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

I. Database Provenance

I.1 Provenance Models and Systems

Boris Glavic

October 31, 2012

Outline

1 Provenance Storage
- Introduction
- Compression Methods for Provenance
- Index Structures for Provenance
- Recap

Storing Provenance

Why and When to Store Provenance

- During transformation execution
  - Temporarily materialize provenance
- Store provenance to preserve it
  - Main consideration: storage size
- To support queries
  - Main consideration: effective access

ILLINOIS INSTITUTE OF TECHNOLOGY

Challenges for Storing Provenance

- Size of provenance information
  - Can outgrow original + result data
- Datamodel mismatch between data + its provenance
  - E.g., provenance could be formula, relationship, tree
  - For data that is set of tuples
- Queries over provenance
  - Need efficient access
  - Access patterns provenance $\neq$ access patterns data

ILLINOIS INSTITUTE OF TECHNOLOGY

Size of Provenance Information

Determined by

- Provenance model
  - Set of witnesses: $O(2^N)$ subsets of instance
  - Many models $O(n^2)$ (relation)
- Data granularity
  - Tuples
  - Attribute values
- Transformation granularity
  - Transaction
  - Query
  - methods below access intermediate results
  - SQL-Block
  - Algebra Operators

Provenance Model

Model Independent View

- Transformation has n inputs and m outputs
- Provenance models which inputs are responsible for an output in result of transformation
- $\Rightarrow$Size of provenance in $O(n \times m)$

Model Specific

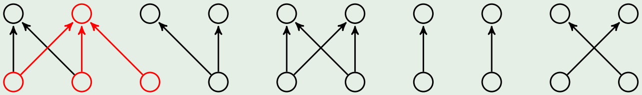
- Additional Storage Requirements
  - Irrelevance
    - Set of witnesses: include data not in input
    - Lineage: set difference
  - Modelling Alternatives
    - Inputs replicated in multiple alternatives
  - Data Independent Parts of Model
    - Provenance Polynomials: $+$ and $\times$ operators

GY

Model Independent Storage Requirements

- Each of the m outputs
- depends on up to all of the n inputs

Example



Irrelevance

- Two types of irrelevance
  - Data not in input
    - E.g., set of witnesses: witness for $R \cup S$ can include tuples from T
  - Non-contributing input data
    - E.g., why-provenance of $\pi_{A,B}(R \bowtie_{a=a'} \pi_{a \rightarrow a'}(R))$: r_1 joins with itself and r_2.

Alternatives in Provenance

- Output tuples may have alterative derivations
- . . . alternatives may repeat same tuple
- ⇒a fixed limit based on query

Example

```
SELECT a FROM R
UNION
SELECT b FROM R
UNION
SELECT c FROM R
```

R

	a	b	c
r_1	1	1	1
r_2	2	3	2
r_3	5	1	1

Q

	a
t_1	1
t_2	2
t_3	3
t_4	5

Alternatives in Provenance

- Output tuples may have alterative derivations
- ... alternatives may repeat same tuple
- $\Rightarrow$a fixed limit based on query

Example

```
SELECT a FROM R
UNION
SELECT b FROM R
UNION
SELECT c FROM R
```

$$\mathbb{N}[X](q, t_1) = 3 \times r_1 + 2 \times r_3$$

R

	a	b	c
r_1	1	1	1
r_2	2	3	2
r_3	5	1	1

Q

	a
t_1	1
t_2	2
t_3	3
t_4	5

Alternatives in Provenance

- Output tuples may have alterative derivations
- ... alternatives may repeat same tuple
- $\Rightarrow$a fixed limit based on query

Example

Each tuple may appear up to three times in provenance!

$$\mathbb{N}[X](q, t_1) = 3 \times r_1 + 2 \times r_3$$

R

	a	b	c
r_1	1	1	1
r_2	2	3	2
r_3	5	1	1

Q

	a
t_1	1
t_2	2
t_3	3
t_4	5

ILLINOIS INSTITUTE OF TECHNOLOGY

Alternatives in Provenance

- Output tuples may have alterative derivations
- . . . alternatives may repeat same tuple
- $\Rightarrow$a fixed limit based on query

Alternatives in Provenance

- Output tuples may have alterative derivations
- ...alternatives may repeat same tuple
- $\Rightarrow$a fixed limit based on query

Example

$$reachable(a, b) :- edge(a, c) \wedge reachable(c, b)$$
$$reachable(a, b) :- edge(a, b)$$

ILLINOIS INSTITUTE OF TECHNOLOGY

Alternatives in Provenance

- Output tuples may have alterative derivations
- ... alternatives may repeat same tuple
- $\Rightarrow$not for recursive queries!

Example

No fixed limit for query, but for query and data

$$reachable(a, b) :- edge(a, c) \wedge reachable(c, b)$$
$$reachable(a, b) :- edge(a, b)$$

ILLINOIS INSTITUTE OF TECHNOLOGY

Data Granularity

Tuples

- Output tuples T_A
- Instance tuples T_{DB}
- Input tuples T_I
- $\Rightarrow$depends on model $T_A \times T_I$ or $T_A \times T_{DB}$
- $\Rightarrow +$ factor of alternatives

ILLINOIS INSTITUTE OF TECHNOLOGY

Data Granularity

Tuples

- Output tuples T_A
- Instance tuples T_{DB}
- Input tuples T_I
- $\Rightarrow$depends on model $T_A \times T_I$ or $T_A \times T_{DB}$
- $\Rightarrow$+ factor of alternatives

Attribute Values

- E.g., A_O output attributes
- A_I total input attributes
- Additional growth factor is $A_O \times A_I$

GY

Transaction Granularity

For SQL queries

- Track provenance for
  - Transactions
  - Queries
  - SQL query blocks
  - Single operators
  - Individual expressions

Example

```
BEGIN TRANSACTION
INSERT INTO persons (SELECT name, salary
                     FROM employee, pay
                     WHERE (SSN = employee));
UPDATE persons SET salary = salary + 1000
       WHERE job = 'consultant';
COMMIT
```

Transaction Granularity

Example

Transaction: provenance is tuples in database state before transaction execution

```
BEGIN TRANSACTION
INSERT INTO persons (SELECT name, salary
                     FROM employee, pay
                     WHERE (SSN = employee));
UPDATE persons SET salary = salary + 1000
       WHERE job = 'consultant';
COMMIT
```

ILLINOIS INSTITUTE OF TECHNOLOGY

Transaction Granularity

Example

Queries/Statements: E.g., provenance of tuple in persons relation after INSERT

```
BEGIN TRANSACTION
INSERT INTO persons (SELECT name, salary
                     FROM employee, pay
                     WHERE (SSN = employee));
UPDATE persons SET salary = salary + 1000
       WHERE job = 'consultant';
COMMIT
```

ILLINOIS INSTITUTE OF TECHNOLOGY

Transaction Granularity

Example

SQL query block: Track for each individual SELECT-FROM-WHERE-... block

```
BEGIN TRANSACTION
INSERT INTO persons (SELECT name, salary
                     FROM employee, pay
                     WHERE (SSN = employee));
UPDATE persons SET salary = salary + 1000
       WHERE job = 'consultant';
COMMIT
```

ILLINOIS INSTITUTE OF TECHNOLOGY

Transaction Granularity

Example

Single operators: E.g., join and projection in `INSERT`

```
BEGIN TRANSACTION
INSERT INTO persons (SELECT name, salary
                    FROM employee, pay
                    WHERE (SSN = employee));
UPDATE persons SET salary = salary + 1000
       WHERE job = 'consultant';
COMMIT
```

ILLINOIS INSTITUTE OF TECHNOLOGY

Transaction Granularity

Example

Individual expressions: E.g., provenance $a = b$ in WHERE condition $a = b \wedge c > 5$

```
BEGIN TRANSACTION
INSERT INTO persons (SELECT name, salary
                    FROM employee, pay
                    WHERE (SSN = employee));
UPDATE persons SET salary = salary + 1000
       WHERE job = 'consultant';
COMMIT
```

ILLINOIS INSTITUTE OF TECHNOLOGY

Example Transformation Granularity

Example

```
CREATE VIEW SalesTotal AS
SELECT Location AS Shop, Month, SSN AS Employee,
       Price * Amount AS Totalprice
FROM Employee E, Shop H, Item I, Sales S
WHERE E.WorksFor = H.Location
      AND E.SSN = S.Employee
      AND I.Id = S.Item
```

Example Transformation Granularity

Example

```
CREATE VIEW MonthlyRevenue
SELECT Shop, Month, sum(Totalprice) AS Revenue
FROM SalesTotal
GROUP BY Shop, Month
```

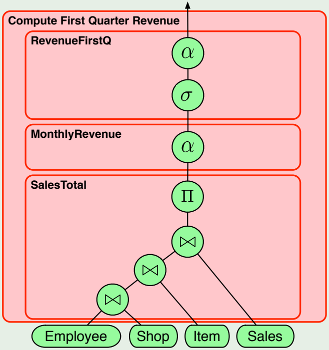
Example Transformation Granularity

Example

```sql
CREATE VIEW RevenueFirstQ
SELECT Shop, sum(Revenue) AS Revenue
FROM MonthlyRevenue
WHERE Month < 5
GROUP BY Shop
```

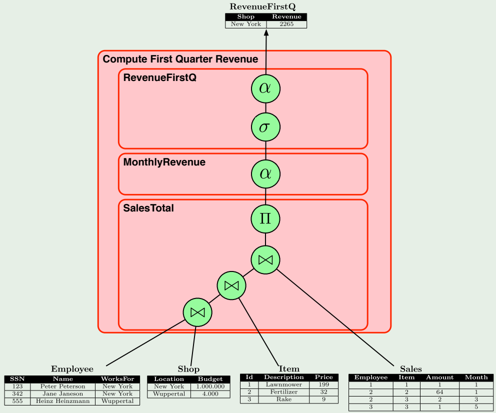
Example Transformation Granularity

Example



Example Transformation Granularity

Example



RevenueFirstQ

Shop	Revenue
New York	2265

Employee

SSN	Name	WorksFor
123	Peter Peterson	New York
342	Jane Janeson	New York
555	Heinz Heinzmann	Wuppertal

Shop

Location	Budget
New York	1.000.000
Wuppertal	4.000

Item

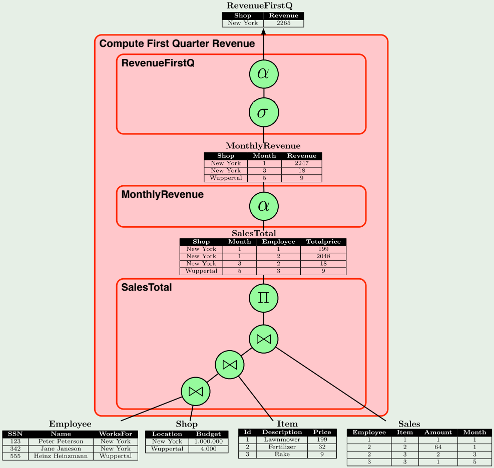
Id	Description	Price
1	Lawnmower	199
2	Fertilizer	32
3	Rake	9

Sales

Employee	Item	Amount	Month
1	1	1	1
2	2	64	1
2	3	2	3
3	3	1	5

Example Transformation Granularity

Example



RevenueFirstQ

Shop	Revenue
New York	2265

MonthlyRevenue

Shop	Month	Revenue
New York	1	2247
New York	3	18
Wuppertal	5	9

SalesTotal

Shop	Month	Employee	Totalprice
New York	1	1	199
New York	1	2	2048
New York	3	2	18
Wuppertal	5	3	9

Employee

SSN	Name	WorksFor
123	Peter Peterson	New York
342	Jane Janeson	New York
555	Heinz Heinzmann	Wuppertal

Shop

Location	Budget
New York	1.000.000
Wuppertal	4.000

Item

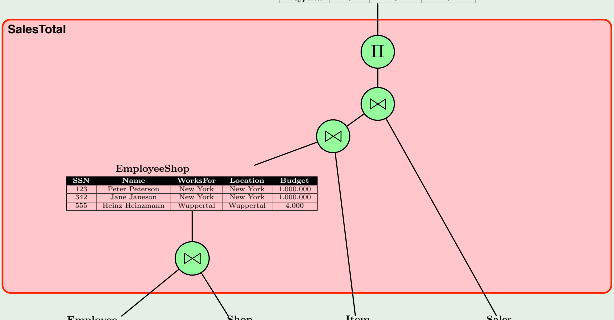
Id	Description	Price
1	Lawnmower	199
2	Fertilizer	32
3	Rake	9

Sales

Employee	Item	Amount	Month
1	1	1	1
2	2	64	1
2	3	2	3
3	3	1	5

Example Transformation Granularity

Example



SalesTotal

Shop	Month	Employee	Totalprice
New York	1	1	199
New York	1	2	2048
New York	3	2	18
Wuppertal	5	3	9

EmployeeShop

SSN	Name	WorksFor	Location	Budget
123	Peter Peterson	New York	New York	1.000.000
342	Jane Janeson	New York	New York	1.000.000
555	Heinz Heinzmann	Wuppertal	Wuppertal	4.000

Employee

SSN	Name	WorksFor
123	Peter Peterson	New York
342	Jane Janeson	New York
555	Heinz Heinzmann	Wuppertal

Shop

Location	Budget
New York	1.000.000
Wuppertal	4.000

Item

Id	Description	Price
1	Lawnmower	199
2	Fertilizer	32
3	Rake	9

Sales

Employee	Item	Amount	Month
1	1	1	1
2	2	64	1
2	3	2	3
3	3	1	5

Datamodel Mismatch

Rationale

- Provenance datamodel often $\neq$ data model of transformation
  - Lineage is list or relations
  - Provenance Polynomials are formulas over tuple variables
  - Causality is set of tuples (but from different relations)
  - ...

GY

Datamodel Mismatch

Rationale

- Provenance datamodel often $\neq$ data model of transformation
  - Lineage is list or relations
  - Provenance Polynomials are formulas over tuple variables
  - Causality is set of tuples (but from different relations)
  - ...
- $\Rightarrow$Map it to transformation data model
  - That what *Perm* does
  - That is what *DBNotes* does

GY

Datamodel Mismatch

Rationale

- Provenance datamodel often $\neq$ data model of transformation
  - Lineage is list or relations
  - Provenance Polynomials are formulas over tuple variables
  - Causality is set of tuples (but from different relations)
  - ...
- $\Rightarrow$Map it to transformation data model
  - That what *Perm* does
  - That is what *DBNotes* does
- $\Rightarrow$Live with different data models
  - Store provenance separately
    - $\Rightarrow$Querying becomes a problem
  - Use datamodel expressive enough to store both data and provenance

Example Map Provenance to Transformation Data Model

Example

- E.g., Perm Provenance Relational Representation
- Provenance is Set of List of Tuples
- Relational Representation: For each result tuple t and witness list w
  - Create one tuple that combines
  - t with all tuples from w

Example (Normal Query Result)

R

a	b
1	2
3	4

S

c	d
1	2
1	3

$\xrightarrow{q}$

Result q

a
1

Example Map Provenance to Transformation Data Model

Example

- E.g., Perm Provenance Relational Representation
- Provenance is Set of List of Tuples
- Relational Representation: For each result tuple t and witness list w
  - Create one tuple that combines
  - t with all tuples from w

Example (Perm Provenance Representation)

R

a	b
1	2
3	4

S

c	d
1	2
1	3

$\xrightarrow{q}$

Result + Provenance

a	P(a)	P(b)	P(c)	P(d)
1	1	2	1	2
1	1	2	1	3

Example Map Provenance to Transformation Data Model

Example

- E.g., Perm Provenance Relational Representation
- Provenance is Set of List of Tuples
- Relational Representation: For each result tuple t and witness list w
  - Create one tuple that combines
  - t with all tuples from w

Example (Perm Provenance Representation)

R

a	b
1	2
3	4

S

c	d
1	2
1	3

$\xrightarrow{q}$

Result + Provenance

a	P(a)	P(b)	P(c)	P(d)
1	1	2	1	2
1	1	2	1	3

Example Use more expressive data model

Example

- Causality is set of tuples from different relation (causes)
- ⇒e.g., XML can express such sets
- map data and provenance into XML

Address

	Id	City	Street
$\{a_1\}$	1	Toronto	52 Bloor Street

Person

	Name	Address
$\{p_1\}$	Peter	1
$\{p_2\}$	Heinz	1

GY

Example Use more expressive data model

Example

- Causality is set of tuples from different relation (causes)
- ⇒e.g., XML can express such sets
- map data and provenance into XML

```
<DB>
  <Person>
    <Tuple id="1"><Attr>Peter</Attr><Attr>1</Attr></Tuple>
    <Tuple id="2"><Attr>Heinz</Attr><Attr>1</Attr></Tuple>
  </Person>
  <Address>
    <Tuple id="1"><Attr>1</Attr><Attr>Toronto</Attr>
                  <Attr>52 Bloor Street</Attr></Tuple>
  </Address>
  <QueryResult query="q">
    <Tuple id="1"><Attr>Peter</Attr></Tuple>
    <Tuple id="2"><Attr>Heinz</Attr></Tuple>
  </QueryResult>
  <Provenance tuple="1" query="q">
    <TupleRef relation="Person" tuple="1"/>
    <TupleRef relation="Address" tuple="1"/>
  </Provenance>
</DB>
```

Outline

1. Provenance Storage
   - Introduction
   - Compression Methods for Provenance
   - Index Structures for Provenance
   - Recap

Overview

- Given that provenance is large
- How to safe storage space?

Approaches

1. Choose coarser granularity
   - $\Rightarrow$loose information
   - $\Rightarrow$sometime positive: Information overload

GY

Overview

- Given that provenance is large
- How to safe storage space?

Approaches

1. Choose coarser granularity
   - ⇒loose information
   - ⇒sometime positive: Information overload
2. Choose compact provenance model
   - ⇒may loose information
   - ⇒restrict to important information

GY

Overview

- Given that provenance is large
- How to safe storage space?

Approaches

1. Choose coarser granularity
   - ⇒loose information
   - ⇒sometime positive: Information overload
2. Choose compact provenance model
   - ⇒may loose information
   - ⇒restrict to important information
3. ⇒Compression!

GY

Compression for Provenance

Goals for Compressing Provenance

1. Reduce storage size
   - That is why we are doing that at all
2. Efficient compression/decompression
   - E.g., Compression useless if it takes days to compress
3. Lossless
   - or only loose unimportant information
4. Show Compressed representation to user
   - Compressed provenance as a summary
   - Reduce information overload
5. Execute queries over compressed representation (partially?)
   - Save cost of compression/decompression
   - Best possible outcome: query performance increase

Choose Compression Techniques

1. Generic Compression Techniques
   - E.g., Dictionary compression (LZ77)

Choose Compression Techniques

1. Generic Compression Techniques
   - E.g., Dictionary compression (LZ77)
2. Exploiting structure of provenance
   - Repeating part in the provenance
     - $\Rightarrow$Specialized dictionary compression
   - Structure in provenance imposed by transformation
     - E.g., provenance of $R \bowtie S$ always has tuples form R combined with tuples from S

ILLINOIS INSTITUTE OF TECHNOLOGY

Generic Compression

Properties

- Good compression rates
- No need to adapt to provenance
- Output incomprehensible for human
- Compression/decompression quite expensive
- Queries over compressed provenance unlikely
    - Unclear how to extract parts of provenance

Example

try:

```
echo "(r1 x s1) + t1" | gzip -cf
```

result:

```
?m|P?(2T?P(6?T?V(1???o?
```

Factoring out Common Parts from Provenance

Overlap in provenance

- Recall alternatives
- Reuse of subquery results

Example

$$q = \pi_a(R \bowtie_{a=d} S) \times U$$

R

	a	b	c
r_1	1	1	1
r_2	2	3	2
r_3	5	1	1

S

	d
s_1	1
s_2	2

U

	e
u_1	Hello
u_2	Test

Q

	a	e
t_1	1	Hello
t_2	1	Test
t_3	2	Hello
t_4	2	Test

ILLINOIS INSTITUTE OF TECHNOLOGY

Factoring out Common Parts from Provenance

Overlap in provenance

- Recall alternatives
- Reuse of subquery results

Example

$$q = \pi_a(R \bowtie_{a=d} S) \times U$$

Provenance Polynomials

$$\mathbb{N}[I](q, t_1) = (r_1 \times s_1) \times u_1$$

$$\mathbb{N}[I](q, t_2) = (r_1 \times s_1) \times u_2$$

R

	a	b	c
r_1	1	1	1
r_2	2	3	2
r_3	5	1	1

S

	d
s_1	1
s_2	2

U

	e
u_1	Hello
u_2	Test

Q

	a	e
t_1	1	Hello
t_2	1	Test
t_3	2	Hello
t_4	2	Test

GY

Factoring out Common Parts from Provenance

Overlap in provenance

- Recall alternatives
- Reuse of subquery results

Example

$$q = \pi_a(R \bowtie_{a=d} S) \times U$$

Why-provenance

$$Why(q, t_1) = \{\{r_1, s_1, u_1\}\}$$

$$Why(q, t_2) = \{\{r_1, s_2, u_2\}\}$$

R

	a	b	c
r_1	1	1	1
r_2	2	3	2
r_3	5	1	1

S

	d
s_1	1
s_2	2

U

	e
u_1	Hello
u_2	Test

Q

	a	e
t_1	1	Hello
t_2	1	Test
t_3	2	Hello
t_4	2	Test

Factoring out Common Parts from Provenance

Overlap in provenance

- Recall alternatives
- Reuse of subquery results

Example

$$q = \pi_a(R \bowtie_{a=d} S) \times U$$

Perm Influence

$$\mathcal{PI}(q, t_1) = \{< {\color{red}r_1, s_1}, u_1 >\}$$

$$\mathcal{PI}(q, t_2) = \{< {\color{red}r_1, s_1}, u_2 >\}$$

R

	a	b	c
r_1	1	1	1
r_2	2	3	2
r_3	5	1	1

S

	d
s_1	1
s_2	2

U

	e
u_1	Hello
u_2	Test

Q

	a	e
t_1	1	Hello
t_2	1	Test
t_3	2	Hello
t_4	2	Test

GY

How to Detect Common Elements?

Brute force

- Use binary or text representation of provenance
- Find common substrings (bit-patterns)
- Simple approach quadratic in size of provenance (of two results)

Improvements

- Take equivalences on provenance into account
  - E.g., Why-provenance witnesses are sets $\Rightarrow \{a, b\} = \{b, a\}$
  - E.g., Equivalences on polynomials $r_1 \times (s_1 + s_2) = (r_1 \times s_1) + (r_1 \times s_2)$
- Use knowledge about query to improve matching performance
  - E.g., $(R \bowtie S) \times U$: the provenance of each result tuple from $R \bowtie S$ will be repeated for each tuple in U

Factorization of Provenance

Properties

- Compression rate depends on query
- No need to adapt to provenance
  - compress provenance as text
  - without adaptation we may loose opportunities
- human readable: e.g., graph representation
- Queries over provenance: only small changes
- General approach of structural matching to detect to expensive
- Unclear how to integrate with provenance computation on demand

GY

Factoring out Structural Provenance Parts

Rationale

- Analyze query to predetermine structure of provenance
- Factor out the structural parts from the representation

Example

- $q = \pi_a(R \bowtie_{b=d} S)$
- Provenance of a result tuple (Set semantics is sum of multiplications of tuple from R and tuple from S)
- $\Rightarrow$No need to store addition and multiplication operations if we know query

$$\mathbb{N}[I](q,t) = r_1 \times s_1 + r_1 \times s_2 + r_3 \times s_5 \Longrightarrow r_1 s_1 r_1 s_2 r_3 s_5$$

Factoring out Structural Provenance Parts

How to store?

- Compressed representation only interpretable with query!
- $\Rightarrow$Store query with compressed representation
  - Easy: $\pi_a(R \bowtie_{b=d} S), r_1 s_1 r_1 s_2 r_3 s_5)$
- Alternatively store instructions for decompression
  - May be more compact
  - E.g., pattern $(\sum e_i \times e_{i+1})$

Factoring out Structural Provenance Parts

Properties

- Compression rate depends on query, usually low constant factor
- Need query to interpret it
- Need representation for alternative patterns
- Not really human readable
- Queries over provenance: some adaptations
- After static analysis of query its simple
- More or less clear how to integrate with on-demand provenance tracking

GY

Symbolic and Declarative Representations of Provenance

- So far: explicitly listing tuples (input data) in provenance
- The amount of input data in provenance may be huge
- $\Rightarrow$Find more compact representations

Example

- The provenance of each result tuple t
- contains all tuples from S with $b < a$
- Assume $|S| = 1,000,000$

```
SELECT *
FROM R
WHERE EXISTS (SELECT * FROM S WHERE S.b < R.a)
```

Symbolic and Declarative Representations of Provenance

Rationale

- Queries are concise representations of large number of tuples
- $\Rightarrow$Replace part of the provenance with an expression to compute it

Example

- $\mathcal{PI}(q,t) = \{< r_1, s_2 >, < r_1, s_5 >, \dots\}$
- contains all tuples from S with $b < a$
- $\Rightarrow \mathcal{PI}(q,t) = \{< r_1, \sigma_{R.a<b}(S) >\}$

```
SELECT *
FROM R
WHERE EXISTS (SELECT * FROM S WHERE S.b < R.a)
```

Symbolic and Declarative Representations of Provenance

Use case: Queries over Provenance

- Assume user runs query over Perm provenance to retrieve provenance of specific tuple t
- Provenance is stored as queries
- ⇒We can delay the generation of actual provenance to when its needed
- ⇒Improve performance of queries

Challenges

- How to decide when to use queries vs. actual data?
- Dynamically interpreting and executing query require significant changes to system

Symbolic and Declarative Representations of Provenance

Properties

- Compression Rates can be significant
- Human readable: depends on user background
- Advantage for query processing over provenance
- Integrate with query engines is hard
- How to determine when to use symbolic representation?

Provenance Model Specific Methods

Factorization of Provenance Polynomials

- Factorization of Polynomials $r_1 \times r_2 + r_1 \times r_3 = r_1(r_2 + r_3)$
- Equivalent polynomials can sometimes differ exponentially in size!
- Find different factorization to save space

ILLINOIS INSTITUTE OF TECHNOLOGY

Provenance Model Specific Methods

Factorization of Provenance Polynomials

- Factorization of Polynomials $r_1 \times r_2 + r_1 \times r_3 = r_1(r_2 + r_3)$
- Equivalent polynomials can sometimes differ exponentially in size!
- Find different factorization to save space

Example

- E.g., $\pi_1(R_1 \times R_2 \times ... \times R_n)$
- provenance is $(r_{11} \times r_{21} \times .. \times r_{n1}) + (r_{12} \times r_{21} \times .. \times r_{n1}) + \ldots$
- Size of provenance polynomial: $|R_1| \times |R_2| \times \ldots \times |R_n|$

GY

Provenance Model Specific Methods

Factorization of Provenance Polynomials

- Factorization of Polynomials $r_1 \times r_2 + r_1 \times r_3 = r_1(r_2 + r_3)$
- Equivalent polynomials can sometimes differ exponentially in size!
- Find different factorization to save space

Example

- E.g., $\pi_1(R_1 \times R_2 \times ... \times R_n)$
- provenance is $(r_{11} \times r_{21} \times .. \times r_{n1}) + (r_{12} \times r_{21} \times .. \times r_{n1}) + \dots$
- Size of provenance polynomial: $|R_1| \times |R_2| \times \dots \times |R_n|$
- Equivalent polynomials: $(r_{11} + r_{12} + \dots r_{1m_1}) \times (r_{21} + r_{22} + \dots) \times \dots$
- Size: $|R_1| + \dots + |R_n|$

Outline

1 Provenance Storage
- Introduction
- Compression Methods for Provenance
- Index Structures for Provenance
- Recap

Overview

Why Index Structures for Provenance?

- Improve performance of specific access patterns
- Specialized index structures needed if access patterns differ from regular data
  - E.g., path queries on provenance

Overview

Why Index Structures for Provenance?

- Improve performance of specific access patterns
- Specialized index structures needed if access patterns differ from regular data
  - E.g., path queries on provenance

Access Patterns

- Forward queries:
  - Give me all output data items that are derived from input data item x
- Backward queries:
  - Give me all input data items that are in the provenance of output data item x
- For sets?

Overview

Why Index Structures for Provenance?

- Improve performance of specific access patterns
- Specialized index structures needed if access patterns differ from regular data
  - E.g., path queries on provenance

Note

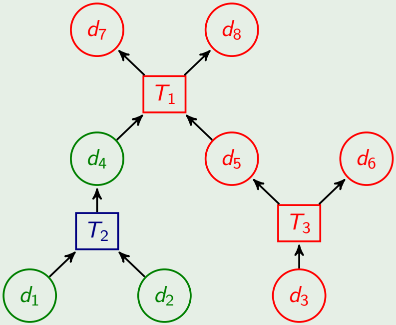
- Use generic data processing model:
  - Atomic unit of input and output data: data item
  - Transformations: DAG (directed acyclic graph)

ILLINOIS INSTITUTE OF TECHNOLOGY

Forward queries

- Given an input data item d
- Determine all outputs influenced by d
- $\Rightarrow d$ is in provenance
- Single Data Item vs. Sets
- Result:
  - Data Items (Intermediate?)
  - Transformations

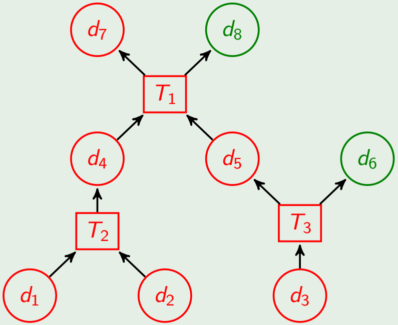
Example (Forward Query d_3)



Backward Query

- Given an output data item d
- Determine all inputs in provenance of d
- Single Data Item vs. Sets
- Result:
  - Data Items (Intermediate?)
  - Transformations

Example (Backward Query d_7)



Interval Encoding of Provenance

Provenance Model

- Tree (directed acyclic graph (DAG))
  - Nodes are data items and transformations
- Stored in relational DB

Queries

- **Forward** find all ancestors of node
- **Backward** find all decentness of node

ILLINOIS INSTITUTE OF TECHNOLOGY

Interval Encoding of Provenance

How to store data?

GY

Interval Encoding of Provenance

How to store data?

1. Edge relation $(node_1, node_2)$
   - Storage cost is small
   - Queries are recursive!

Interval Encoding of Provenance

How to store data?

1. Edge relation $(node_1, node_2)$
   - Storage cost is small
   - Queries are recursive!
2. Transitive closure of edge relation
   - Storage costs are high!
   - Queries are simple

GY

Interval Encoding of Provenance

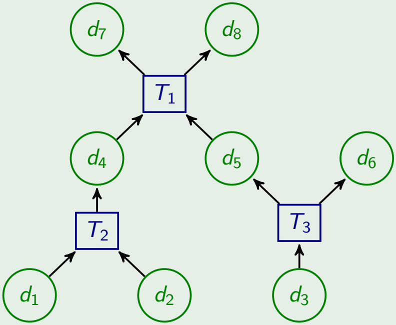
How to store data?

1. Edge relation $(node_1, node_2)$
   - Storage cost is small
   - Queries are recursive!
2. Transitive closure of edge relation
   - Storage costs are high!
   - Queries are simple
3. Path relation
   - Give each path an identifier
   - Store triples
     - `(path_id, position, node)`
   - ⇒Storage costs are high, queries cheaper

Example Provenance DAG Storage

Example (Edge Relation)

from	to
d_1	T_2
d_2	T_2
d_3	T_3
T_2	d_4
T_3	d_5
T_3	d_6
d_4	T_1
d_5	T_1
T_1	d_7
T_1	d_8



Example Provenance DAG Storage

Example (Edge Relation)

Backward query for d_7

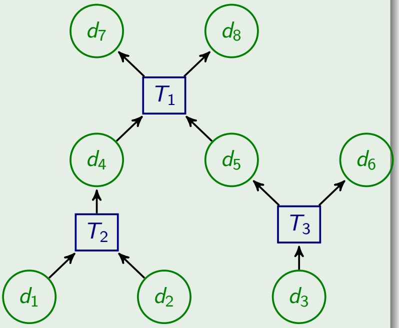
```
WITH RECURSIVE reachable(from,to)
AS (
     SELECT * FROM edges
   UNION ALL
     SELECT l.from, r.to
     FROM reachable l,
          reachable r
     WHERE l.to = r.from
)
SELECT from
FROM reachable
WHERE to = d7;
```

Example Provenance DAG Storage

Example (Transitive Edge Relation)

from	to
d_1	T_2
d_2	T_2
d_3	T_3
T_2	d_4
T_3	d_5
T_3	d_6
d_4	T_1
d_5	T_1
T_1	d_7
T_1	d_8

...	...
d_1	d_4
d_1	T_1
d_1	d_7
d_1	d_8
d_2	d_4
...	...



Example Provenance DAG Storage

Example (Transitive Edge Relation)

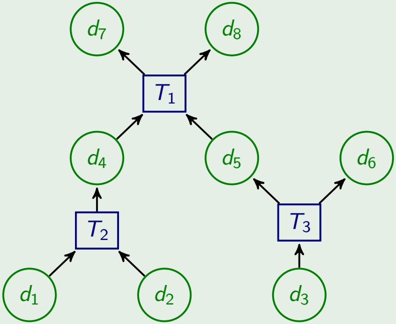
Backward query for d_7

```
SELECT from
FROM edge
WHERE to = d7
```

Example Provenance DAG Storage

Example (Path Relation)

path_id	order	node
1	1	d_1
1	2	T_2
1	3	d_4
1	4	T_1
1	5	d_7
2	1	d_2
2	2	T_2
...	...	...



Example Provenance DAG Storage

Example (Path Relation)

Backward query for d_7

```
SELECT from
FROM path l, path r
WHERE l.path_id = r.path_id
      AND r.node = d7
      AND l.position < r.position
```

Discussion of Storage Alternatives

- Both presented alternatives are unsatisfactory
- Number of nodes in graph: n
- Edge Relation
  - Storage Size: n
  - Recursive querying
- Transitive Closure
  - Storage size: n^2
  - No recursive querying
  - Query table of size n^2
- Paths
  - Storage size: $O(f^h)$ with $h =$ height and f is fan-out
  - No recursive querying
  - Query table of exponential size

Interval Encoding of Trees

Rationale

- Represent nodes as numeric intervals (l, r)
- Interval inclusion determines parent child relationship
  - If c is child of p then
    - $p.l \leq c.l$ and $c.r \leq p.r$
  - $\Rightarrow$Reverse condition for reverse check
  - $\Rightarrow$Same condition for descendent instead of child

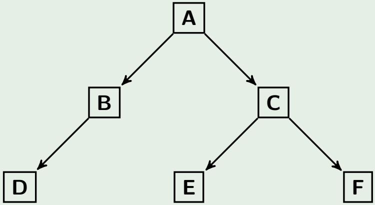
ILLINOIS INSTITUTE OF TECHNOLOGY

Interval Encoding of Trees

Rationale

- Represent nodes as numeric intervals (l, r)
- Interval inclusion determines parent child relationship
  - If c is child of p then
    - $p.l \leq c.l$ and $c.r \leq p.r$
  - $\Rightarrow$Reverse condition for reverse check
  - $\Rightarrow$Same condition for descendent instead of child

Example

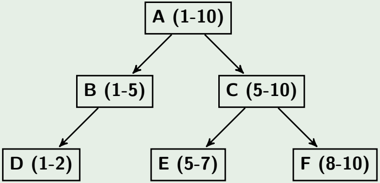


Interval Encoding of Trees

Rationale

- Represent nodes as numeric intervals (l, r)
- Interval inclusion determines parent child relationship
  - If c is child of p then
    - $p.l \leq c.l$ and $c.r \leq p.r$
  - $\Rightarrow$Reverse condition for reverse check
  - $\Rightarrow$Same condition for descendent instead of child

Example



Interval Encoding of Trees

Example

node	l	r
A	1	10
B	1	5
C	5	10
D	1	2
E	5	7
F	8	10

ILLINOIS INSTITUTE OF TECHNOLOGY

Interval Encoding of Trees

Example

Backward query for d_7

```
SELECT from
FROM nodes a, nodes d
WHERE a.l < d.l AND d.r < a.r
      a.node = d7
```

Interval Encoding of Trees

Example

Direct parent query for d_7

```
SELECT from
FROM nodes a, nodes d
WHERE a.l < d.l AND d.r < a.r
      a.node = d7
      AND NOT EXISTS (
            SELECT *
            FROM nodes m
            WHERE a.l < m.l AND m.r < a.r
                  AND m.l < d.l AND d.r < m.r
      )
```

Properties of Interval Encoding for Trees

- Storage size: n
- Ancestor/Decedent Queries: $O(n^2)$ (no index)
- Parent/Child Queries: $O(n^3)$ (no index)

Properties of Interval Encoding for Trees

Comparison

- Storage
  - $O(n)$ (edge, interval), $O(n^2)$ (edge transitive), $O(f^h)$ (paths)

Properties of Interval Encoding for Trees

Comparison

- Storage
  - $O(n)$ (edge, interval), $O(n^2)$ (edge transitive), $O(f^h)$ (paths)
- Ancestor/Decedent Queries
  - $O(n * h)$ (edge), $O(n^2)$ (interval, edge transitive), $O(f^{2h})$ (paths)
  - edge recursive!
  - "interval" has inequality condition while "edge transitive" has equality

Properties of Interval Encoding for Trees

Comparison

- Storage
  - $O(n)$ (edge, interval), $O(n^2)$ (edge transitive), $O(f^h)$ (paths)
- Ancestor/Decedent Queries
  - $O(n * h)$ (edge), $O(n^2)$ (interval, edge transitive), $O(f^{2h})$ (paths)
  - edge recursive!
  - "interval" has inequality condition while "edge transitive" has equality
- Parent/Child Queries
  - $O(n)$ (edge), $O(f^h)$ (paths), $O(n^2)$ (edge transitive), $O(n^3)$ (interval)

GY

Interval Encoding for Provenance DAGs

Rationale

- Interval encoding is advantageous
- Only works for trees
  - One-dimensional intervals not enough!
- ... but provenance is DAG
- $\Rightarrow$need extension that deals with DAGs

Interval Encoding for Provenance DAGs

Rationale

- Interval encoding is advantageous
- Only works for trees
  - One-dimensional intervals not enough!
- . . . but provenance is DAG
- $\Rightarrow$need extension that deals with DAGs

N-dimensional Encoding

- Number of dimensions determined by graph structure
- Sometimes even DAGs can be encoded using just one dimension

GY

Turning Graphs into Trees

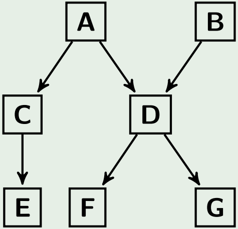
Idea

- Replicate nodes with more than one parent
- A node with n parents
  - Create n nodes with one parent
- $\Rightarrow$Parent/child relationships are the same
- $\Rightarrow$Can use interval encoding

ILLINOIS INSTITUTE OF TECHNOLOGY

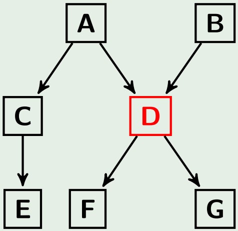
Example Transform To Tree

Example



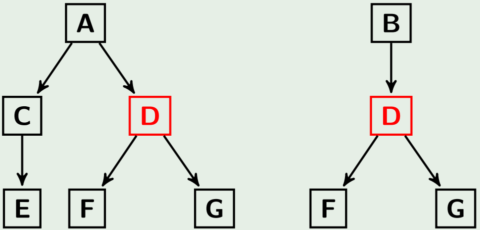
Example Transform To Tree

Example



Example Transform To Tree

Example



Discussion Transformation to Tree

- Result can be interval encoded
- Size may be much larger
- Not all transformations necessary (overlapping)
- $\Rightarrow$Need method to test whether subgraph is interval encodable
- $\Rightarrow$Avoid "tree-ifying" subgraphs under replicated nodes

ILLINOIS INSTITUTE OF TECHNOLOGY

When is a DAG Interval Encodable?

Incomparability Graph I_G

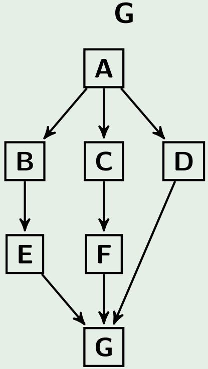
- Model which nodes in a graph are incomparable
  - $\Rightarrow$have no parent/child relationship
- Can be used to determine whether DAG is interval encodable
- Same node as original graph
- Edge between two node $\Rightarrow$nodes are incomparable

Transitive orientable

- I_G is transitively orientable if edges can be directed ...
  - Edges (u, v) and (v, w)
  - $\Rightarrow$there exists edge (u, w)
- If I_G is transitively orientable $\Rightarrow$interval-encodable

Example I_G

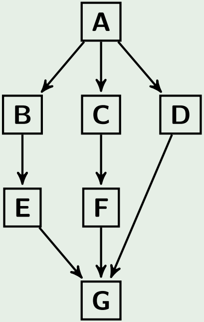
Example



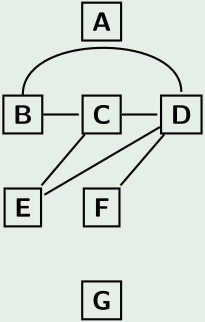
Example I_G

Example

G

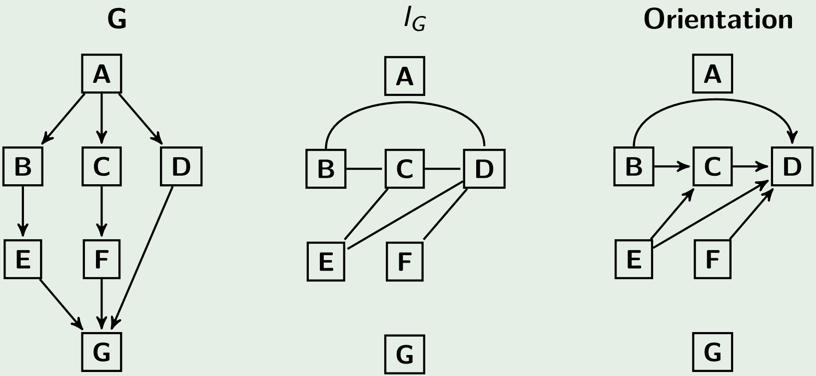


I_G



Example I_G

Example



Derive an Interval Encoding from I_G

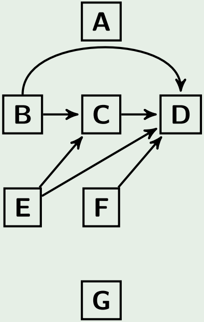
Approach

- Create two total orders L and L'
  - Order L is created by traversing the nodes
    - From sources to sinks
    - Never visit a node before visiting its parents
    - If multiple nodes are ok $\Rightarrow$traverse in order of orientation
  - Order L' is created by using reversed orientation
- Assign each node n an interval based on its position in the orders
  - Let $P_L(n)$ be the position in the order L
  - Assign node n the interval $[-P_L(n), P'_L(n)]$

OF TECHNOLOGY

Example Derive an Interval Encoding

Example



Order L

7		6		5		4		3		2		1
A	>	B	>	E	>	C	>	F	>	D	>	G

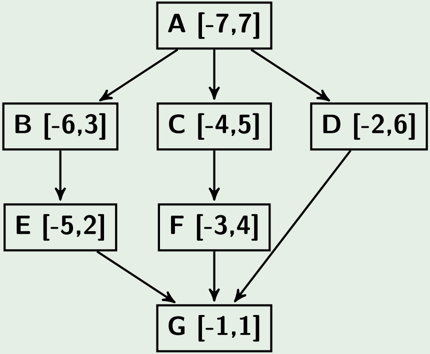
Order L'

7		6		5		4		3		2		1
A	>	D	>	C	>	F	>	B	>	E	>	G

OF TECHNOLOGY

Example Derive an Interval Encoding

Example



Complete Approach for Interval Encoding

1. Detect maximal interval-encodable subgraphs (using I_G's)
2. Replace these subgraphs with new nodes
3. Transform into tree
4. Substitute new nodes with original sub-graphs
5. Recompute I_G and create interval-encoding

ILLINOIS INSTITUTE OF TECHNOLOGY

Conclusions Interval Encoding

Advantages

- Index structure for provenance DAGs
- Supports efficient backward and forward queries
- Lower storage costs than alternatives

Broader Perspective

- One provenance-specific access pattern
- ⇒Improves performance and storage for this pattern
- E.g., provenance for relational queries
    - Non-typical access pattern for normal data access

Outlook Additional Indexing Approaches for Provenance

Information retrieval index structures for provenance

- Important information retrieval problem
  - Answer query that returns all documents that contain a keyword
  - ⇒Efficient access of binary relation (document, object)
  - Optimized index structures for this kind of access
- ⇒The transitive closure of graphs is a binary relation
- ⇒Can use these index structures to store provenance graphs
- ⇒Efficient backward and (possibly forward) lookup
- Most indices are not meant to be incrementally maintained!

OF TECHNOLOGY

Open Problems for Indexing Provenance

- Integrate indices with automatic provenance generation
  - Construct index during provenance generation
  - Use index during provenance generation
- Incremental maintenance of indices
  - Most provenance index type are not incrementally maintainable
- Indices for both data and its provenance
- Combine indexing with compression

Outline

1 Provenance Storage
- Introduction
- Compression Methods for Provenance
- Index Structures for Provenance
- Recap

Recap

Provenance Storage Challenges

- Size
  - Can exceed input + output size
  - Even for compact representations
- Data model mismatch
  - Different model than transformation input and output data
    1. use more expressive data model
    2. use different data models for provenance and data
    3. map provenance data model to transformation data model
- Queries
  - Need efficient access to data
  - Access patterns provenance not necessarily = access pattern data

Recap

Provenance Compression

- Goals:
  - Reduce Size ;-)
  - Lossless?
  - Use as summary
  - Efficient compression/decompression (queries)
  - Integrate with generation and querying
- Approaches:
  - Generic Compression
    - Good compression
    - Expensive
    - Hard to evaluate queries over
    - Not human readable

GY

Recap

Provenance Compression

- Goals:
  - Reduce Size ;-)
  - Lossless?
  - Use as summary
  - Efficient compression/decompression (queries)
  - Integrate with generation and querying
- Approaches:
  - Store common parts only once
    - Compression depends on query
    - Relatively cheap
    - Good for user presentation or querying
    - Unclear how to effectively apply during provenance computation

GY

Recap

Provenance Compression

- Goals:
  - Reduce Size ;-)
  - Lossless?
  - Use as summary
  - Efficient compression/decompression (queries)
  - Integrate with generation and querying
- Approaches:
  - Use knowledge about transformation to reduce storage costs
    - Compression reasonable
    - Cost mostly payed upfront
    - Needs transformation to interpret it
    - Not human readable
    - Good for querying

Recap

Provenance Compression

- Goals:
  - Reduce Size ;-)
  - Lossless?
  - Use as summary
  - Efficient compression/decompression (queries)
  - Integrate with generation and querying
- Approaches:
  - Store expressions instead of actual provenance
    - Good compression if provenance follows pattern
    - Unclear how to determine expressions that represent provenance
    - Query processing needs run-time interpretation and execution of expressions to reconstruct provenance
    - Expressions can be good summaries if user understands expression language well

Recap

Provenance Indexing

- What kind of access to support
  - Path-queries
  - Forward queries (which data items are derived from x)
  - Backward queries (which data items are in provenance of x)
  - For single data item x or sets?
- Approaches
  - Interval encoding of paths
    - For backward and forward queries
    - Less storage than storing all paths
    - Better performance than recursive queries
  - Adapted Information Retrieval Index Structure
    - Efficient forward and backward lookup for single items

Literature I

P. Buneman, A. Chapman, J. Cheney, and S. Vansummeren.
A Provenance Model for Manually Curated Data.
In IPAW '06: International Provenance and Annotation Workshop, 162–170, 2006.

A. Chapman, H. V. Jagadish, and P. Ramanan.
Efficient Provenance Storage.
In SIGMOD '08: Proceedings of the 35th SIGMOD International Conference on Management of Data, 993–1006, 2008.

T. Heinis and G. Alonso.
Efficient Lineage Tracking for Scientific Workflows.
In SIGMOD '08: Proceedings of the 34th SIGMOD International Conference on Management of Data, 1007–1018, 2008.

A. Chapman and H. V. Jagadish.
Provenance and the Price of Identity.
In IPAW '08: International Provenance and Annotation Workshop, 106–119, 2008.

A. Kementsietsidis and M. Wang.
On the Efficiency of Provenance Queries.
In Proceedings of the 2009 IEEE International Conference on Data Engineering, 1223–1226, IEEE Computer Society Washington, DC, USA, 2009.

ILLINOIS INSTITUTE OF TECHNOLOGY

Literature II

A. Kementsietsidis and M. Wang.
Provenance Query Evaluation: What's so Special about it?.
In CIKM '09: Proceeding of the 18th Conference on Information and Knowledge Management, 681–690, 2009.

D. Olteanu and J. Zvodn.
On factorisation of provenance polynomials.
In TaPP '11: 3rd USENIX Workshop on the Theory and Practice of Provenance, 2011.

Y. Amsterdamer, D. Deutch, T. Milo, and V. Tannen.
On provenance minimization.
In Proceedings of the 30th Symposium on Principles of Database Systems (PODS), 141–152, 2011.

ILLINOIS INSTITUTE OF TECHNOLOGY