclassified	$t_1$
Dave	Analyst	Paris	confidential	$t_3$
Magdalen	Double agent	Paris	top secret	$t_5$
Susan	Analyst	Berlin	secret	$t_7$

$$\nu: \begin{array}{ccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \\ \top & \perp & \top & \perp & \top & \perp & \top \end{array}$$



## Boolean provenance of query results

- $\nu(D)$ : the **subdatabase** of  $D$  where all tuples whose provenance annotation evaluates to  $\perp$  by  $\nu$  are removed
- The **Boolean provenance**  $\text{prov}_{q,D}(t)$  of tuple  $t \in q(D)$  is the function:

$$\nu \mapsto \begin{cases} \top & \text{if } t \in q(\nu(D)) \\ \perp & \text{otherwise} \end{cases}$$

### Example (What cities are in the table?)

name	position	city	classification	prov
John	Director	New York	unclassified	$t_1$
Paul	Janitor	New York	restricted	$t_2$
Dave	Analyst	Paris	confidential	$t_3$
Ellen	Field agent	Berlin	secret	$t_4$
Magdalen	Double agent	Paris	top secret	$t_5$
Nancy	HR director	Paris	restricted	$t_6$
Susan	Analyst	Berlin	secret	$t_7$

city	prov
New York	$t_1 \vee t_2$
Paris	$t_3 \vee t_5 \vee t_6$
Berlin	$t_4 \vee t_7$



## What now?

- How to **compute** Boolean provenance for practical query languages? What complexity?
- What can we do with provenance?
- How should we **represent** provenance annotations?
- How can we **implement** support for provenance management in a relational database management system?



# Outline

## Provenance

Preliminaries

Boolean provenance

**Semiring provenance**

And beyond...

## Applications

Implementing Provenance Support

## Conclusion



## Commutative semiring $(K, 0, 1, \oplus, \otimes)$

- Set  $K$  with distinguished elements  $0, 1$
- $\oplus$  **associative, commutative** operator, with identity  $0_K$ :
  - $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
  - $a \oplus b = b \oplus a$
  - $a \oplus 0 = 0 \oplus a = a$
- $\otimes$  **associative, commutative** operator, with identity  $1_K$ :
  - $a \otimes (b \otimes c) = (a \otimes b) \otimes c$
  - $a \otimes b = b \otimes a$
  - $a \otimes 1 = 1 \otimes a = a$
- $\otimes$  **distributes** over  $\oplus$ :

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

- $0$  is **annihilating** for  $\otimes$ :

$$a \otimes 0 = 0 \otimes a = 0$$



## Example semirings

- $(\mathbb{N}, 0, 1, +, \times)$ : **counting** semiring
- $(\{\perp, \top\}, \perp, \top, \vee, \wedge)$ : **Boolean** semiring
- $(\{unclassified, restricted, confidential, secret, top\ secret\}, top\ secret, unclassified, \min, \max)$ : **security** semiring
- $(\mathbb{N} \cup \{\infty\}, \infty, 0, \min, +)$ : **tropical** semiring
- $(\{\text{Boolean functions over } \mathcal{X}\}, \perp, \top, \vee, \wedge)$ : semiring of **Boolean functions** over  $\mathcal{X}$
- $(\mathbb{N}[\mathcal{X}], 0, 1, +, \times)$ : semiring of integer-valued **polynomials** with variables in  $\mathcal{X}$  (also called **How**-semiring or **universal** semiring, see further)
- $(\mathcal{P}(\mathcal{P}(\mathcal{X})), \emptyset, \{\emptyset\}, \cup, \uplus)$ : **Why**-semiring over  $\mathcal{X}$   
 $(A \uplus B := \{a \cup b \mid a \in A, b \in B\})$



## Semiring provenance [Green et al., 2007]

- We **fix** a semiring  $(K, 0, 1, \oplus, \otimes)$
- We assume provenance annotations are **in  $K$**
- We consider a query  $q$  from the **positive relational algebra** (selection, projection, renaming, cross product, union; joins can be simulated with renaming, cross product, selection, projection)
- We define a semantics for the provenance of a tuple  $t \in q(D)$  **inductively** on the structure of  $q$



## Selection, renaming

Provenance annotations of selected tuples are **unchanged**

Example ( $\rho_{\text{name} \rightarrow n}(\sigma_{\text{city}=\text{"New York"}}(R))$ )

name	position	city	classification	prov
John	Director	New York	unclassified	$t_1$
Paul	Janitor	New York	restricted	$t_2$
Dave	Analyst	Paris	confidential	$t_3$
Ellen	Field agent	Berlin	secret	$t_4$
Magdalen	Double agent	Paris	top secret	$t_5$
Nancy	HR director	Paris	restricted	$t_6$
Susan	Analyst	Berlin	secret	$t_7$

n	position	city	classification	prov
John	Director	New York	unclassified	$t_1$
Paul	Janitor	New York	restricted	$t_2$



## Projection

Provenance annotations of identical, merged, tuples are  $\oplus$ -ed

Example ( $\pi_{\text{city}}(R)$ )

name	position	city	classification	prov
John	Director	New York	unclassified	$t_1$
Paul	Janitor	New York	restricted	$t_2$
Dave	Analyst	Paris	confidential	$t_3$
Ellen	Field agent	Berlin	secret	$t_4$
Magdalen	Double agent	Paris	top secret	$t_5$
Nancy	HR director	Paris	restricted	$t_6$
Susan	Analyst	Berlin	secret	$t_7$

city	prov
New York	$t_1 \oplus t_2$
Paris	$t_3 \oplus t_5 \oplus t_6$
Berlin	$t_4 \oplus t_7$



## Union

Provenance annotations of identical, merged, tuples are  $\oplus$ -ed

### Example

$$\pi_{\text{city}}(\sigma_{\text{ends-with}(\text{position}, \text{"agent"})}(R)) \cup \pi_{\text{city}}(\sigma_{\text{position}=\text{"Analyst"}}(R))$$

name	position	city	classification	prov
John	Director	New York	unclassified	$t_1$
Paul	Janitor	New York	restricted	$t_2$
Dave	Analyst	Paris	confidential	$t_3$
Ellen	Field agent	Berlin	secret	$t_4$
Magdalen	Double agent	Paris	top secret	$t_5$
Nancy	HR director	Paris	restricted	$t_6$
Susan	Analyst	Berlin	secret	$t_7$

city	prov
Paris	$t_3 \oplus t_5$
Berlin	$t_4 \oplus t_7$



## Cross product

Provenance annotations of combined tuples are  $\otimes$ -ed

### Example

$$\pi_{\text{city}}(\sigma_{\text{ends-with}(\text{position}, \text{"agent"})}(R)) \bowtie \pi_{\text{city}}(\sigma_{\text{position}=\text{"Analyst"}}(R))$$

name	position	city	classification	prov
John	Director	New York	unclassified	$t_1$
Paul	Janitor	New York	restricted	$t_2$
Dave	Analyst	Paris	confidential	$t_3$
Ellen	Field agent	Berlin	secret	$t_4$
Magdalen	Double agent	Paris	top secret	$t_5$
Nancy	HR director	Paris	restricted	$t_6$
Susan	Analyst	Berlin	secret	$t_7$

city	prov
Paris	$t_3 \otimes t_5$
Berlin	$t_4 \otimes t_7$



## What can we do with it?

**counting semiring:** count the number of times a tuple can be derived, multiset semantics

**Boolean semiring:** determines if a tuple exists when a subdatabase is selected

**security semiring:** determines the minimum clearance level required to get a tuple as a result

**tropical semiring:** minimum-weight way of deriving a tuple (think shortest path in a graph)

**Boolean functions:** Boolean provenance, as previously defined

**integer polynomials:** universal provenance, see further

**Why-semiring:** Why-provenance [Buneman et al., 2001], set of combinations of tuples needed for a tuple to exist



## Example of security provenance

$$\pi_{\text{city}}(\sigma_{\text{name} < \text{name}_2}(\pi_{\text{name}, \text{city}}(R) \bowtie \rho_{\text{name} \rightarrow \text{name}_2}(\pi_{\text{name}, \text{city}}(R))))$$

name	position	city	prov
John	Director	New York	unclassified
Paul	Janitor	New York	restricted
Dave	Analyst	Paris	confidential
Ellen	Field agent	Berlin	secret
Magdalen	Double agent	Paris	top secret
Nancy	HR director	Paris	restricted
Susan	Analyst	Berlin	secret

city	prov
New York	restricted
Paris	confidential
Berlin	secret



## Notes [Green et al., 2007]

- Computing provenance has a **PTIME** data complexity overhead
- Semiring **homomorphisms commute** with provenance computation: if there is a homomorphism from  $K$  to  $K'$ , then one can compute the provenance in  $K$ , apply the homomorphism, and obtain the same result as when computing provenance in  $K'$
- The integer polynomial semiring is **universal**: there is a unique homomorphism to any other commutative semiring that respects a given valuation of the variables
- This means **all computations can be performed in the universal semiring**, and homomorphisms applied next
- Two **equivalent queries** can have two **different provenance annotations** on the same database, in some semirings



# Outline

## Provenance

Preliminaries

Boolean provenance

Semiring provenance

**And beyond...**

## Applications

Implementing Provenance Support

## Conclusion



## Semirings with monus [Amer, 1984, Geerts and Poggi, 2010]

- Some semirings can be equipped with a  $\ominus$  verifying:
  - $a \oplus (b \ominus a) = b \oplus (a \ominus b)$
  - $(a \ominus b) \ominus c = a \ominus (b + c)$
  - $a \ominus a = 0 \ominus a = 0$
- Boolean function semiring with  $\wedge, \neg$ , Why-semiring with  $\setminus$ , counting semiring with **truncated difference**...
- Most natural semirings (but not all semirings [Amarilli and Monet, 2016]!) can be extended into **semirings with monus**
- Sometimes strange things happen [Amsterdamer et al., 2011a]: e.g,  $\otimes$  does **not always distribute** over  $\ominus$
- Allows supporting **full relational algebra** with the  $\setminus$  operator, still **PTIME**
- Semantics for Boolean function semiring **coincides** with that of Boolean provenance



## Difference

Provenance annotations of diff-ed tuples are  $\Theta$ -ed

### Example

$$\pi_{\text{city}}(\sigma_{\text{ends-with}(\text{position}, \text{"agent"})}(R)) \setminus \pi_{\text{city}}(\sigma_{\text{position}=\text{"Analyst"}}(R))$$

name	position	city	classification	prov
John	Director	New York	unclassified	$t_1$
Paul	Janitor	New York	restricted	$t_2$
Dave	Analyst	Paris	confidential	$t_3$
Ellen	Field agent	Berlin	secret	$t_4$
Magdalen	Double agent	Paris	top secret	$t_5$
Nancy	HR director	Paris	restricted	$t_6$
Susan	Analyst	Berlin	secret	$t_7$

city	prov
Paris	$t_5 \ominus t_3$
Berlin	$t_4 \ominus t_7$



## Provenance for aggregates

[Amsterdamer et al., 2011b, Fink et al., 2012]

- **Trickier** to define provenance for queries with aggregation, even in the Boolean case
- One can construct a  $K$ -**semimodule**  $K * M$  for each monoid aggregate  $M$  over a provenance database with a semiring in  $K$
- Data **values** become elements of the semimodule

Example ( $\text{count}(\pi_{\text{name}}(\sigma_{\text{city}=\text{"Paris"}}(R)))$ )

$$t_3 * 1 + t_5 * 1 + t_6 * 1$$



## Provenance in XML databases

[Foster et al., 2008]

**Data:** Trees (with different kinds of nodes, with data values on leaves. . .)

**Queries:** XPath, XQuery, expressing in particular **tree-pattern queries**

**Provenance annotations:** on nodes of the tree; a node “inherits” **annotations of its ancestors**

Boolean and semiring provenance extend quite naturally to this setting, cf. works on Probabilistic XML [Abiteboul et al., 2009] and Annotated XML [Foster et al., 2008].



## Provenance in graph databases

[Ramusat et al., 2018]

**Data:** Graphs (with properties on nodes, edges...)

**Queries:** Graph query languages (such as Cypher),  
especially **Regular Path Queries**

**Provenance annotations:** on nodes or edges of the graphs

Semiring provenance extends to this setting, but queries inherently **recursive**, so need for technical conditions on semiring (e.g.,  $\omega$ -continuity [Green et al., 2007], absorptivity [Deutch et al., 2014], existence of a  $*$  operator [Ramusat et al., 2018]) for provenance to be definable and for specific algorithms.



## Provenance in triple stores

[Damásio et al., 2012]

**Data:** Triples (subject, predicate, object) in an **open world**

**Queries:** SPARQL (including **negation** capabilities, e.g., optionality)

**Provenance annotations:** on triples

Provenance definition extends, but need for negation support, so m-semiring provenance; **additional axioms** need to be satisfied for compatibility with SPARQL semantics [Geerts et al., 2016]



# Outline

Provenance

**Applications**

Probabilistic databases

Views

Explanation

Implementing Provenance Support

Conclusion



## Application: Probabilistic databases

[Green and Tannen, 2006, Suciu et al., 2011]

- **Tuple-independent database:** each tuple  $t$  in a database is annotated with **independent** probability  $\Pr(t)$  of existing
- Probability of a possible world  $D' \subseteq D$ :

$$\Pr(D') = \prod_{t \in D'} \Pr(t) \times \prod_{t \in D' \setminus D} (1 - \Pr(t))$$

- Probability of a tuple for a query  $q$  over  $D$ :

$$\Pr(t \in q(D)) = \sum_{\substack{D' \subseteq D \\ t \in q(D')}} \Pr(D')$$

- If  $\Pr(x_i) := \Pr(t_i)$  where  $x_i$  is the provenance annotation of tuple  $t_i$  then  **$\Pr(t \in q(D)) = \Pr(\text{prov}_{q,D}(t))$**
- Computing the probability of a query in probabilistic databases thus amounts to **computing Boolean provenance**, and then computing the **probability of a Boolean function**
- Also works for more complex probabilistic models

## Example of probability computation

name	position	city	classification	prov	prob
John	Director	New York	unclassified	$t_1$	0.5
Paul	Janitor	New York	restricted	$t_2$	0.7
Dave	Analyst	Paris	confidential	$t_3$	0.3
Ellen	Field agent	Berlin	secret	$t_4$	0.2
Magdalen	Double agent	Paris	top secret	$t_5$	1.0
Nancy	HR director	Paris	restricted	$t_6$	0.8
Susan	Analyst	Berlin	secret	$t_7$	0.2

city	prov
New York	$t_1 \vee t_2$
Paris	$t_3 \vee t_5 \vee t_6$
Berlin	$t_4 \vee t_7$

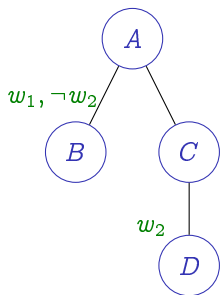
## Example of probability computation

name	position	city	classification	prov	prob
John	Director	New York	unclassified	$t_1$	0.5
Paul	Janitor	New York	restricted	$t_2$	0.7
Dave	Analyst	Paris	confidential	$t_3$	0.3
Ellen	Field agent	Berlin	secret	$t_4$	0.2
Magdalen	Double agent	Paris	top secret	$t_5$	1.0
Nancy	HR director	Paris	restricted	$t_6$	0.8
Susan	Analyst	Berlin	secret	$t_7$	0.2

city	prov	prob
New York	$t_1 \vee t_2$	$1 - (1 - 0.5) \times (1 - 0.7) = 0.85$
Paris	$t_3 \vee t_5 \vee t_6$	1.00
Berlin	$t_4 \vee t_7$	$1 - (1 - 0.2) \times (1 - 0.2) = 0.36$



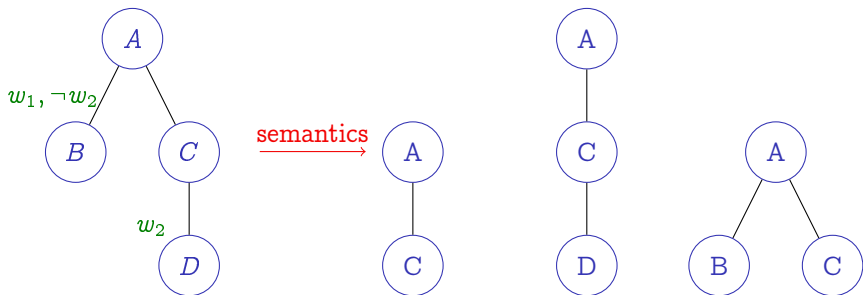
## Application: Probabilistic XML



Event	Prob.
$w_1$	0.8
$w_2$	0.7



## Application: Probabilistic XML



$$p_1 = 0.06$$

$$p_2 = 0.70$$

$$p_3 = 0.24$$

Event	Prob.
$w_1$	0.8
$w_2$	0.7



# Outline

Provenance

**Applications**

Probabilistic databases

**Views**

Explanation

Implementing Provenance Support

Conclusion



# Views

- Views are **named queries**
- They are used **in the same way as tables** within other queries
- **Semantics:** one **replaces** the view by the result of the evaluation of the corresponding query



## Virtual and materialized views

- A view may be **virtual** or **materialized**
- No **semantic** difference
- **Operational** difference, with an impact on the efficiency of query evaluation:

**virtual view:** the query defining the view is **evaluated each time** the view is used in a query

**materialized view:** the query defining the view is evaluated **when the view is created** and the result is stored in an auxiliary table; this table is directly used each time the view is used in another query



## Why using views?

**Logical independence:** an application can access views, without the need to know how data is effectively organized in the database (the organization can change in a transparent manner, by just redefining the views)

**Access control:** different access rights can be given to base tables and to views, so that a given user or application only has access to a restricted subset of the content of the database

**Data integration:** views can be defined to gather data from multiple sources with different schemas

**Optimization:** materialized views can be defined for frequent queries or subqueries, so that they do not need to be evaluated each time they are used



## Views and updates

Views interact in complex ways with updates (insertions, modifications, deletions).

- View maintenance:** when an update is performed on base tables, this update should be **reflected in the views**
- **Nothing to do** for virtual views
  - More complex for materialized views, that need to be **maintained** in terms of the updates



## Views and updates

Views interact in complex ways with updates (insertions, modifications, deletions).

- View maintenance:** when an update is performed on base tables, this update should be **reflected in the views**
- **Nothing to do** for virtual views
  - More complex for materialized views, that need to be **maintained** in terms of the updates
- View update:** one wants in some settings to perform an update directly on a view, which causes **appropriate updates on base tables**



## Views and updates

Views interact in complex ways with updates (insertions, modifications, deletions).

**View maintenance:** when an update is performed on base tables, this update should be **reflected in the views**

- **Nothing to do** for virtual views
- More complex for materialized views, that need to be **maintained** in terms of the updates

**View update:** one wants in some settings to perform an update directly on a view, which causes **appropriate updates on base tables**

How to do it? With provenance! At least for deletions



## View maintenance for deletions

- Just use **Boolean provenance**!
- Remove all tuples whose provenance annotation evaluates to  $\perp$

## View maintenance for deletions

- Just use **Boolean provenance**!
- Remove all tuples whose provenance annotation evaluates to  $\perp$

name	position	city	prov
John	Director	New York	$t_1$
Paul	Janitor	New York	$t_2$
Dave	Analyst	Paris	$t_3$
Ellen	Field agent	Berlin	$t_4$
Magdalen	Double agent	Paris	$t_5$
Nancy	HR director	Paris	$t_6$
Susan	Analyst	Berlin	$t_7$

city	prov
New York	$t_1 \wedge t_2$
Paris	$t_3 \wedge t_5 \vee t_3 \wedge t_6 \vee t_5 \wedge t_6$
Berlin	$t_4 \wedge t_7$

If  $t_1$  disappears

## View maintenance for deletions

- Just use **Boolean provenance**!
- Remove all tuples whose provenance annotation evaluates to  $\perp$

name	position	city	prov
John	Director	New York	$t_1$
Paul	Janitor	New York	$t_2$
Dave	Analyst	Paris	$t_3$
Ellen	Field agent	Berlin	$t_4$
Magdalen	Double agent	Paris	$t_5$
Nancy	HR director	Paris	$t_6$
Susan	Analyst	Berlin	$t_7$

city	prov
New York	$t_1 \wedge t_2$
Paris	$t_3 \wedge t_5 \vee t_3 \wedge t_6 \vee t_5 \wedge t_6$
Berlin	$t_4 \wedge t_7$

If  $t_1$  disappears, New York disappears from the result of the view.



## View update for deletions [Buneman et al., 2002]

- Use case for **Why-provenance!**
- To delete a tuple  $t$  in the result of a view, select a **minimal subset of tuples** (in terms of size, or in terms of side effects on other tuples of the deleted view) whose annotation appears in every set of annotations of the Why-provenance of  $t$
- **NP-complete** in general

## View update for deletions [Buneman et al., 2002]

- Use case for **Why-provenance!**
- To delete a tuple  $t$  in the result of a view, select a **minimal subset of tuples** (in terms of size, or in terms of side effects on other tuples of the deleted view) whose annotation appears in every set of annotations of the Why-provenance of  $t$
- **NP-complete** in general

name	position	city	prov
John	Director	New York	$t_1$
Paul	Janitor	New York	$t_2$
Dave	Analyst	Paris	$t_3$
Ellen	Field agent	Berlin	$t_4$
Magdalen	Double agent	Paris	$t_5$
Nancy	HR director	Paris	$t_6$
Susan	Analyst	Berlin	$t_7$

ville	prov
New York	$\{t_1, t_2\}$
Paris	$\{t_3, t_5\}, \{t_3, t_6\}, \{t_5, t_6\}$
Berlin	$\{t_4, t_7\}$

To delete Paris

## View update for deletions [Buneman et al., 2002]

- Use case for **Why-provenance!**
- To delete a tuple  $t$  in the result of a view, select a **minimal subset of tuples** (in terms of size, or in terms of side effects on other tuples of the deleted view) whose annotation appears in every set of annotations of the Why-provenance of  $t$
- **NP-complete** in general

name	position	city	prov
John	Director	New York	$t_1$
Paul	Janitor	New York	$t_2$
Dave	Analyst	Paris	$t_3$
Ellen	Field agent	Berlin	$t_4$
Magdalen	Double agent	Paris	$t_5$
Nancy	HR director	Paris	$t_6$
Susan	Analyst	Berlin	$t_7$

ville	prov
New York	$\{t_1, t_2\}$
Paris	$\{t_3, t_5\}, \{t_3, t_6\}, \{t_5, t_6\}$
Berlin	$\{t_4, t_7\}$

To delete Paris, delete **two tuples among  $t_3, t_5, t_6$** .



# Outline

Provenance

**Applications**

Probabilistic databases

Views

**Explanation**

Implementing Provenance Support

Conclusion



## Using provenance for explanation

- Semiring provenance can be used to provide a user with explanation on the query result:
  - How-provenance (provenance polynomials) explains precisely **how** a result has been computed: often too fine-grained
  - Why-provenance explains **why** a particular result is generated by providing combinations of tuples required for a tuple to be produced
- Provenance often too long and complex, (imperfect) **summarization** may be required [Ainy et al., 2015]
- Still far from a natural language explanation!
- **Why-not** provenance: why a result was **not** produced. Expressible with m-semirings, but requires dedicated techniques [Chapman and Jagadish, 2009] for compact explanations



## Where-provenance [Buneman et al., 2001]

- Different form of provenance: captures from which database **values** come which output **values**
- **Bipartite graph** of provenance: two attribute values are connected if one can be produced from the other
- Axiomatized in [Buneman et al., 2001, Cheney et al., 2009]
- **Cannot** be captured by provenance semirings [Cheney et al., 2009], because of renaming (does not keep track of relation attributes), projection (does not remember which attribute values still exist), join (in a join, an output value comes from two different input values)



# Outline

Provenance

Applications

**Implementing Provenance Support**

Representation Systems for Provenance

Systems

Conclusion



## Representation systems

- In the Boolean semiring, the counting semiring, the security semiring: provenance annotations are **elementary**
- In the Boolean function semiring, the universal semiring, etc., provenance annotations can become quite **complex**
- Needs for **compact representation** of provenance annotations
- Lower the **provenance computation complexity** as much as possible



## Provenance formulas

- Quite **straightforward**
- Formalism used in most of the provenance literature
- **PTIME** data complexity
- Expanding formulas (e.g., computing the monomials of a  $\mathbb{N}[\mathcal{X}]$  provenance annotation) can result in an **exponential blowup**

### Example

Is there a city with both an analyst and an agent, and if Paris is such a city, is there a director in the agency?

$$((t_3 \otimes t_5) \oplus (t_4 \otimes t_7)) \otimes ((t_3 \otimes t_5) \otimes t_1)$$

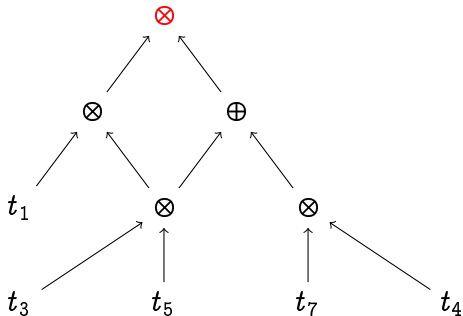


## Provenance circuits [Deutch et al., 2014, Amarilli et al., 2015]

- Use **arithmetic circuits** (Boolean circuits for Boolean provenance) to represent provenance
- Every time an operation reuses a previously computed result, link to the previously created circuit gate
- Allow **linear-time** data complexity of provenance computation when restricted to **bounded-treewidth databases** [Amarilli et al., 2015] (MSO queries for Boolean provenance, positive relational algebra queries for arbitrary semirings)
- Formulas can be **quadratically larger** than provenance circuits for MSO formulas, (log log)-larger for positive relational algebra queries [Wegener, 1987, Amarilli et al., 2016]



## Example provenance circuit





## OBDD and d-DNNF

- Various subclasses of **Boolean** circuits commonly used:
  - **OBDD**: Ordered Binary Decision Diagrams
  - **d-DNNF**: deterministic Decomposable Negation Normal Form
- **OBDDs** can be obtained in **P****TIME** data complexity on **bounded-treewidth databases** [Amarilli et al., 2016]
- **d-DNNFs** can be obtained in **linear-time** data complexity on **bounded-treewidth databases**
- **Application**: **probabilistic query evaluation** in **linear-time** data complexity on bounded-treewidth databases (d-DNNF evaluation is in linear-time)



## Provenance cycluits [Amarilli et al., 2017]

- Cycluit (cyclic circuit): arithmetic circuit with **cycles**
- Well-defined semantics on **some** semirings where infinite loops do not matter
- Allows computing provenance in **linear-time combined complexity** for **recursive** queries of a certain form (ICG-Datalog of bounded body size [Amarilli et al., 2017], capturing  $\alpha$ -acyclic conjunctive queries, 2RPQs, etc.), on bounded tree-width databases
- Related to **provenance equation systems** and formal series introduced in [Green et al., 2007]



# Outline

Provenance

Applications

**Implementing Provenance Support**

Representation Systems for Provenance  
Systems

Conclusion

## Desiderata for a provenance-aware DBMS

- Extends a **widely used** database management system
- **Easy to deploy**
- **Easy to use**, transparent for the user
- Provenance **automatically maintained** as the user interacts with the database management system
- Provenance computation **benefits from query optimization** within the DBMS
- Allow **probability computation** based on provenance
- **Any form of provenance** can be computed: Boolean provenance, semiring provenance in any semiring (possibly, with monus), aggregate provenance, where-provenance, **on demand**



## ProvSQL: Provenance within PostgreSQL (1/2)

[Senellart et al., 2018]

- **Lightweight** extension/plugin for PostgreSQL  $\geq 9.5$
- Provenance annotations stored as **UUIDs**, in an extra attribute of each provenance-aware relation
- A provenance circuit **relating UUIDs** of elementary provenance annotations and arithmetic gates stored in shared memory of the DBMS
- All computations done in the **universal semiring** (more precisely, with monus, in the free semiring with monus; for where-provenance, in a free term algebra)

## ProvSQL: Provenance within PostgreSQL (2/2)

[Senellart et al., 2018]

- **Query rewriting** to automatically compute output provenance attributes in terms of the query and input provenance attributes:
  - Duplicate elimination (DISTINCT, set union) results in aggregation of provenance values with  $\oplus$
  - Cross products, joins results in combination of provenance values with  $\otimes$
  - Difference rewritten in a join, with combination of provenance values with  $\ominus$
- Additional circuit gates on projection, join for support of **where-provenance**
- **Probability computation** from the provenance circuits, via various methods (naive, sampling, compilation to d-DNNFs)



## Challenges

- **Low-level** access to PostgreSQL data structures in extensions
- No simple **query rewriting** mechanism
- SQL is much **less clean** than the relational algebra
- **Multiset semantics** by default in SQL
- SQL is a very **rich language**, with many different ways of expressing the same thing
- Inherent **limitations**: e.g., no aggregation within recursive queries
- Implementing provenance computation should **not slow down** the computation
- User-defined functions, updates, etc.: **unclear** how provenance should work



## ProvSQL: Current status

- **Supported** SQL language features:
  - Regular SELECT-FROM-WHERE queries (aka conjunctive queries with multiset semantics)
  - JOIN queries (regular joins and outer joins; semijoins and antijoins are not currently supported)
  - SELECT queries with nested SELECT subqueries in the FROM clause
  - GROUP BY queries
  - SELECT DISTINCT queries (i.e., set semantics)
  - UNION's or UNION ALL's of SELECT queries
  - EXCEPT queries
  - Aggregate queries (terminal, for simple aggregates)
- Try it (and see a demo) from <https://github.com/PierreSenellart/provsql>



## Other databases with provenance management

- Older probabilistic database systems can compute some forms of provenance (especially, Boolean provenance); but tied to a specific version of PostgreSQL, **hard to deploy**

**Trio:** <http://infolab.stanford.edu/trio/>  
[Benjelloun et al., 2006]

**MayBMS:** <http://maybms.sourceforge.net/> [Huang et al., 2009]

- **Perm** <https://github.com/IITDBGroup/perm> [Glavic and Alonso, 2009] now **obsolete** system for provenance management; also tied to a specific version of PostgreSQL
- **GProM** <http://www.cs.iit.edu/~dbggroup/projects/gprom.html> [Arab et al., 2018] is similar to ProvSQL (though no probabilistic database capabilities), with some extra features; implemented as a **middleware**



# Outline

Provenance

Applications

Implementing Provenance Support

**Conclusion**



## Database Provenance [Senellart, 2017]

- Quite **rich foundations** of provenance management:
  - Different types of provenance
  - Semiring formalism to unify most provenance forms
  - (Partial) extensions for difference, recursive queries, aggregation, updates [Bourhis et al., 2020]; to other data models
  - Compact provenance representation formalisms
- Now is the time to work on **concrete, efficient, usable implementation**
- Now is the time to work with **actual users**, to adapt to actual needs of users who want to track the provenance of the data at a fine-grained level!

# Tutorial

<https://github.com/PierreSenellart/provsql/tree/master/doc/tutorial>

## Bibliography I

- Serge Abiteboul, Benny Kimelfeld, Yehoshua Sagiv, and Pierre Senellart. On the expressiveness of probabilistic XML models. *VLDB Journal*, 18(5):1041–1064, October 2009.
- Eleanor Ainy, Pierre Bourhis, Susan B. Davidson, Daniel Deutch, and Tova Milo. Approximated summarization of data provenance. In *CIKM*, 2015.
- Antoine Amarilli and Mikaël Monet. Example of a naturally ordered semiring which is not an m-semiring.  
<http://math.stackexchange.com/questions/1966858>, 2016.
- Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In *Proc. ICALP*, pages 56–68, Kyoto, Japan, July 2015.

## Bibliography II

- Antoine Amarilli, Pierre Bourhis, and Pierre Senellart.  
Tractable lineages on treelike instances: Limits and extensions. In *Proc. PODS*, pages 355–370, San Francisco, USA, June 2016.
- Antoine Amarilli, Pierre Bourhis, Mikaël Monet, and Pierre Senellart. Combined tractability of query evaluation via tree automata and cycluits. In *ICDT*, 2017.
- K. Amer. Equationally complete classes of commutative monoids with monus. *Algebra Universalis*, 18(1), 1984.
- Yael Amsterdamer, Daniel Deutch, and Val Tannen. On the limitations of provenance for queries with difference. In *TaPP*, 2011a.
- Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *PODS*, 2011b.

## Bibliography III

- Bahareh Sadat Arab, Su Feng, Boris Glavic, Seokki Lee, Xing Niu, and Qitian Zeng. GProM - A swiss army knife for your provenance needs. *IEEE Data Eng. Bull.*, 41(1):51–62, 2018.
- Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.
- Pierre Bourhis, Daniel Deutch, and Yuval Moskovitch. Equivalence-invariant algebraic provenance for hyperplane update queries. In *SIGMOD*, pages 415–429. ACM, 2020. doi: 10.1145/3318464.3380578. URL <https://doi.org/10.1145/3318464.3380578>.

## Bibliography IV

- Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*, 2001.
- Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. On propagation of deletions and annotations through views. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 150–158, 2002. doi: 10.1145/543613.543633. URL <http://doi.acm.org/10.1145/543613.543633>.
- Adriane Chapman and H. V. Jagadish. Why not? In *SIGMOD*, 2009.

## Bibliography V

- James Cheney, Laura Chiticariu, and Wang Chiew Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4), 2009.
- Carlos Viegas Damásio, Anastasia Analyti, and Grigoris Antoniou. Provenance for SPARQL queries. In *ISWC*, 2012.
- Susan B. Davidson, Sarah Cohen Boulakia, Anat Eyal, Bertram Ludäscher, Timothy M. McPhillips, Shawn Bowers, Manish Kumar Anand, and Juliana Freire. Provenance in scientific workflow systems. *IEEE Data Eng. Bull.*, 30(4): 44–50, 2007. URL <http://sites.computer.org/debull/A07dec/susan.pdf>.
- Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. Circuits for Datalog provenance. In *ICDT*, 2014.

## Bibliography VI

- Robert Fink, Larisa Han, and Dan Olteanu. Aggregation in probabilistic databases via knowledge compilation. *Proceedings of the VLDB Endowment*, 5(5):490–501, 2012.
- J. Nathan Foster, Todd J. Green, and Val Tannen. Annotated XML: queries and provenance. In *PODS*, 2008.
- Floris Geerts and Antonella Poggi. On database query languages for k-relations. *J. Applied Logic*, 8(2), 2010.
- Floris Geerts, Thomas Unger, Grigoris Karvounarakis, Irini Fundulaki, and Vassilis Christophides. Algebraic structures for capturing the provenance of SPARQL queries. *J. ACM*, 63(1), 2016.
- Boris Glavic and Gustavo Alonso. Perm: Processing provenance and data on the same data model through query rewriting. In *ICDE*, pages 174–185, 2009.

## Bibliography VII

- Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. *IEEE Data Eng. Bull.*, 29(1), 2006.
- Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, 2007.
- Jiewen Huang, Lyublena Antova, Christoph Koch, and Dan Olteanu. MayBMS: a probabilistic database management system. In *SIGMOD*, pages 1071–1074, 2009.
- Tomasz Imieliński and Jr. Lipski, Witold. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.
- Yann Ramusat, Silviu Maniu, and Pierre Senellart. Semiring provenance over graph databases. In *TaPP*, 2018.
- Pierre Senellart. Provenance and probabilities in relational databases: From theory to practice. *SIGMOD Record*, 46(4), December 2017.

## Bibliography VIII

- Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. ProvSQL: provenance and probability management in postgresql. In *VLDB*, 2018. Demonstration.
- Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool, 2011.
- Ingo Wegener. *The Complexity of Boolean Functions*. Wiley, 1987.