CS594

Provenance & Explanations

2 - Provenance Models

Course webpage

Boris Glavic

bglavic@uic.edu

Provenance Models

Overview

Black-Box Provenance Models & Requirements for Provenance

Excursion - Relational Algebra

Provenance Models For Relational Queries

Provenance Applications & Querying Provenance

Overview

Overview

- What is a Provenance Model?

Black-Box Provenance Models & Requirements for Provenance

Excursion - Relational Algebra

Provenance Models For Relational Queries

Provenance Applications & Querying Provenance

Overview

Overview

- What is a Provenance Model?

What is a Provenance Model?

- For this section of the course, a **provenance model** enables us to determine mechanically which **data dependencies** (`wasDerivedFrom`) hold for a computation
  - For now we will assume that computations take in
  - later we will generalize this
- **Black Box Models**
  - treat the computation as a black box, we can only test data dependencies by feeding inputs into the computation
  - declarative (state what properties the provenance should fulfill)
  - can be applied to any type of computation
- **White Box Models**
  - have knowledge about the computation $\Rightarrow$ specific to particular computations
  - often more efficient by exploiting properties of the computation

Black Box Models

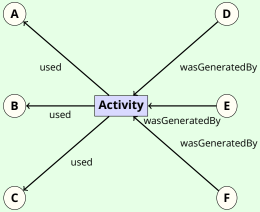
- **Assumption:**
  - **Computation** $\mathcal{C}$ is a function $\mathcal{C}(I) = O$
  - **Input**: a set $I = \{i_1, \ldots, i_n\}$
  - **Output**: a set $O = \{o_1, \ldots, o_m\}$
- Determine $Prov(\mathcal{C}, I, o) \subseteq I$, the subset of inputs that contribute to $o \in O$
  1. What conditions should $Prov(\mathcal{C}, I, o)$ fulfill?
  2. How can we test these conditions by evaluating $\mathcal{C}$ over different inputs I'?

Relational Database Queries

- Computation is a **query** Q
  - Input is a **database** D which is a **set of tuples** $\{t_1, \ldots, t_n\}$
  - Output is a **table** $Q(D)$ which a **set of tuples** $\{o_1, \ldots, o_m\}$
- $Prov(Q, D, t) \subseteq D$ are the tuples from D that contribute to $t \in Q(D)$

What Do We Want From a Provenance Model?

- **What do we want from** $Prov(\mathcal{C}, I, o)$**?**
  - Include every input $i \in I$ that is **needed** to produce o
  - Exclude every input $i \in I$ that is **irrelevant** for producing o
- **What are the right declarative requirements to enforce this?**



Black-Box Provenance Models & Requirements for Provenance

Overview

Black-Box Provenance Models & Requirements for Provenance

- Agenda
- Sufficiency & Minimality
- Causality
- Recap

Excursion - Relational Algebra

Black-Box Provenance Models & Requirements for Provenance

Black-Box Provenance Models & Requirements for Provenance

- Agenda
- Sufficiency & Minimality
- Causality
- Recap

Agenda

- Come up with **requirements for provenance** that are **"testable"**
- Reason about the **computational cost** of testing these conditions
- We will often reason about this for queries over relational databases where tables are sets of rows
  - The concepts, however, are applicable to any computation on sets of elements!

Example

	name
t_1	Peter
t_2	Bob
t_3	Alice

```
SELECT DISTINCT name FROM student;
```

	name	major	gpa
s_1	Peter	CS	3.9
s_2	Peter	BIO	3.6
s_3	Peter	Law	2.5
s_4	Bob	CS	4.0
s_5	Alice	CS	3.5

Provenance

- Subsets of D that are enough to produce a result tuple (test by running Q)
- $\{s_1\}$ is enough to produce t_1
- $\{s_2\}$ is enough to produce t_1
- $\{s_1, s_2, s_3, s_4\}$ is enough to produce t_1

Black-Box Provenance Models & Requirements for Provenance

Sufficiency

- Test whether $w \subseteq D$ is enough for producing a result tuple t by running Q on w and testing whether the result contains t
  - If **yes**, then apparently w contains sufficient information for computing t through Q

Definition (Sufficiency)

Given a query Q, database D, and tuple $t \in Q(D)$, a set of tuples $w \subseteq D$ is **sufficient** for producing t iff:

$$t \in Q(w)$$

If w is sufficient, we call it a **witness** for t

Example Sufficiency and Witnesses

	name
t_1	Peter
t_2	Bob
t_3	Alice

```
SELECT DISTINCT name FROM student;
```

	name	major	gpa
s_1	Peter	CS	3.9
s_2	Peter	BIO	3.6
s_3	Peter	Law	2.5
s_4	Bob	CS	4.0
s_5	Alice	CS	3.5

Witnesses

- Subsets of D that are enough to produce a result tuple (test by running Q)
- $\s_1\, \s_2\,$ $\{s_3\}$, $\{s_1, s_2, s_3\}$, $\{s_1, s_4\}$, D are all witnesses for t_1
- $\{s_4\}$, $\{s_3, s_4\}$, D are witnesses for t_2
- $\{s_5\}$, $\{s_1, s_2, s_3, s_5\}$, D are witnesses for t_3

Monotone Queries

- An important property of queries: if we insert new data to the input database, then the query returns more results
- Our running example query is monotone

Definition (Monotone Queries)

A query Q is monotone iff:

$$\forall D_1 \subseteq D_2 : Q(D_1) \subseteq Q(D_2)$$

Witnesses for Monotone Queries

Lemma (Sufficiency closed under monotonicity)

Let Q be a monotone query and D a database. Consider $w \subset w' \subseteq D$,

- *If $w \subseteq D$ is sufficient for $t \in Q(D)$ then w' is also sufficient*

Remarks

- D is always a trivial witness!
- Witnesses may include irrelevant data!
- Witnesses are guaranteed to include irrelevant data for monotone queries

Minimality

- How to prune irrelevant inputs from witnesses?
- If we can remove a tuple from a witness and the result is still sufficient, then this tuple was apparently irrelevant and can be removed

Definition (Minimal Witnesses)

A witness $w \subseteq D$ for a tuple t wrt. a query Q and database D is **minimal**, if there does **not exist** a witness $w' \subset w$

Example Minimal Witnesses

	name
t_1	Peter
t_2	Bob
t_3	Alice

```
SELECT DISTINCT name FROM student;
```

	name	major	gpa
s_1	Peter	CS	3.9
s_2	Peter	BIO	3.6
s_3	Peter	Law	2.5
s_4	Bob	CS	4.0
s_5	Alice	CS	3.5

Witnesses

- Minimal witnesses highlighted in red
- $\{s_1\}$, $\{s_2\}$, $\{s_3\}$, $\{s_1, s_2, s_3\}$, $\{s_1, s_4\}$, D are all witnesses for t_1
- $\{s_4\}$, $\{s_3, s_4\}$, D are witnesses for t_2
- $\{s_5\}$, $\{s_1, s_2, s_3, s_5\}$, D are witnesses for t_3

Computational Complexity

- If we do not have any information about the query Q then we have to test all subsets of D
- If $|D| = n$ then there are 2^n subsets of D which each could potentially be witnesses
- Assume that $|Q(D)| = m$
- For each result tuple $t \in Q(D)$ we have to test 2^n candidate witnesses in worst-case
- If we have k witnesses then we can identify minimal witnesses in $O(k^2 \cdot |D|)$ time by comparing every witness w with every other witness w'

Lemma (Complexity of computing all (minimal) witnesses)

The computational complexity of computing all (minimal) witnesses is $O(m \cdot (2^n)^2 \cdot cost(Q))$ where $cost(Q)$ is the time of running Q

Black-Box Provenance Models & Requirements for Provenance

Black-Box Provenance Models & Requirements for Provenance

- Agenda
- Sufficiency & Minimality
- Causality
- Recap

Intuition

- Intuitively, we can test whether an input tuple *s* is **needed** to produce a result tuple *t* through a query *Q* by:
  1. removing *s* from *D* and reevaluating *Q* over the modified input
  2. if *t* is still in the result then apparently it was **necessary** for producing the result

Counterfactual Causes

Definition (Counterfactual Cause)

Given query Q, database D, and tuple $t \in Q(D)$, a tuple $s \in D$ is a **counterfactual cause** for t if:

$$t \notin Q(D - \{s\})$$

Example Counterfactual Causes

	name
t_1	Peter
t_2	Bob
t_3	Alice

```
SELECT DISTINCT name FROM student;
```

	name	major	gpa
s_1	Peter	CS	3.9
s_2	Peter	BIO	3.6
s_3	Peter	Law	2.5
s_4	Bob	CS	4.0
s_5	Alice	CS	3.5

Counterfactual Causes

- $\{s_4\}$ for t_2
- $\{s_5\}$ for t_3
- **There are no counter-factual causes for t_1**

Limitations of Counterfactual Causes

- As shown in the previous example
- Counterfactual causes fail if there are alternative ways to derive a result tuple
  - **that means that there are multiple minimal witnesses**

Actual Causes

- **Intuition**: delete all alternative witnesses until only one remains that is then counterfactual

Definition (Actual Cause)

Given query Q, database D, and tuple $t \in Q(D)$, a tuple $s \in D$ is a **counterfactual cause** for t if here exists $\Gamma \subseteq D - \{s\}$, called a **contingency**, such that:

1. $t \in Q(D - \Gamma)$
2. $t \notin Q(D - \Gamma - \{s\})$

Remarks

- Any counterfactual cause is an actual cause (by setting $\Gamma = \emptyset$)

Example Actual Causes

	name
t_1	Peter
t_2	Bob
t_3	Alice

```
SELECT DISTINCT name FROM student;
```

	name	major	gpa
s_1	Peter	CS	3.9
s_2	Peter	BIO	3.6
s_3	Peter	Law	2.5
s_4	Bob	CS	4.0
s_5	Alice	CS	3.5

Actual Causes

- $\{s_4\}$ for t_2 with $\Gamma = \emptyset$
- $\{s_5\}$ for t_3 with $\Gamma = \emptyset$
- t_1
  - $\{s_1\}$ with $\Gamma = \{s_2, s_3\}$
  - $\{s_2\}$ with $\Gamma = \{s_1, s_3\}$
  - $\{s_3\}$ with $\Gamma = \{s_1, s_2\}$

Black-Box Provenance Models & Requirements for Provenance

Literature References

Surveys on Provenance Models

- [Gla21]
- [CCT09]

Sufficiency and Witnesses

- [CWW00]
- [BKT01]

Causality

- [MGMS10]

Recap

- Declarative notions of
  - **minimal witnesses**
  - **actual causes**
- Can be applied to **any computation** the consumes and returns **sets**
- **Exponential complexity!**
  - not practical

Excursion - Relational Algebra

Overview

Black-Box Provenance Models & Requirements for Provenance

Excursion - Relational Algebra
- Relational Algebra
- Extended Relational Algebra
- Incompleteness

Provenance Models For Relational Queries

Excursion - Relational Algebra

Properties

- **Procedural, set-oriented language**
- **Operators** are functions from relations to relations
  - **input**: 0 or more relations
  - **output**: 1 relation
  - **closed** language: outputs are of the same type as inputs (*relations*) $\rightarrow$ **composition**
- **Pure**: no side-effects
- An **algebra** over relations

Standard relational algebra

- Seven basic operators
  - **Table access**: R
  - **Selection**: σ_θ
  - **Projection**: π_A
  - **Union**: $\cup$
  - **Set difference**: $-$
  - **Cross product**: $\times$
  - **Renaming**: ρ

Excursion: Set Comprehension

- We will use the concept of **set comprehension** to define the semantics of relational algebra operators

Definition (Set Comprehension)

A comprehension $\{e \mid \phi(e)\}$ where $\phi(e)$ is a Boolean condition over variable *e* define a set containing all elements *e* such that $\phi(e)$ evaluates to true.

Excursion: Set Comprehension

- We will use the concept of **set comprehension** to define the semantics of relational algebra operators

Definition (Set Comprehension)

A comprehension $\{e \mid \phi(e)\}$ where $\phi(e)$ is a Boolean condition over variable *e* define a set containing all elements *e* such that $\phi(e)$ evaluates to true.

Examples

- $\{n \mid n \in \mathbb{N} \land n < 3\} = ?$
- $\{(n, m) \mid n, m \in \mathbb{N} \land n + m = 5\} = ?$

Excursion: Set Comprehension

- We will use the concept of **set comprehension** to define the semantics of relational algebra operators

Definition (Set Comprehension)

A comprehension $\{e \mid \phi(e)\}$ where $\phi(e)$ is a Boolean condition over variable e define a set containing all elements e such that $\phi(e)$ evaluates to true.

Examples

- $\{n \mid n \in \mathbb{N} \wedge n < 3\} = \{1, 2\}$
- $\{(n, m) \mid n, m \in \mathbb{N} \wedge n + m = 5\} = \{(1, 4), (2, 3), (3, 2), (4, 1)\}$

Table Access

Intuition

- Return the content of relation R

Definition (Syntax)

- **Table Access** R

Definition (Semantics)

Given a relational algebra expression R and database D:

$$R_D = \{t \mid t \in R\}$$

Table Access - Example

Example Expression

persons

Input

name	salary	age
Gertrud	24,000	34
Sanjiv	65,000	45
Alice	55,000	38
Sudeepa	90,000	39

Output

name	salary	age
Gertrud	24,000	34
Sanjiv	65,000	45
Alice	55,000	38
Sudeepa	90,000	39

Selection

Intuition

- Filter out rows that do not fulfill condition θ

Definition (Syntax)

- **Selection** $\sigma_\theta(R)$
- θ is a Boolean condition constructed from
  - constants (e.g., 1, `Peter`, `2023-01-01`, ...)
  - attribute references (e.g., `a`, `item`, `name`, ...)
  - arithmetic expressions (e.g., 1, a, $(a + 10) * 2$)
  - comparisons between arithmetic expressions (e.g., $a < 3$, $a + 1 < 2 * b$)
  - logical operators: $\wedge$ (and), $\vee$ (or), $\neg$ (not)

Selection

Definition (Semantics)

Given a relational algebra expression $\sigma_\theta(R)$ and database D:

$$\sigma_{\theta D} = \{t \mid t \in R_D \wedge t \models \theta\}$$

Selection - Example

Example Expression

$$\sigma_{salary > 50{,}000 \wedge age < 40}(persons)$$

Input

name	salary	age
Gertrud	24,000	34
Sanjiv	65,000	45
Alice	55,000	38
Sudeepa	90,000	39

Output

name	salary	age
Alice	55,000	38
Sudeepa	90,000	39

Projection

Intuition

- For each row only keep attributes from A

Definition (Syntax)

- **Projection** $\pi_A(R)$
- $A = (a_1, \ldots, a_n)$ is a list of attributes from R
  - attributes cannot appear more than once in A

Projection

Definition (Semantics)

Given a relational algebra expression $\pi_A(R)$ and database D:

$$\pi_{A}{}_{D} = \{t.A \mid t \in R_D\}$$

where $t.A$ denotes the restriction of tuple t to attributes from A

Projection - Example

Example Expression

$$\pi_{age,salary}(persons)$$

Input

name	salary	age
Gertrud	24,000	34
Sanjiv	65,000	45
Alice	55,000	38
Sudeepa	90,000	39

Output

age	salary
34	24,000
45	65,000
38	55,000
39	90,000

Union

Intuition

- Combine the rows from tables R and S into one table

Definition (Syntax)

- **Union** $R \cup S$
- R and S have to have the same **arity** (number of attributes)
- also same types

Definition (Semantics)

Given a relational algebra expression $R \cup S$ and database D:

$$R \cup S_D = \{t \mid t \in R_D \vee t \in S_D\}$$

Union - Example

Example Expression

$$customer \cup employee$$

Input

customer

name	salary	age
Gertrud	24,000	34
Sanjiv	65,000	45

employee

name	salary	age
Alice	55,000	38
Sudeepa	90,000	39

Output

name	salary	age
Gertrud	24,000	34
Sanjiv	65,000	45
Alice	55,000	38
Sudeepa	90,000	39

Set Difference

Intuition

- Return all rows from R that do not exist in S

Definition (Syntax)

- **Set difference** $R - S$
- R and S have to have the same **arity** (number of attributes)
- also same types

Definition (Semantics)

Given a relational algebra expression $R - S$ and database D:

$$R - S_D = \{t \mid t \in R_D \wedge \neg t \in S_D\}$$

Set Difference - Example

Example Expression

$student - instructor$

Input

student

name	department
Gertrud	CS
Sanjiv	CS
Jun	BIO

instructor

name	department
Sanjiv	CS
Sudeepa	BIO

Output

name	department
Gertrud	CS
Jun	BIO

Cross product

Intuition

- Return the concatenation of each row from R with each row from S

Definition (Syntax)

- **Cross product** $R \times S$
- $\text{Sch}(R) \cap \text{Sch}(S) = \emptyset$ (no common attribute names)

Cross product

Definition (Semantics)

Given a relational algebra expression $R \times S$ and database D:

$$R \cup S_D = \{r \circ s \mid r \in R_D \wedge s \in S_D\}$$

where $\circ$ denotes concatenation of tuples $r = (c_1, \ldots, c_n)$ and $s = (d_1, \ldots, d_m)$:

$$r \circ s = (c_1, \ldots, c_n, d_1, \ldots, d_m)$$

Cross product - Example

Example Expression

$$year \times month$$

Input

year

year
2022
2023

month

month	name
01	Jan
02	Feb

Output

year	month	name
2022	01	Jan
2022	02	Feb
2023	01	Jan
2023	02	Feb

Renaming

Intuition

- Return the input relation with new attribute names

Definition (Syntax)

- **Rename** $\rho_B(R)$
- $B = (b_1, \ldots, b_n)$ is a list of attributes with the same arity as $\mathbf{R}(a_1, \ldots, a_n)$
  - attributes cannot appear more than once in B

Renaming

Definition (Semantics)

Given a relational algebra expression $\sigma_\theta(R)$ and database D:

$$\rho_{B} D = \{t[b_1 \leftarrow a_1, \ldots, b_n \leftarrow a_n] \mid t \in R_D\}$$

Here $t[b \leftarrow a]$ renames attribute a to b in tuple t

Notational convenience

- If we want to only rename some attributes we will use $\rho_{b_i \leftarrow a_i, \ldots}$ to denote renaming where all attributes not explicitly mentioned are assumed to not be renamed

Renaming - Example

Example Expression

$$\rho_{lastname,salary,howold}(persons)$$

Input

name	salary	age
Gertrud	24,000	34
Sanjiv	65,000	45
Alice	55,000	38
Sudeepa	90,000	39

Output

lastname	salary	howold
Gertrud	24,000	34
Sanjiv	65,000	45
Alice	55,000	38
Sudeepa	90,000	39

Combining Operators

- Each operator is quite simple and of limited expressiveness
- The power of relational algebra stems from combining operators

Return instructors older than 40 that are not students

$$\pi_{name}(\sigma_{age>40}(instructor)) - \pi_{name}(student)$$

Input

instructor

name	age
Fatima	45
Rohit	35
Luis	50

student

name	major
Fatima	CS
Nattawut	BIO
Rohit	CS

Output

name
Luis

Sharing Subexpressions

- To simplify writing of complex queries we will allow for modularization by giving subqueries a name using $Name \leftarrow Query$

Assignment

$$q_1 \leftarrow person \bowtie_{addr=aid} address$$
$$q \leftarrow q_1 \cup q_1$$

Excursion - Relational Algebra

Excursion - Relational Algebra

- Relational Algebra
- Extended Relational Algebra
- Incompleteness

Limited Expressive Power

- There are certain queries that we cannot express using the operators we have discussed so far:
  - *How many rows are in the student table?*
  - *Return a particular tuples independent of the database content*
  - *For each row in the employee table return* `income - tax`

Additional Operators

Adding expressive power

- **Constant relation**
- **Aggregation** with **group-by** γ
- **Generalized projection**

Additional Operators

Syntactic sugar

- Operators that can be expressed using the standard relational algebra operators
- **Natural join** and **Theta join** $\bowtie$
- **Relational division** $\div$
- **Intersection** $\cap$
- **Outer joins** ⟗, ⟕, ⟖
- **Semi join** and **Anti-join**

Constant Relation

Intuition

- Return a fixed table

Definition (Syntax)

- **Constant Relation** $\{t_1, \ldots, t_n\}_{(a_1,\ldots,a_m)}$
- $(a_1, \ldots, a_m)$ defines the attribute names for the result relation
- each t_i is expected to be a tuple over $(a_1, \ldots, a_m)$

Definition (Semantics)

Given a relational algebra expression $\{t_1, \ldots, t_n\}_{(a_1,\ldots,a_m)}$:

$$\{t_1, \ldots, t_n\}_{(a_1,\ldots,a_m)_D} = \{t_1, \ldots, t_n\}$$

Constant Relation - Example

Example Expression

$$\{(Peter, 30), (Bob, 45)\}_{(name,age)}$$

Input

Output

name	age
Peter	30
Bob	45

Aggregation Functions

- An **aggregration function** f takes a **set of values** and returns a **single value**
  - for convenience we will all aggregation functions to take a set of tuples with a single attribute
- Aggregation functions we consider here:
  - $\textbf{count}(v_1, ..., v_n) = n$
  - $\textbf{sum}(v_1, \ldots, v_n) = \sum_{i=1}^{n} v_i$
  - $\textbf{min}(v_1, \ldots, v_n) = v_i$ such that $\forall j \in [1, n] : v_i \leq v_j$
  - $\textbf{max}(v_1, \ldots, v_n) = v_i$ such that $\forall j \in [1, n] : v_i \geq v_j$
  - $\textbf{avg}(v_1, \ldots, v_n) = \frac{\textbf{sum}(v_1, \ldots, v_n)}{\textbf{count}(v_1, \ldots, v_n)}$

Aggregation Function Examples

- $S = \{1, 10, 15, 25\}$
- $\textbf{count}(S) = 4$
- $\textbf{sum}(S) = 51$
- $\textbf{min}(S) = 1$
- $\textbf{max}(S) = 25$
- $\textbf{avg}(S) = 12.75$

Aggregation Functions on Empty Inputs

- Consider an input set $\emptyset$:
  - — $\textbf{count}(\emptyset) = 0$
  - — $\textbf{sum}(\emptyset) = \textbf{null}$
  - — $\textbf{min}(\emptyset) = \textbf{null}$
  - — $\textbf{max}(\emptyset) = \textbf{null}$
  - — $\textbf{avg}(\emptyset) = \textbf{null}$

Aggregation

Intuition

- **without group-by**: compute an aggregation function over all values in a column
- **with group-by**: group rows based on their group-by attributes and compute the aggregation function for each group of tuples

Definition (Syntax)

- **Aggregation** $\gamma_{f(a);G}(R)$
- $a \in \text{Sch}(R)$
- f is an aggregation function (one of **sum**, **avg**, **count**, **min**, **max**)
  - aggregation functions take a set of values and return a single value
- G is a list of **group-by** attributes ($G = \emptyset$ is allowed)

Aggregation Semantics w/o Group-by

Definition (Semantics - aggregation w/o group-by)

Given a relational algebra expression $\gamma_{f(a)}(R)$ and database D:

$$\gamma_{f(a)}(R)_D = \{(f(\pi_a(R)_D))\}$$

- Aggregation returns a single row even if the input relation is the empty set
- The attribute storing the result of the aggregation function $f(a)$ is named `f(a)`

Aggregation Semantics With Group-by

Definition (Semantics - aggregation with group-by)

Given a relational algebra expression $\gamma_{f(a);G}(R)$ and database D:

$$\gamma_{f(a);G}(R)_D = \{(f(Group(R, G, t))) \circ t.G \mid t \in R\}$$
$$Group(R, G, t) = \{t' \mid t' \in R \wedge t.G = t'.G\}$$

Tuple concatenation

$t \circ t'$ denotes the concatenation of tuples, i.e.,

$$(a_1, \ldots, a_n) \circ (b_1, \ldots, b_m) = (a_1, \ldots, a_n, b_1, \ldots, b_m)$$

Aggregation Example (w/o Group-By)

Example Expression

$$\gamma_{\mathbf{sum}(salary)}(persons)$$

Input

name	salary	age
Gertrud	24,000	30
Sanjiv	65,000	30
Alice	55,000	40
Arthur	100,000	40
Sudeepa	90,000	40

Output

sum(salary)
334,000

Aggregation Example (w/o Group-By)

Example Expression

$$\gamma_{\mathbf{sum}(salary)}(persons)$$

Input

name	salary	age

Output

sum(salary)
null

Aggregation Example (Group-by)

Example Expression

$$\gamma_{age;\mathbf{sum}(salary)}(persons)$$

Input

name	salary	age
Gertrud	24,000	30
Sanjiv	65,000	30
Sudeepa	90,000	40
Jose	100,000	40
Alice	55,000	40

Output

age	salary
30	89,000
40	245,000

Aggregation (Multiple Functions)

- We will allow aggregation to compute multiple aggregation functions at once

Multiple aggregation functions

$$\gamma_{dept;\mathbf{sum}(salary),\mathbf{avg}(tax)}(employee)$$

No extra expressive power

- This does not add any expressive power
- We can rewrite this into individual aggregations + join

Generalized Projection

Intuition

- Allow for arithmetic expressions in projection

Definition (Syntax)

- **Projection** $\pi_A(R)$
- $A = (e_1, \ldots, e_n)$ is a list of expressions:
  - **basic expressions:**
    - an attribute $a \in \mathbf{R}$
    - a constant c
  - **composite expressions:**
    - $e_1 \diamond e_n$ for arithmetic operator $\diamond$ (e.g., $+$, or $\cdot$)
    - $e_1 \diamond e_n$ where $\diamond$ is a comparison operator (e.g., $<$ or $\geq$)
    - $e_1 \wedge e_2$, $e_1 \vee e_2$, $\neg e$

Generalized Projection

Definition (Semantics)

Given a relational algebra expression $\sigma_\theta(R)$ and database D:

$$\pi_{A D} = \{(e_1(t), \ldots, e_n(t)) \mid t \in R_D\}$$

Result Schema

- Expressions are used as attribute names, e.g., $\pi_{salary - tax}(person)$ has a single attribute named `salary - tax`

Generalized Projection - Example

Example Expression

$$\pi_{name, salary - tax + bonus}(employee)$$

Input

name	salary	tax	bonus
Gertrud	24,000	3,500	0
Sanjiv	65,000	4,700	10,000

Output

name	salary
Gertrud	20,500
Sanjiv	70,300

Query Equivalence

Definition (Query Equivalence)

Two queries Q_1 and Q_2 are equivalent, written as $Q_1 \equiv Q_2$ iff:

$$\forall D : Q_1(D) = Q_2(D)$$

- Two queries are equivalent if they return the **same result** over **every** database
- Equivalently, they encode the same function

Natural Join

Intuition

- Join two tables on equality of common attributes

Definition (Syntax)

- **Natural Join** $R \bowtie S$

Definition (Semantics)

Given a relational algebra expression $R \bowtie S$ and database D, let $O = \text{Sch}(S) - \text{Sch}(R)$ and $C = \text{Sch}(R) \cap \text{Sch}(S) = (a_1, \ldots, a_n)$ and $C' = ({a_1}', \ldots, {a_n}')$

$$R \bowtie S \equiv \pi_{R,O}(\sigma_{\bigwedge_{a \in C} a = a'}(R \times \rho_{C',O}(S)))$$

Natural Join - Example

Example Expression

$$president \bowtie provost$$

Input

president

year	president
2023	Bob
2024	Alice

provost

year	provost
2023	Les
2024	Joe

Output

year	president	provost
2023	Bob	Les
2024	Alice	Joe

Natural Join

No common attributes

- It is permissible to natural join two relations that do not share any common attributes
- This is a cross product!

Theta Join

Intuition

- Join tuples on a condition θ.

Definition (Syntax)

- **Theta Join** $R \bowtie_\theta S$
- $\text{Sch}(R) \cap \text{Sch}(S) = \emptyset$ (no common attribute names)

Definition (Semantics)

Given a relational algebra expression $R \bowtie_\theta S$ and database D:

$$R \bowtie_\theta S \equiv \sigma_\theta(R \times S)$$

Theta Join - Example

Example Expression

$$\pi_{name,manager}(\pi_{name,manager,salary}(employee) \bowtie_{manager=man \land salary>mansalary} (\rho_{man,mansalary}(\pi_{name,salary}(employee))))$$

Input

employee

name	manager	salary
Lin	null	60,000
Faizan	Lin	50,000
Saeed	Lin	100,000

Output

name	manager
Saeed	Lin

Semi-join

Intuition

- Return all tuples from R that join with at least one tuple from S.

Definition (Syntax)

- **Semi-join** $R \rhd_\theta S$
- $\text{Sch}(R) \cap \text{Sch}(S) = \emptyset$ (no common attribute names)

Definition (Semantics)

$$R \rhd_\theta S \equiv \pi_{\text{Sch}(R)}(R \bowtie_\theta S)$$

Semi-join - Example

Example Expression

$$student \rhd_{name=name'} \rho_{name',course}(takes)$$

Input

student

name
Gertrud
Sanjiv
Alice
Sudeepa

takes

name	course
Gertrud	CS480
Sanjiv	CS480
Sanjiv	CS430

Output

name
Gertrud
Sanjiv

Natural Semi-join

Natural Semi-join

- If no condition θ is provided, then the semi join will be assumed to be a natural join
- In this case we join on equality of the common attributes
- We still only return attributes from the left input

Anti-join

Intuition

- Return all tuples from R that do not join with any tuple from S.

Definition (Syntax)

- **Anti-join** $R \blacktriangleright_\theta S$

Definition (Semantics)

$$R \blacktriangleright_\theta S \equiv R - (R \rhd_\theta S)$$

Anti-join - Example

Example Expression

$$student \blacktriangleright_{name=name'} \rho_{name',course}(takes)$$

Input

student

name
Gertrud
Sanjiv
Alice
Sudeepa

takes

name	course
Gertrud	CS480
Sanjiv	CS480
Sanjiv	CS430

Output

name
Alice
Sudeepa

Outer Joins

Intuition

- Like a regular join but retain tuples form one or both sides that do not have join partners. Tuples that do not have join partners are padded with null values.

Definition (Syntax)

- **Left Outer Join** $R ⟕_\theta S$
- **Right Outer Join** $R ⟖_\theta S$
- **Full Outer Join** $R ⟗_\theta S$
- $\text{Sch}(R) \cap \text{Sch}(S) = \emptyset$ (no common attribute names)

Outer Joins

Definition (Semantics)

Consider R and S with $|\,\text{Sch}(R)\,| = n$ and $|\,\text{Sch}(S)\,| = m$

$$R \mathbin{⟕}_\theta S \equiv (R \bowtie_\theta S) \cup ((R - (R \rhd_\theta S)) \times \{(\underbrace{\textbf{null}, \ldots, \textbf{null}}_{m \text{ times}})\})$$

$$R \mathbin{⟖}_\theta S \equiv (R \bowtie_\theta S) \cup (\{(\underbrace{\textbf{null}, \ldots, \textbf{null}}_{n \text{ times}})\} \times (S - (S \rhd_\theta R)))$$

$$R \mathbin{⟗}_\theta S \equiv (R \bowtie_\theta S) \cup ((R - (R \rhd_\theta S)) \times \{(\underbrace{\textbf{null}, \ldots, \textbf{null}}_{m \text{ times}})\})$$

$$\cup\, (\{(\underbrace{\textbf{null}, \ldots, \textbf{null}}_{n \text{ times}})\} \times (S - (S \rhd_\theta R)))$$

Left Outer Joins - Example

Example Expression

$$\pi_{name,city}(person ⟕_{address=aid}\ address)$$

Input

person

name	address
Peter	NULL
Bob	1

address

aid	city
1	Chicago
2	New York

Output

name	city
Peter	NULL
Bob	Chicago

Right Outer Joins - Example

Example Expression

$$\pi_{name,city}(person \rtimes_{address=aid} address)$$

Input

person

name	address
Peter	NULL
Bob	1

address

aid	city
1	Chicago
2	New York

Output

name	city
Bob	Chicago
NULL	New York

Full Outer Joins - Example

Example Expression

$$\pi_{name,city}(person ⟗_{address=aid} address)$$

Input

person

name	address
Peter	NULL
Bob	1

address

aid	city
1	Chicago
2	New York

Output

name	city
Peter	NULL
Bob	Chicago
NULL	New York

Intersection

Intuition

- Return all tuples that exist in both R and S.

Definition (Syntax)

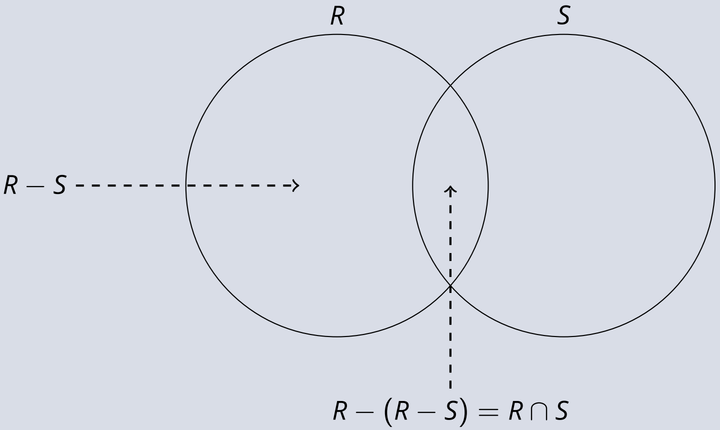
- **Intersection** $R \cap S$
- R and S have to have the same **arity** (number of attributes)
- also same types

Definition (Semantics)

Given a relational algebra expression $R \cap S$ and database D:

$$R \cap S \equiv R - (S - R)$$

Intersection



Intersection - Example

Example Expression

customer ∩ employee

Input

customer

name	salary	age
Gertrud	24,000	34
Sanjiv	65,000	45

employee

name	salary	age
Gertrud	24,000	34
Alice	55,000	38
Sudeepa	90,000	39

Output

name	salary	age
Gertrud	24,000	34

Division

Intuition

- The maximal T such that $T \times S \subseteq R$
  - compare integer division $n \div m$ is the largest u such that $u \cdot m \leq n$
- For the attributes O only in R find tuples t such that **all** combinations of $t.O$ with tuples from S exist in R
  - this is a type of **universal quantification**

Division

Definition (Syntax)

- **Division** $R \div S$
  - $\text{Sch}(S) \subset \text{Sch}(R)$

Definition (Semantics)

Given a relational algebra expression $R \div S$ and database D. Let $U = \text{Sch}(R) - \text{Sch}(S)$.

$$E_1 \leftarrow \pi_U(R)$$
$$E_2 \leftarrow \pi_U((E_1 \times S) - \pi_{U,\text{Sch}(S)}(R \bowtie S))$$
$$R \div S \equiv E_1 - E_2$$

Division - Example

Example Expression

$$takes \div course$$

Input

takes

student	course
Bob	CS480
Bob	CS100
Alice	CS480

course

course
CS480
CS100

Output

student
Bob

Excursion - Relational Algebra

Excursion - Relational Algebra

- Relational Algebra
- Extended Relational Algebra
- Incompleteness

Missing Values

- Real information is often **incomplete**
  - The information may not be available at all
  - The information may be too expensive to obtain
  - We may not have gotten the information yet
- **Incomplete Databases** are a principled way to model missing or uncertain information
- The relational model as defined by Codd and implemented in database system only has very limited support for incompleteness
- We will have a brief look at the general model to understand the limitation of the relational model according to Codd and to SQL

Incomplete Databases

Definition (Incomplete Database)

A incomplete database $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of deterministic databases D_i called possible worlds

Intuition

- Each possible world represents one possible true state of the world, but we do not know which world correctly represents the real world

Incomplete Database Example

Incomplete Database

D_1

name	age	salary
Peter	30	60,000
Alice	40	90,000
Bob	40	100,000

D_2

name	age	salary
Peter	31	70,000
Alice	40	90,000

Nulls for Incomplete Information

- In the relational model, null values are used to model incompleteness
- A **null value** means that we have complete uncertainty about what value is the correct value for a tuple's attribute
  - any domain value is considered possible
- A database with null values encodes an incomplete database where each possible world is generated from the database by replacing each null value with a value from the attribute's domain

Database with Nulls Example

name	is-graduate	active
Peter	NULL	1
Bob	1	NULL

D_1

name	is-graduate	active
Peter	0	1
Bob	1	0

D_3

name	is-graduate	active
Peter	1	1
Bob	1	0

D_2

name	is-graduate	active
Peter	0	1
Bob	1	1

D_4

name	is-graduate	active
Peter	1	1
Bob	1	1

Limited Expressive Power of Databases with Nulls

- Databases with nulls are not powerful enough to express all types of incompleteness
- We **can not express**:
  - Correlations between missing values (*e.g., Peter and Bob both work on the same unknown project*)
  - Restrictions of allowable values (*e.g., we do not know Peter's salary but it is either 70k or 71k*)
  - Uncertainty about tuple existence (/e.g., Peter may or may not exist)

Nulls & Three-valued Logic

- *How to we deal with the incompleteness encoded by null values?*
- Redefine arithmetic operations and comparisons with null

Definition (Operations with Null)

$$c \diamond \textbf{null} = \textbf{null} \qquad (\diamond \in \{+, \cdot, <, =, \leq, \dots\})$$

Definition (Logical Operators and Null)

OR

AND

NOT

textcolorwhitetextbffalse	
textcolorwhitetextbftrue	

textcolorwhitetextbffalse	
textcolorwhitetextbftrue	

Relational Algebra Operators over Databases with Nulls

Selection & join

- Filter rows where $\theta(t) = \mathit{false}$ or $\theta(t) = \textbf{null}$

Aggregation

- Null values are ignored in aggregation
- For group-by values nulls are treated like actual values
  - *e.g., there may be a group* $(\textbf{null}, 3)$

Inconsistencies Arising From Three-valued Logic

- **Tautologies fail**
  - `A = A` does not return true if $A =$ **null**
- **Certain rows are missed**
  - Query results that exist in every possible world may not be returned
- **Impossible rows may be returned**
  - Query results that are impossible (are not returned in any world) may be returned

Omitting Certain Rows

$$\sigma_{isalive=isalive}(beings)$$

Input

beings

name	isalive
Schroedinger's cat	NULL

D_1

name	isalive
Schroedinger's cat	0

D_2

name	isalive
Schroedinger's cat	1

Output

name	isalive

Returning Impossible Rows

$$beings - \sigma_{isalive=isalive}(beings)$$

Input

beings

name	isalive
Schr. cat	NULL

D_1

name	isalive
Schr. cat	0

D_2

name	isalive
Schr. cat	1

Output

name	isalive
Schr. cat	NULL

Provenance Models For Relational Queries

Provenance Models For Relational Queries

- Provenance Models For Relational Queries
  - Agenda
  - Why Provenance
  - Provenance Polynomials
  - Beyond Positive Relational Algebra

Agenda

- we now will consider specific classes of queries
- we will develop provenance models which can be computed efficiently

Provenance Models For Relational Queries

Witness Sets

- Recall our declarative notion of a **witness** and **minimal witness**

Definition (Witness Sets)

Given a query Q, database D, and tuple $t \in Q(D)$, we introduce the following notation:

- $Wit(Q, D, t) = \{w \mid w \subseteq D \wedge t \in Q(w)\}$
- $MWit(Q, D, t) = \{w \mid w \in Wit(Q, D, t) \wedge \neg \exists w' \subset w : w' \in Wit(Q, D, t)\}$

Positive Relational Algebra

- For **Why provenance** we will restrict our attention to **positive relational algebra** $\mathbb{RA}^+$ queries which are all queries that only use:
  - Selection σ, Projection Π, Union $\cup$, Cross Product $\times$ and renaming ρ
- We will provide a recursive definition that expresses the witnesses for the result of a relational algebra operator based on the witnesses for the operators input
  - The **why-provenance** of a query result is then computed top-down starting with the top-most operator of a query and finishes at the leaves (table accesses)

Why Provenance

Definition (Why Provenance)

Let Q be an $\mathcal{RA}^+$ query, D a database, and $t \in Q(D)$. $\text{Why}(Q, D, t)$ is:

$$\text{Why}(\text{R}, D, t) = \begin{cases} \{\{t\}\} & \text{if } t \in R \\ \emptyset & \text{otherwise} \end{cases} \qquad \text{Why}(\rho_{AB}(Q), D, t) = \text{Why}(Q, D, t.AB)$$

$$\text{Why}(\sigma_\theta(Q), D, t) = \begin{cases} \text{Why}(Q, D, t) & \text{if } t \models \theta \\ \emptyset & \text{otherwise} \end{cases} \qquad \text{Why}(\pi_A(Q), D, t) = \bigcup_{u \in Q(D): u.A = t} \text{Why}(Q, D, u)$$

$$\text{Why}(Q_1 \bowtie Q_2, D, t) = \{D' \cup D'' \mid D' \in \text{Why}(Q, D, t.Q_1) \wedge D'' \in \text{Why}(Q, D, t.Q_2)\}$$

$$\text{Why}(Q_1 \cup Q_2, D, t) = \text{Why}(Q_1, D, t) \cup \text{Why}(Q_2, D, t)$$

Minimal Why Provenance

Definition (Minimal Why Provenance)

$$\text{MWhy}(Q, D, t) = \{w \mid w \in \text{Why}(Q, D, t) \wedge \neg w' \subset w : w' \in \text{Why}(Q, D, t)$$

Lemma (Minimal Why Provenance Contains All Minimal Witnesses)

$$\text{MWhy}(Q, D, t) = MWit(Q, D, t)$$

Syntax Independence

- Declarative models have the beneficial property that **equivalent queries have the same provenance**
  - — provenance only depends on how the query behaves not on how it is written
- For models that are based on specific query syntax this is not necessarily true

Definition (Syntax Independence)

A provenance model $\mathcal{P}$ is **syntax independent** if for any two queries $Q_1 \equiv Q_2$ we have that:

$$\forall D : \forall t \in Q_1(D) : \mathcal{P}(Q_1, D, t) = \mathcal{P}(Q_2, D, t)$$

Why Provenance and Syntax Dependence

Theorem (Syntax Independence of Minimal Why-provenance)

- *Minimal Why-provenance is* ***syntax independent***
- *Why-provenance is* ***syntax dependent***

- $Q_1 = \pi_A(R) \equiv \pi_A(R) \bowtie \pi_A(R) = Q_2$

R

A	B	
1	1	r_1
1	2	r_2

Query result

A	Why	MWhy
1	$\{\{r_1\},\{r_2\},\{r_1,r_2\}\}$	$\{r_1\},\{r_2\}\}$

□

Discussion

- **Why provenance distinguishes between conjunctive and disjunctive use**
  - Each witness *w* is a set of tuples that are together sufficient for producing the result (**conjunctive**)
  - Multiple witnesses model alternative ways to derive a tuple (**disjunctive**)
- **Why provenance works for set semantics, but not for bags**
  - Minimization can lead to incorrect results under bag semantics
  - Defining provenance as sets of tuples does not work well with bags
- **Why provenance is computed top-down**

Provenance Models For Relational Queries

Bag Semantics

- so far we have focused only on **set semantics**
- SQL databases (almost) exclusively use bag semantics
- Under **bag semantics** (**multisets**) a relation can contain multiple copies of a tuple
  - The **multiplicity** of a tuple is the number of duplicates of the tuple

A	B
1	1
1	1
1	1
1	2

- $(1, 1)$: multiplicity 3
- $(1, 2)$: multiplicity 1

Equivalence in Bag and Set Semantics

- Queries that are equivalent under set semantics may not be equivalent under bag semantics

- $Q_1 = R$ is equivalent to $Q_2 = R \bowtie R$ under set semantics
- Q_1 and Q_2 are not equivalent under bag semantics

R

A	B
1	1
1	1

Q_1

A	B
1	1
1	1

Q_2

A	B
1	1
1	1
1	1
1	1

Agenda - K-relations

- We want a provenance model that **works for both bags and sets**
- For our provenance model to be syntax independent it should have exactly the **same equivalences as regular query semantics**
- Allows us to **track how tuples are combined** by a query to derive a result
- Can express other models like minimal why provenance

Annotation to the Rescue

- **Annotations** associate data with additional metadata
  - Comments from users
  - Trust annotations
  - Provenance
  - ...
- **we will annotate tuples with provenance**

Semirings

- Use elements from a **semiring** as tuple annotations

Definition (Semiring)

A **semiring** over a set K is a structure $\mathcal{K} = (K, \oplus_{\mathcal{K}}, \otimes_{\mathcal{K}}, 0_{\mathcal{K}}, 1_{\mathcal{K}})$ where $\oplus_{\mathcal{K}} : K \times K \to K$ and $\otimes_{\mathcal{K}} : K \times K \to K$ are binary operations and $0_{\mathcal{K}}$ and $1_{\mathcal{K}}$ are elements from K. $\mathcal{K}$ has to obey the algebraic laws shown on the next slide.

K-relations

Definition (K-relations)

Consider a be a universal domain of values $\mathcal{U}$. An n-ary **$\mathcal{K}$-relation** R is a function $\mathcal{U}^n \rightarrow K$ such that the set $\{t \mid t \in \mathcal{U}^n \wedge R(t) \neq 0\}$ is finite.

Tuples That Do Not Exist

- K-relations are total functions (every possible tuple gets an annotation)
- Non-existing tuples are annotated with $0_{\mathcal{K}}$
- **Convention**: do not explicitly list tuples annotated with $0_{\mathcal{K}}$

K-relations Examples

- **Natural Numbers** ($\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$: bag semantics by annotating each tuple with its multiplicity
- **Boolean Semiring** $\mathbb{B} = (\{\top, \bot\}, \vee, \wedge, \bot, \top)$: Tuples that exist are annotated with $\top$ and tuples that do not with $\bot$.
- **Possible Worlds Semiring** $\mathbb{W} = (2^{W}, \cup, \cap, \emptyset, W)$: W denotes the set of possible worlds. Each tuple is annotated with the set of worlds it appears in.

K-relations Examples (cont.)

Sets ($\mathbb{B}$)

Student	Activity	
A	hike	$\top$
B	tennis	$\top$
C	tennis	$\top$

Bags ($\mathbb{N}$)

Student	Activity	
a	hike	2
b	tennis	3
b	tennis	4

Semirings Laws

$$k_1 \oplus_{\mathcal{K}} k_2 = k_2 \oplus_{\mathcal{K}} k_1 \qquad \textbf{(commutativity of addition)}$$

$$k_1 \otimes_{\mathcal{K}} k_2 = k_2 \otimes_{\mathcal{K}} k_1 \qquad \textbf{(commutativity of multiplication)}$$

$$(k_1 \oplus_{\mathcal{K}} k_2) \oplus_{\mathcal{K}} k_3 = k_1 \oplus_{\mathcal{K}} (k_2 \oplus_{\mathcal{K}} k_3) \qquad \textbf{(associativity of addition)}$$

$$(k_1 \otimes_{\mathcal{K}} k_2) \otimes_{\mathcal{K}} k_3 = k_1 \otimes_{\mathcal{K}} (k_2 \otimes_{\mathcal{K}} k_3) \qquad \textbf{(associativity of multiplication)}$$

$$k_1 \oplus_{\mathcal{K}} 0_{\mathcal{K}} = k_1 \qquad \textbf{(neutral element of addition)}$$

$$k_1 \otimes_{\mathcal{K}} 1_{\mathcal{K}} = k_1 \qquad \textbf{(neutral element of multiplication)}$$

$$k_1 \otimes_{\mathcal{K}} 0_{\mathcal{K}} = 0_{\mathcal{K}} \qquad \textbf{(annihilation by zero)}$$

$$k_1 \otimes_{\mathcal{K}} (k_2 \oplus_{\mathcal{K}} k_3) = (k_1 \otimes_{\mathcal{K}} k_2) \oplus_{\mathcal{K}} (k_1 \otimes_{\mathcal{K}} k_3)$$

(**multiplication distributes over addition**)

Queries over K-relations

- Queries returns the same tuples as under set semantics
-

Definition ($\mathcal{RA}^+$ Query Semantics)

- **Rename**: $\rho_{A \leftarrow B}(R)(t) = R(t[B \leftarrow A])$
- **Projection**: $\pi_U(R)(t) = \sum_{t=t'[U]} R(t')$
- **Selection**: $\sigma_\theta(R)(t) = R(t) \otimes_{\mathcal{K}} \theta(t)$
- **Natural Join**: $(R_1 \bowtie R_2)(t) = R_1(t[\mathbf{R}_1]) \otimes_{\mathcal{K}} R_2(t[\mathbf{R}_2])$
- **Union**: $(R_1 \cup R_2)(t) = R_1(t) \oplus_{\mathcal{K}} R_2(t)$

Provenance Semirings

Positive Boolean Algebra Semiring

- $\text{PosBool}[X] = (\text{PosBool}[X], \vee, \wedge, \perp, \top)$:

The elements of the PosBool[X] are positive boolean formulas over a set of variables X

- **minimal why-provenance**
- **same equivalences as set semantics**

Why-provenance Semiring

- $\text{Why}[X] = (2^{2^X}, \cup, \Cup, \emptyset, \{\emptyset\})$:
  - $k_1 \Cup k_2 = \{w_1 \cup w_2 \mid w_1 \in k_1 \wedge w_2 \in k_2\}$
- **why provenance**

Provenance Semirings (cont)

Which-provenance Semiring

- $\text{Which}[X] = (2^X \cup \{\bot\}, \cup_+, \cup_\times, \bot, \emptyset)$:
  - Operations $\cup_+$ and $\cup_\times$ are set union, but
  - $k_1 \cup_+ \bot = \bot \cup_+ k_1 = k_1$
  - $k_1 \cup_\times \bot = \bot \cup_\times k_1 = \bot$
- **lineage**

Provenance Polynomials Semiring

- $\mathbb{N}[X] = (\mathbb{N}[X], +, \times, 0, 1)$
  - Polynomials with integer coefficients over variables X
- **the right provenance model for bag semantics**
- **same equivalences as bag semantics**

Provenance Polynomial Intuition

- **Provenance polynomials** record **how** input annotations where **combined** to derive an output annotations
  - only fulfills the equivalences needed to be a semiring
- Works for **every** semiring

K-relational Queries Example - Sets

- **query** $Q = \pi_A(R \bowtie S)$

R

A	B	C	
1	1	1	$\top$
1	2	1	$\top$

S

B	D	
1	3	$\top$
1	4	$\top$
2	3	$\top$

Q

A	
1	$(\top \wedge \top) \vee (\top \wedge \top) \vee (\top \wedge \top) = \top$

K-relational Queries Example - Bags

- **query** $Q = \pi_A(R \bowtie S)$

R

A	B	C	
1	1	1	2
1	2	1	5

S

B	D	
1	3	1
1	4	2
2	3	3

Q

A	
1	$(2 \cdot 1) + (2 \cdot 2) + (5 \cdot 3) = 21$

K-relational Queries Example - Minimal Why

- **query** $Q = \pi_A(R \bowtie S)$

R

A	B	C	
1	1	1	x_1
1	2	1	x_2

S

B	D	
1	3	y_1
1	4	y_2
2	3	y_3

Q

A	
1	$(x_1 \wedge y_1) \vee (x_1 \wedge y_2) \vee (x_2 \wedge y_3)$

K-relational Queries Example - Provenance Polynomials

- **query** $Q = \pi_A(R \bowtie S)$

R

A	B	C	
1	1	1	x_1
1	2	1	x_2

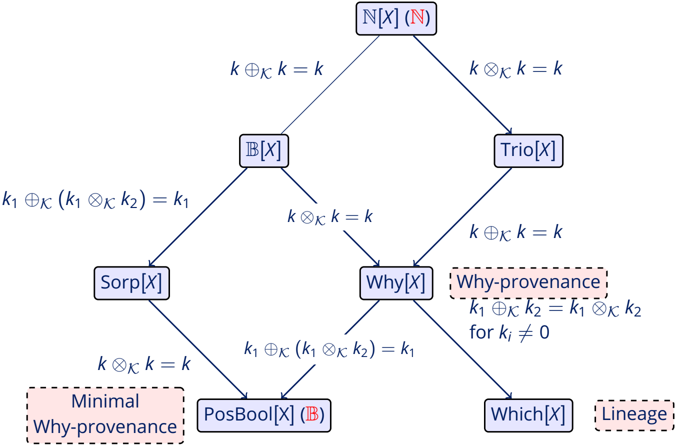
S

B	D	
1	3	y_1
1	4	y_2
2	3	y_3

Q

A	
1	$(x_1 \cdot y_1) + (x_1 \cdot y_2) + (x_2 \cdot y_3)$

Semirings and Query Equivalence



Homomorphisms

- **homomorphisms**: technical tool to understand the relationship between semirings and prove the generality of provenance polynomials

Definition (Homomorphism)

Let $\mathcal{K}_1$ and $\mathcal{K}_2$ be semirings, a mapping $h : K_1 \to K_2$ is a homomorphism if for all $k_1, k_2 \in K_1$, we have:

$$h(k_1 \oplus_{\mathcal{K}_1} k_2) = h(k_1) \oplus_{\mathcal{K}_2} h(k_2)$$
$$h(k_1 \otimes_{\mathcal{K}_1} k_2) = h(k_1) \otimes_{\mathcal{K}_2} h(k_2)$$
$$h(0_{\mathcal{K}_1}) = 0_{\mathcal{K}_2}$$
$$h(1_{\mathcal{K}_1}) = 1_{\mathcal{K}_2}$$

Homomorphisms on K-relations

- Consider a semiring homomorphism: $h : \mathcal{K}_1 \rightarrow \mathcal{K}_2$, we define its application to a $\mathcal{K}_1$ relation R by applying it to the annotation of very tuple:

$$h(R)(t) = h(R(t))$$

Homomorphism on K-relations - Example

- $h_1 : \mathbb{N} \rightarrow \mathbb{B}$

$$h_1(k) = \begin{cases} \bot & \text{if k = 0} \\ \top & \text{otherwise} \end{cases}$$

- $Q = \pi_A(R)$

$Q(D)$

A	
1	3

R

A	B	
1	a	2
1	b	1

$h(Q(D)) = Q(h(D))$

A	
1	$\top = h_1(3) = \top \vee \top$

h(R)

A	B	
1	a	$\top = h_1(2)$
1	b	$\top = h_1(1)$

Homomorphisms Commute with Queries

- As relational algebra over K-relations is defined based on the semiring operations, it follows homomorphisms commute with queries

Theorem (Homomorphism commute with queries)

Consider a $\mathcal{K}_1$ database D and query Q and a homomorphism $h : \mathcal{K}_1 \rightarrow \mathcal{K}_2$:

$$h(Q(D)) = Q(h(D))$$

Homomorphisms and Expressiveness

- Homomorphisms can "delete" information, but can never generate new information
- Consider two provenance semirings $\mathcal{K}_1$ and $\mathcal{K}_2$ that consists of symbolic expressions over variables X (e.g., provenance polynomials)
  - If there exist a semiring homomorphism $\mathcal{K}_1 \rightarrow \mathcal{K}_2$ then $\mathcal{K}_1$ tracks more information (see [Gre11])

Homomorphisms and Deletions

Lemma (Deleting tuples is a homomorphism)

Consider a $\mathbb{N}[X]$ relation where each tuple t_i is annotated with a unique variable x_i from set X. Consider a subset $Y \subseteq X$, then h_{del} as defined below is a semiring homomorphism.

$$h_{del}(x) = \begin{cases} 0 & \text{if } x \in Y \\ x & \text{otherwise} \end{cases}$$

Implications

- Given the provenance polynomials for the result of a query over a database D, we can determine the correct provenance polynomials for any database $D' \subset D$ by applying h_{del} to these polynomials!

Provenance Polynomials and "Computability"

- As provenance polynomials track semiring computations in a generic way, we can evaluate any supported query Q over an $\mathbb{N}[X]$ database where every tuple is annotated with a unique variable x_i and derive the query results for any $\mathcal{K}$ database with the same support by:
  1. Design a homomorphism $Eval_\mu$ that assigns to each variable x_i an annotation from $\mathcal{K}$
  2. Apply $Eval_\mu$ to the query result

Generality of Provenance Polynomials

- Provenance polynomials track the provenance of queries for any $\mathcal{K}$ database for any semiring $\mathcal{K}$

Provenance Polynomials and "Computability"

Definition (Computability)

We say a provenance model has the **computability** property if from the provenance of a query Q over database D we can reconstruct $Q(D)$.

Theorem (Computability of Provenance Polynomials)

Provenance polynomials have the computability property for K-relations any semiring $\mathcal{K}$.

Provenance Models For Relational Queries

Queries With Negation

- So far we have only considered **positive** relational algebra
  - — all queries are **monotone**
- We assumed that provenance is **transitive**
- Introducing negation leads to new challenges
  - — the absence of tuples can be required to produce a result
  - — transitivity breaks down

Negation and Transitivity - Counterexample

- $Q_1 = R - Q_2$ and $Q_2 = S - T$
- $w_2 = \{s_1\}$ is a witness for Q_2
- $w_1 = \{r_1\}$ is a witness for Q_1

R

A	
1	r_1

S

B	
1	s_1

T

A	
1	t_1

Q_2

WHITEB

Q_1

WHITEA

Provenance Applications & Querying Provenance

Overview

Black-Box Provenance Models & Requirements for Provenance

Excursion - Relational Algebra

Provenance Models For Relational Queries

Provenance Applications & Querying Provenance

Applications & Requirements

Provenance Applications & Querying Provenance

Provenance Applications & Querying Provenance

- Applications & Requirements
- Provenance for Debugging
- Querying Provenance

Provenance Applications & Querying Provenance

Provenance Applications & Querying Provenance

- Applications & Requirements
- Provenance for Debugging
- Querying Provenance

Debugging Computations

Forward Tracing

Backward Tracing

Provenance Applications & Querying Provenance

Provenance Applications & Querying Provenance

- Applications & Requirements
- Provenance for Debugging
- Querying Provenance

Backward Provenance Queries

Forward Provenance Queries

References

Overview

Black-Box Provenance Models & Requirements for Provenance

Excursion - Relational Algebra

Provenance Models For Relational Queries

Provenance Applications & Querying Provenance

References

References

- References

References I

[BKT01] 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*, pages 316–330, 2001.

[CCT09] 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.

[CWW00] Yingwei Cui, Jennifer Widom, and Janet L. Wiener.
Tracing the Lineage of View Data in a Warehousing Environment.
ACM Transactions on Database Systems (TODS), 25(2):179–227, 2000.

[Gla21] Boris Glavic.
Data provenance - origins, applications, algorithms, and models.
Foundations and Trends® in Databases, 9(3-4):209–441, 2021.

[Gre11] T.J. Green.
Containment of conjunctive queries on annotated relations.
Theory of Computing Systems, 49(2):429–459, 2011.

[MGMS10] A. Meliou, W. Gatterbauer, K.F. Moore, and D. Suciu.
The Complexity of Causality and Responsibility for Query Answers and non-Answers.
Proceedings of the VLDB Endowment, 4(1):34–45, 2010.