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

I. Database Provenance

I.1 Provenance Models and Systems

Boris Glavic

October 3, 2012

Outline

Overview - Provenance for Nested Subqueries

Rationale

- Provenance especially useful in domains with complex queries
  - E.g., datawarehousing
- Nested subqueries prevalent feature for such domains
  - `SELECT * FROM employee WHERE EXISTS (SELECT ...);`
- ⇒Need provenance support for that

ILLINOIS INSTITUTE OF TECHNOLOGY

Overview - Provenance for Nested Subqueries

Rationale

- Provenance especially useful in domains with complex queries
  - E.g., datawarehousing
- Nested subqueries prevalent feature for such domains
  - `SELECT * FROM employee WHERE EXISTS (SELECT ...);`
- ⇒Need provenance support for that

Approach

1. Extend relational algebra with nested subquery expressions
2. Apply Perm declarative definition to determine provenance
3. Create rewrite rules

Excursion: Nested Subqueries in SQL

Idea

- Quantification (boolean result):
  - Property holds for all results of query
  - Query produces results

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion: Nested Subqueries in SQL

Idea

- Quantification (boolean result):
  - Property holds for all results of query
  - Query produces results
- Scalar results (any datatype):
  - Query produces single result tuple ⇒use like constant value

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion: Nested Subqueries in SQL

Idea

- Quantification (boolean result):
  - Property holds for all results of query
  - Query produces results
- Scalar results (any datatype):
  - Query produces single result tuple ⇒use like constant value
- Correlation:
  - Query results depend on results of other query

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion: Where can I use Nested Subqueries

- Nested subquery expression evaluates to constant ⇒
- `WHERE`: filter out tuples
  - e.g., `SELECT * FROM R WHERE EXISTS (SELECT * FROM S)`
  - ⇒return tuples from r if S not empty
- `SELECT`: acts like a constant
  - e.g., `SELECT a, (SELECT count(*) FROM S) AS sc FROM R`
- `GROUP-BY`, `ORDER-BY`, `HAVING`:
  - Practical use questionable, but allowed

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion: Types of Nested Subqueries

- Existential Quantification: `EXISTS` (q)
- Scalar: q
- Operator:
  - Universal Quantification: `a op ALL` (q)
  - Existential Quantification: `a op ANY` (q)

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion: Existential Quantified Subqueries

Syntax

- `EXISTS` (q)
- q can be any query

Semantics

- Evaluates to ...
  - true: q has non-empty result
  - false: q has empty result

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion: Existential Quantified Subqueries

Example

```
SELECT *
FROM Employee e
WHERE EXISTS (SELECT *
              FROM Employee e2
              WHERE e2.Salary < 30)
```

Employee

	Id	Name	Salary	Dep
e_1	1	Peter	100	CS
e_2	2	Gertrud	67	CS
e_3	3	Michael	22	HR

Excursion: Existential Quantified Subqueries

Example

```
SELECT *
FROM Employee e
WHERE EXISTS (SELECT *
              FROM Employee e2
              WHERE e2.Salary < 30)
```

Employee

	Id	Name	Salary	Dep
e_1	1	Peter	100	CS
e_2	2	Gertrud	67	CS
e_3	3	Michael	22	HR

Result

	Id	Name	Salary	Dep
t_1	1	Peter	100	CS
t_2	2	Gertrud	67	CS
t_3	3	Michael	22	HR

Excursion: Scalar Subqueries

Syntax

- q
- q has to return
  - single tuple
  - with single attribute

Semantics

- Evaluates to . . .
  - The value of the single attribute in the single result tuple of q
  - Query fails during run-time if subquery returns more than one tuple

OF TECHNOLOGY

Excursion: Scalar Subqueries

Example

```
SELECT *
FROM Employee E
WHERE salary < (SELECT avg(salary)
                FROM Employee)
```

Employee

	Id	Name	Salary	Dep
e_1	1	Peter	100	CS
e_2	2	Gertrud	67	CS
e_3	3	Michael	22	HR

Excursion: Scalar Subqueries

Example

```
SELECT *
FROM Employee E
WHERE salary < (SELECT avg(salary)
             FROM Employee)
```

Employee

	Id	Name	Salary	Dep
e_1	1	Peter	100	CS
e_2	2	Gertrud	67	CS
e_3	3	Michael	22	HR

Result

	Id	Name	Salary	Dep
t_1	3	Michael	22	HR

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion: Comparison Subqueries - Universal Quantification

Syntax

- *e op* `ALL (q))`
- *e* is expression
  - e.g., access attribute or constant
- *q* has to return
  - a single attribute
- *op* is comparison operator: compatible with *Q* and *e*
  - NOT OK: `"Hello"= ALL (SELECT count(*) ...`

Semantics

- Evaluates to ...
  - true: if *e op t* evaluates to true for every tuple $t \in Q$
  - false: otherwise

Excursion: Comparison Subqueries - Universal Quantification

Example

```
SELECT *
FROM Employee E
WHERE e.Salary >= ALL (SELECT salary
                       FROM Employee)
```

Employee

	Id	Name	Salary	Dep
e_1	1	Peter	100	CS
e_2	2	Gertrud	67	CS
e_3	3	Michael	22	HR

OF TECHNOLOGY

Excursion: Comparison Subqueries - Universal Quantification

Example

```
SELECT *
FROM Employee E
WHERE e.Salary >= ALL (SELECT salary
                       FROM Employee)
```

Employee

	Id	Name	Salary	Dep
e_1	1	Peter	100	CS
e_2	2	Gertrud	67	CS
e_3	3	Michael	22	HR

Result

	Id	Name	Salary	Dep
t_1	1	Peter	100	CS

OF TECHNOLOGY

Excursion: Comparison Subqueries - Existential Quantification

Syntax

- *e op* `ANY` (q)
- Syntactic sugar for $op =$ is: *e* `IN` (q)
- *e* is expression
  - e.g., access attribute or constant
- *q* has to return
  - a single attribute
- *op* is comparison operator: compatible with Q and e

Semantics

- Evaluates to ...
  - true: if e op t evaluates to true for at least one tuple $t \in Q$
  - false: otherwise

Excursion: Comparison Subqueries - Existential Quantification

Example

```
SELECT *
FROM Employee E
WHERE e.Salary > ANY (SELECT salary
                      FROM Employee)
```

Employee

	Id	Name	Salary	Dep
e_1	1	Peter	100	CS
e_2	2	Gertrud	67	CS
e_3	3	Michael	22	HR

OF TECHNOLOGY

Excursion: Comparison Subqueries - Existential Quantification

Example

```
SELECT *
FROM Employee E
WHERE e.Salary > ANY (SELECT salary
                      FROM Employee)
```

Employee

	Id	Name	Salary	Dep
e_1	1	Peter	100	CS
e_2	2	Gertrud	67	CS
e_3	3	Michael	22	HR

Result

	Id	Name	Salary	Dep
t_1	1	Peter	100	CS
t_2	2	Gertrud	67	CS

OF TECHNOLOGY

Execution of Nested Subqueries

Approach

- Existential and Scalar subqueries:
  - Execute subquery once and store result
  - Handle as constant in execution of outer query
- Comparison subqueries
  - Execute subquery once and cache result table
  - For each result of outer query
    - Scan through cached result table once and check the comparison
    - For `ANY`-subqueries return true if evaluates to true
    - For `ALL`-subqueries return false if evaluates to false
    - If done return true (`ALL`) or false (`ANY`)

OF TECHNOLOGY

Excursion - Correlated Subqueries

Idea

- Subquery references attributes from outer query
- Semantics: Evaluate subquery for every tuple
  - in the result of `FROM` clause of outer query

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion - Correlated Subqueries

Example

```
SELECT *
FROM Employee E
WHERE e.Salary > ANY (SELECT salary
                      FROM Employee E2
                      WHERE E.Dep = E2.Dep)
```

Employee

	Id	Name	Salary	Dep
e_1	1	Peter	100	CS
e_2	2	Gertrud	67	CS
e_3	3	Michael	22	HR

Excursion - Correlated Subqueries

Example

```
SELECT *
FROM Employee E
WHERE e.Salary > ANY (SELECT salary
                      FROM Employee E2
                      WHERE E.Dep = E2.Dep)
```

Employee

	Id	Name	Salary	Dep
e_1	1	Peter	100	CS
e_2	2	Gertrud	67	CS
e_3	3	Michael	22	HR

Result

	Id	Name	Salary	Dep
t_1	1	Peter	100	CS

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion - Correlated Subqueries Evaluation

Nested Iteration

- For each result tuple of outer query
  - Substitute values from outer queries
  - Execute nested query
  - Evaluate nested subquery expression (e.g., $a =$ `ALL (SELECT ...)`)
- Naive implementation: Need to optimize subquery for each outer tuple
- Cache plan: Optimize query once and reuse plan with slight modifications
- Still expensive: Assume outer query returns 1'000'000 tuples!

Excursion - Un-nesting and De-correlation

Idea

- Correlations are hindering efficient query evaluation
- Exploit query equivalences
- Sometimes nested subquery can be expressed as join
- Try to rewrite nested subqueries during logical optimization
- $\Rightarrow$Standard query that is easier to optimize

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion - Example EXISTS

EXISTS with = correlation

- Subquery is `EXISTS` (q)
- q is SPJ query
- Correlation is $e_o = e_i$ between
  - outer query correlated attribute e_o
  - inner query attribute or constant e_i
- Turn into Join
  - `FROM` ..., (`SELECT DISTINCT` e_i `FROM` q) `AS` q `WHERE` $e_o = e_i$

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion - Example EXISTS

Example

Original Query

```
SELECT *
FROM Employee E
WHERE EXISTS (SELECT *
              FROM Employee E2
              WHERE E.Dep = E2.Dep )
```

Excursion - Example EXISTS

Example

Rewritten Query

```
SELECT E.*
FROM Employee E,
    (SELECT DISTINCT Dep
    FROM Employee) AS sub
WHERE E.Dep = sub.Dep
```

Excursion - Example Scalar Aggregation

Scalar Subquery with Aggregation and = Correlation

- Subquery is scalar subquery
- q is ASPJ query
- Correlation is $e_o = e_i$ between
  - outer query correlated attribute e_o
  - inner query attribute or constant e_i
- Add group-by and turn into join
  - FROM ..., (SELECT DISTINCT e_i FROM q) AS q WHERE e_o = e_i

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion - Example Scalar Aggregation

Example

Original Query

```
SELECT *
FROM Employee E
WHERE E.Salary = (SELECT max(Salary)
                  FROM Employee E2
                  WHERE E.Dep = E2.Dep )
```

ILLINOIS INSTITUTE OF TECHNOLOGY

Excursion - Example Scalar Aggregation

Example

Rewritten Query

```
SELECT E.*
FROM Employee E,
     (SELECT max(Salary)
     FROM Employee
     GROUP BY Dep) AS sub
WHERE  E.Dep = sub.Dep AND max(Salary) = E.Salary
```

ILLINOIS INSTITUTE OF TECHNOLOGY

Un-nesting can be Hard (and Expensive)

Problems with Un-nesting and De-correlation

- Complex correlation expressions under aggregations
  - E.g., UDFs
  - ⇒no obvious way to turn WHERE into GROUP BY

Example

```
SELECT *
FROM S
WHERE a > (SELECT sum(b)
           FROM R WHERE f(R.b, a) > 5)
```

Un-nesting can be Hard (and Expensive)

Problems with Un-nesting and De-correlation

- Complex correlation expressions under aggregations
  - E.g., UDFs
  - ⇒no obvious way to turn WHERE into GROUP BY
- Nested subqueries can contain nested subqueries!
  - Correlated attributes can reference more than one level up

Example

```
SELECT *
FROM R
WHERE R.a IN (SELECT * FROM S
          WHERE S.b = R.a
          AND S.c IN (SELECT * FROM T
                WHERE T.d > R.a))
```

Un-nesting can be Hard (and Expensive)

Solutions

- Inject outer query in nested query
  - Group on and join on correlated attribute
- Recursive Un-nesting

ILLINOIS INSTITUTE OF TECHNOLOGY

Un-nesting can be Hard (and Expensive)

Example

```
SELECT *
FROM S
WHERE a > (SELECT sum(b)
           FROM R WHERE f(R.b, a) > 5)
```

⇒

```
SELECT *
FROM S,
     (SELECT sum(b) AS s, a
     FROM R,
          (SELECT DISTINCT a FROM S)
     WHERE f(R.b, a) > 5)
     GROUP BY a) sub
WHERE a > s AND sub.a = S.a
```

Excursion - Recap

- Nested subqueries powerful feature
  - Especially if using correlations
- Evaluation can be expensive
  - Need advanced optimizer that applies un-nesting and de-correlation rules

ILLINOIS INSTITUTE OF TECHNOLOGY

Outline

ILLINOIS INSTITUTE OF TECHNOLOGY

Overview

Approach

1. Extend relational algebra with nested subquery expressions
2. Apply Perm declarative definition to determine provenance
3. Create rewrite rules

Extended algebra

- C_{sub} expressions modelling a nested subquery expression
- q_{sub} nested queries
  - Normal algebra expressions

ILLINOIS INSTITUTE OF TECHNOLOGY

C_{sub} Expressions

- $e \in q_{sub}$: IN-subquery
- $e \notin q_{sub}$: NOT IN-subquery
- $\exists t \in (q_{sub}) : t \ op \ e$: ANY-subquery
- $\forall t \in (q_{sub}) : t \ op \ e$: ALL-subquery
- $\exists t \in q_{sub}$: EXISTS-subquery
- q_{sub}: Scalar-subquery

Example Nested Expressions

Example

```
SELECT *
FROM Employee e
WHERE EXISTS (SELECT *
              FROM Employee e2
              WHERE e2.Salary < 30)
```

Example Nested Expressions

Example

```
SELECT *
FROM Employee e
WHERE EXISTS (SELECT *
              FROM Employee e2
              WHERE e2.Salary < 30)
```

$\Rightarrow$

$$q = \sigma_{C_{sub}}(E)$$
$$C_{sub} = \exists t \in \sigma_{Salary<30}(E)$$
$$q_{sub} = \sigma_{Salary<30}(E)$$

Example Nested Expressions

Example

```
SELECT *
FROM Employee E
WHERE e.Salary >= ALL (SELECT salary
                       FROM Employee)
```

ILLINOIS INSTITUTE OF TECHNOLOGY

Example Nested Expressions

Example

```
SELECT *
FROM Employee E
WHERE e.Salary >= ALL (SELECT salary
                       FROM Employee)
```

$\Rightarrow$

$$q = \sigma_{C_{sub}}(E)$$

$$C_{sub} = \forall t \in (\pi_{Salary}(E)) : t \geq Salary$$

$$q_{sub} = \pi_{Salary}(E)$$

C_{sub} Expression Semantics

$$
\begin{aligned}
[[e \in q_{sub}]] &= \exists t \in Q_{sub} : t = e \\
[[e \notin q_{sub}]] &= \neg \exists t \in Q_{sub} : t = e \\
[[\exists t \in (q_{sub}) : t \; op \; e]] &= \exists t \in Q_{sub} : e \; op \; t \\
[[\forall t \in (q_{sub}) : t \; op \; e]] &= \forall t \in Q_{sub} : e \; op \; t \\
[[\exists t \in q_{sub}]] &= \exists t \in Q_{sub} \\
[[q_{sub}]] &= Q_{sub}
\end{aligned}
$$

ILLINOIS INSTITUTE OF TECHNOLOGY

Outline

1. Provenance for Nested Subqueries in Perm
   - Introduction and Nested Subqueries
   - Relational Algebra with Nested Subexpressions
   - Provenance of Subqueries and Compositional Rules
   - The Generic Rewrite Strategy
   - Alternative Rewrite Strategies
   - Recap

Overview

Approach

1. Extend relational algebra with nested subquery expressions
2. Apply Perm declarative definition to determine provenance
3. Create rewrite rules

ILLINOIS INSTITUTE OF TECHNOLOGY

Applying the Declarative Definition

Idea

- Consider algebra expression with nesting as single tree
- For now only selection subqueries $q = \sigma_C(q)$
  - C contains single sublink expression C_{sub}
- $\Rightarrow$Provenance is witness lists $< u, v >$
  - $u \in Q_{sub}$: tuple in subquery
  - $v \in Q$: tuple from outer query
- For tuple t in Q:
  - Quantified subqueries: either true or false
  - Scalar: `NULL` or constant

ILLINOIS INSTITUTE OF TECHNOLOGY

ANY-subqueries

Notation and Conventions

- $q = \sigma_C(q_1)$
- Nested expression $C_{Sub} = \exists t \in (q_{sub}) : t \; op \; e$
- Tuple $t \in Q_1$
- $Q_{sub}^{true}(t) = \{t' \mid t' \in Q_{sub} \wedge t.e \; op \; t'\}$
  - Tuples from the subquery for which the expression evaluates to true
- $Q_{sub}^{false}(t) = \{t' \mid t' \in Q_{sub} \wedge \neg(t.e \; op \; t')\}$
  - Tuples from the subquery for which the expression evaluates to false

ILLINOIS INSTITUTE OF TECHNOLOGY

ANY-subqueries Provenance

Case 1: C_{sub} evaluates to true

- **Intuition**: All tuples that make the nested expression true for tuple t belong to provenance
- Provenance is $Q_{Sub}^{true}(t)$

ILLINOIS INSTITUTE OF TECHNOLOGY

ANY-subqueries Provenance

Checking conditions

1. $[[op(\mathcal{PI}(op, t, I))]] = \{t^x\}$
   - Provenance contains only tuples t' for which $e(t)\ op\ t'$ evaluates to true
2. $\forall w \in \mathcal{PI}(op, t, I) : [[op(w)]] \neq \emptyset$
   - Selection condition is true for every tuple t' in provenance because $e(t)\ op\ t'$ is true
3. $w, w' \in \mathcal{W}(q, I) : w \prec w' \wedge w \in \mathcal{PI}(q, t, I) \Rightarrow w' \notin \mathcal{PI}(q, t, I)$
   - No $\perp$ values $\Rightarrow$holds!
4. $\nexists \mathcal{PI}(q, t) \supset \mathcal{P}' \subseteq \mathcal{W}(q, I) : \mathcal{P}' \models (1), (2), (3)$
   - adding tuple from Q_{sub}^{false} causes condition (2) to fail

ANY-subqueries Provenance

Case 1: C_{sub} evaluates to false

- **Intuition**: No tuples make the expression true for tuple t, all tuple contribute
- Provenance is $Q_{sub}^{false}(t) = Q_{sub}$

ILLINOIS INSTITUTE OF TECHNOLOGY

ANY-subqueries Provenance

Checking conditions

1. $[[op(\mathcal{PI}(op,t,I))]] = \{t^x\}$
   - Trivially holds $Q_{sub}^{false}(t) = Q_{sub}$
2. $\forall w \in \mathcal{PI}(op,t,I) : [[op(w)]] \neq \emptyset$
   - Evaluates also to false on single tuples
3. $w, w' \in \mathcal{W}(q,I) : w \prec w' \wedge w \in \mathcal{PI}(q,t,I) \Rightarrow w' \notin \mathcal{PI}(q,t,I)$
   - NO: $< t, \perp >$ would fulfill conditions
4. $\nexists \mathcal{P} \supset \mathcal{P}' \subseteq \mathcal{W}(q,I) : \mathcal{P}' \models (1),(2),(3)$
   - Trivially holds $Q_{sub}^{false}(t) = Q_{sub}$

ANY-subqueries Provenance

Solution

- Modify condition (3):
- Consider only the part of a witness list corresponding to outer query
- $\prec_{reg}$ instead of $\prec$
  - Only tested on tuples from provenance of non-nested queries
- **Motivation**: Will help with `ALL`-subqueries

ILLINOIS INSTITUTE OF TECHNOLOGY

ANY-subqueries Compositional Rule

Rule

$$q = \sigma_C(q_1)$$

$$C_{sub} = \exists t \in (q_{sub}) : t \; op \; e$$

$$\mathcal{PI}(q,t) = \begin{cases} \{< t, v >^{n \times m} | \; t^n \in Q_1 \wedge v^m \in Q_{sub}^{true}(t)\} & [[C_{sub}(t)]] = true \\ \{< t, v >^{n \times m} | \; t^n \in Q_1 \wedge v^m \in Q_{sub}(t)\} & \text{else} \end{cases}$$

ALL-subqueries Provenance

Case 1: C_{sub} evaluates to true

- **Intuition**: All tuples make the nested expression true for tuple t $\Rightarrow$ provenance is all tuples
- Provenance is $Q_{Sub}^{true}(t) = Q_{Sub}(t)$

ILLINOIS INSTITUTE OF TECHNOLOGY

ALL-subqueries Provenance

Checking conditions

1. $[[op(\mathcal{PI}(op, t, I))]] = \{t^x\}$
   - Trivially holds $Q^{true}_{sub}(t) = Q_{sub}$
2. $\forall w \in \mathcal{PI}(op, t, I) : [[op(w)]] \neq \emptyset$
   - Nested expression holds for every tuple t' in result Q_{sub} $\Rightarrow$fulfilled
3. $w, w' \in \mathcal{W}(q, I) : w \prec_{reg} w' \wedge w \in \mathcal{PI}(q, t, I) \Rightarrow w' \notin \mathcal{PI}(q, t, I)$
   - $< t, \perp >$ would be witness $\Rightarrow$provenance would be empty with $\prec$
4. $\nexists \mathcal{P} \supset \mathcal{P}' \subseteq \mathcal{W}(q, I) : \mathcal{P}' \models (1), (2), (3)$
   - Trivially holds $Q^{true}_{sub}(t) = Q_{sub}$

ALL-subqueries Provenance

Case 1: C_{sub} evaluates to false

- **Intuition**: Some tuple tuples make the expression false for tuple t
- Provenance is $Q_{sub}^{false}(t)$

ALL-subqueries Provenance

Checking conditions

1. $[[op(\mathcal{PI}(op, t, I))]] = \{t^x\}$
   - Provenance only contains tuples for which the nested expression evaluates to false
2. $\forall w \in \mathcal{PI}(op, t, I) : [[op(w)]] \neq \emptyset$
   - Provenance only contains tuples for which the nested expression evaluates to false
3. $w, w' \in \mathcal{W}(q, I) : w \prec w' \wedge w \in \mathcal{PI}(q, t, I) \Rightarrow w' \notin \mathcal{PI}(q, t, I)$
   - $\perp$ not in provenance because ALL-subqueries evaluate to true over empty input
4. $\nexists \mathcal{P} \supset \mathcal{P}' \subseteq \mathcal{W}(q, I) : \mathcal{P}' \models (1), (2), (3)$
   - Adding a tuple from Q_{sub}^{true} would break condition (2)

ALL-subqueries Compositional Rule

Rule

$$q = \sigma_C(q_1)$$

$$C_{sub} = \forall t \in (q_{sub}) : t \ op \ e$$

$$\mathcal{PI}(q,t) = \begin{cases} \{< t, v >^{n \times m} | \ t^n \in Q_1 \wedge v^m \in Q_{sub}(t)\} & [[C_{sub}(t)]] = true \\ \{< t, v >^{n \times m} | \ t^n \in Q_1 \wedge v^m \in Q_{sub}^{false}(t)\} & \text{else} \end{cases}$$

EXISTS- and Scalar-subqueries Provenance

C_{sub}

- **Intuition**: Every result tuple influences result of nested expression
- Provenance is $Q_{sub}(t) = Q_{sub}(t)$

ILLINOIS INSTITUTE OF TECHNOLOGY

EXISTS- and Scalar-subqueries Provenance

Checking conditions

1. $[[op(\mathcal{PI}(op, t, I))]] = \{t^x\}$
   - Trivially holds provenance equals $Q_{sub}(t)$
2. $\forall w \in \mathcal{PI}(op, t, I) : [[op(w)]] \neq \emptyset$
   - EXISTS subquery evaluates to true for single tuples, scalar subquery has only single result
3. $w, w' \in \mathcal{W}(q, I) : w \prec w' \wedge w \in \mathcal{PI}(q, t, I) \Rightarrow w' \notin \mathcal{PI}(q, t, I)$
   - $< t, \perp >$ $\Rightarrow$EXISTS evaluates to false
4. $\nexists \mathcal{P} \supset \mathcal{P}' \subseteq \mathcal{W}(q, I) : \mathcal{P}' \models (1), (2), (3)$
   - Trivially holds provenance equals $Q_{sub}(t)$

OF TECHNOLOGY

EXISTS- and Scalar-subqueries Compositional Rule

Rule

$$q = \sigma_C(q_1)$$
$$C_{sub} = \forall t \in (q_{sub}) : t \; op \; e$$
$$\mathcal{PI}(q, t) = \{< t, v >^{n \times m} |\; t^n \in Q_1 \wedge v^m \in Q_{sub}(t)\}$$

ILLINOIS INSTITUTE OF TECHNOLOGY

Example

Example

$$q = \sigma_{C_{sub}}(E)$$
$$C_{sub} = \forall t \in (q_{sub}) : t \geq Salary$$
$$q_{sub} = \pi_{Salary}(\sigma_{Dep=Dep'}(\pi_{Salary,Dep \rightarrow Dep'}(E)))$$

```
SELECT *
FROM Employee E
WHERE e.Salary >= ALL (SELECT salary
                      FROM Employee E2.Dep
                      WHERE E.Dep = E2.Dep)
```

Employee

	Id	Name	Salary	Dep
e_1	1	Peter	100	CS
e_2	2	Gertrud	67	CS
e_3	3	Michael	22	HR

Result

	Id	Name	Salary	Dep
t_1	1	Peter	100	CS
t_2	3	Michael	22	HR

Example

Example

$$q = \sigma_{C_{sub}}(E)$$
$$C_{sub} = \forall t \in (q_{sub}) : t \geq Salary$$
$$q_{sub} = \pi_{Salary}(\sigma_{Dep=Dep'}(\pi_{Salary,Dep \to Dep'}(E)))$$

$$\mathcal{PI}(q, t_1) = \{< e_1, e_1 >, < e_1, e_2 >\}$$
$$\mathcal{PI}(q, t_2) = \{< e_3, e_3 >\}$$

Employee

	Id	Name	Salary	Dep
e_1	1	Peter	100	CS
e_2	2	Gertrud	67	CS
e_3	3	Michael	22	HR

Result

	Id	Name	Salary	Dep
t_1	1	Peter	100	CS
t_2	3	Michael	22	HR

Multiple Subqueries

Problem

- Ambiguous: more than one solution fulfills conditions

ILLINOIS INSTITUTE OF TECHNOLOGY

Multiple Subqueries

Problem

- Ambiguous: more than one solution fulfills conditions

Example

$$q = \sigma_{C_1 \vee C_2}(U) \quad C_1 = \exists t \in (R) : t = c \quad C_2 = \forall t \in (S) : t < c$$

R

	a
r_1	1
r_2	2
	...
r_5	5
	...
r_{100}	100

S

	b
s_1	1
s_2	5

U

	c
u_1	5

Q

	a
t_1	5

Multiple Subqueries

Problem

- Ambiguous: more than one solution fulfills conditions

Example

$$q = \sigma_{C_1 \vee C_2}(U) \quad C_1 = \exists t \in (R) : t = c \quad C_2 = \forall t \in (S) : t < c$$

Solution 1: $\mathcal{PI}(q, t) = \{< u_1, r_5, s_1 >, < u_1, r_5, s_2 >\}$

R

	a
r_1	1
r_2	2
	...
r_5	5
	...
r_{100}	100

S

	b
s_1	1
s_2	5

U

	c
u_1	5

Q

	a
t_1	5

Multiple Subqueries

Problem

- Ambiguous: more than one solution fulfills conditions

Example

$$q = \sigma_{C_1 \vee C_2}(U) \quad C_1 = \exists t \in (R) : t = c \quad C_2 = \forall t \in (S) : t < c$$

Solution 1: $\mathcal{PI}(q,t) = \{< u_1, r_5, s_1 >, < u_1, r_5, s_2 >\}$
Solution 2: $\mathcal{PI}'(q,t) = \{< u_1, r_1, s_1 >, ..., < u_1, r_{100}, s_1 >\}$

R

	a
r_1	1
r_2	2
	...
r_5	5
	...
r_{100}	100

S

	b
s_1	1
s_2	5

U

	c
u_1	5

Q

	a
t_1	5

Multiple Subqueries - Solution

- Cause of ambiguity: the definition does not force that C_{sub} evaluates to the same result over the provenance as over the original subquery result
- $\Rightarrow$Add condition to definition
  - enforcing that for all w in provenance: $C_{sub}(Q_{sub}, t) = C_{sub}(w, t)$

ILLINOIS INSTITUTE OF TECHNOLOGY

Outline

Overview

Approach

1. Extend relational algebra with nested subquery expressions
2. Apply Perm declarative definition to determine provenance
3. Create rewrite rules

Rewrite Rules for Nested Subqueries

- Generic Rule (**Gen** strategy)
  - Works for all nested subqueries
  - Expensive
  - ⇒Use as fallback if no better strategy applicable
- Un-nesting and De-correlation based rules
  - More efficient
  - Only applicable if preconditions fulfilled
  - Adapt query un-nesting for provenance computation

Overview

Approach

1. Extend relational algebra with nested subquery expressions
2. Apply Perm declarative definition to determine provenance
3. Create rewrite rules

Rewrite Rules for Nested Subqueries

- $\Rightarrow$ Generic Rule (**Gen** strategy)
  - Works for all nested subqueries
  - Expensive
  - $\Rightarrow$Use as fallback if no better strategy applicable
- Un-nesting and De-correlation based rules
  - More efficient
  - Only applicable if preconditions fulfilled
  - Adapt query un-nesting for provenance computation

Gen strategy

Why provenance for nested subqueries is hard?

- (1) How to access results of a nested query for provenance computation?
  - Put query into `FROM`?
  - ⇒Have to un-nest if uses correlation
  - ⇒Need special un-nesting for provenance
  - Recall that un-nesting can be hard and expensive!
- (2) How to determine result of nested subexpression C_{sub}?
  - Need this to determine provenance
  - Hard to compute without nesting if has universal quantification

Gen strategy

Approach

1. Join outer query with all potential witness lists
   - ⇒Crossproduct with relations accessed by nested subquery
   - ⇒Get potential provenance without un-nesting!
2. Rewrite nested subqueries using standard rules
   - No new rules needed
   - Problem: the rewritten subqueries can only be used in nested expressions
3. Simulate join between **(1)** (potential provenance) and **(2)** (real provenance) using correlations
   - Add **(2)** as nested expressions
   - Add nested expressions that determines result of C_{sub}
   - Add equality conditions

(1) - Join Outer Query with all Potential Witness Lists

Potential Witness Lists

- Subquery q_{sub} that accessess relations $R_1, \ldots, R_n$
- Recall: $\mathcal{W}(q_{sub}) = (R_1 \cup \{\bot\}) \times \ldots \times (R_n \cup \{\bot\})$
- Is algebra expression ⇒just rename attributes to match naming convention:
  - $\pi_{\mathbf{R_1} \rightarrow \mathcal{P}(R_1)}(R_1 \cup \{\bot\}) \times \ldots \times \pi_{\mathbf{R_n} \rightarrow \mathcal{P}(R_n)}(R_n \cup \{\bot\})$

Join with Potential Witness Lists

- Query with nested subquery $q = \sigma_{C_{sub}}(q_1)$

$$q^P = \sigma_{C_{sub}}(q_1 \times \pi_{\mathbf{R_1} \rightarrow \mathcal{P}(R_1)}(R_1 \cup \{\bot\}) \times \ldots \times \pi_{\mathbf{R_n} \rightarrow \mathcal{P}(R_n)}(R_n \cup \{\bot\}))$$

OF TECHNOLOGY

(1) - Join Outer Query with all Potential Witness Lists

Join with Potential Witness Lists

- Query with nested subquery $q = \sigma_{C_{sub}}(q_1)$

$$q^P = \sigma_{C_{sub}}(q_1 \times \pi_{\mathbf{R_1} \rightarrow \mathcal{P}(R_1)}(R_1 \cup \{\bot\}) \times \ldots \times \pi_{\mathbf{R_n} \rightarrow \mathcal{P}(R_n)}(R_n \cup \{\bot\}))$$

Example

```
SELECT *
FROM R
WHERE EXISTS (SELECT *
              FROM S)
```

(1) - Join Outer Query with all Potential Witness Lists

Join with Potential Witness Lists

- Query with nested subquery $q = \sigma_{C_{sub}}(q_1)$

$$q^P = \sigma_{C_{sub}}(q_1 \times \pi_{\mathbf{R_1} \rightarrow \mathcal{P}(R_1)}(R_1 \cup \{\perp\}) \times \ldots \times \pi_{\mathbf{R_n} \rightarrow \mathcal{P}(R_n)}(R_n \cup \{\perp\}))$$

Example

```
SELECT *
FROM R
WHERE EXISTS (SELECT *
              FROM S)
```

$\Rightarrow$

```
SELECT *
FROM R,
     (SELECT b AS P(b) FROM S UNION SELECT NULL AS P(b)) AS wit
WHERE EXISTS (SELECT *
              FROM S)
```

(2) - Rewrite Nested Queries

Approach

- Use standard rewrite rules
- Recursive application of Gen strategy if nested query has nested subqueries

Example

```
SELECT * FROM S
```

ILLINOIS INSTITUTE OF TECHNOLOGY

(2) - Rewrite Nested Queries

Approach

- Use standard rewrite rules
- Recursive application of Gen strategy if nested query has nested subqueries

Example

```
SELECT * FROM S
```

ILLINOIS INSTITUTE OF TECHNOLOGY

(2) - Rewrite Nested Queries

Approach

- Use standard rewrite rules
- Recursive application of Gen strategy if nested query has nested subqueries

Example

```
SELECT * FROM S
```

$\Rightarrow$

```
SELECT b, b AS P(b) FROM S
```

(3) - Simulate Join using Correlations

Joining Potential with Real Provenance

- **(a)** Compute C_{sub} to determine which tuples belong to provenance
- **(b)** Filter out these tuples by adding correlations that equate potential provenance with real provenance

ILLINOIS INSTITUTE OF TECHNOLOGY

(3) - Query Rewrite

- Have to distinguish two cases:
  1. $Q_{sub} \neq \emptyset$: determine provenance of subquery and simulate join
  2. $Q_{sub} = \emptyset$ and provenance is $< t, \perp, \ldots, \perp >$
- $\mathcal{P}(q_{sub}) = X$: Simulated join
- J_{sub}: use result of C_{sub} to filter provenance

$$
\begin{aligned}
q =& \sigma_{C_{sub}}(q_1) \\
q^+ =& \sigma_{C_{sub} \wedge C_{sub}{}^+}(q_1 \times \pi_{\mathbf{R_1} \rightarrow \mathcal{P}(R_1)}(R_1 \cup \{\perp\}) \\
& \times \ldots \times \pi_{\mathbf{R_n} \rightarrow \mathcal{P}(R_n)}(R_n \cup \{\perp\})) \\
C_{sub}{}^+ =& [\exists t \in \sigma_{J_{sub} \wedge \mathcal{P}(q_{sub}) = X}(\pi_{\mathcal{P}(q_{sub}) \rightarrow X}(q_{sub}{}^+))] \\
& \vee [\neg \exists t \in q_{sub} \wedge \mathcal{P}(q_{sub}) \text{ is } \varepsilon)]
\end{aligned}
$$

(3) - Filtering Provenance using J_{sub}

ANY-subqueries

- ANY-subquery $C_{sub} = \exists t \in (q_{sub}) : t\ op\ e$
- Recall
  - if C_{sub} is true then provenance is $Q_{sub}^{true}(t)$
  - if C_{sub} is false then provenance is $Q_{sub}(t)$
- Define: $C'_{sub} = e\ op\ t$
- Filter tuples from $Q_{sub}^{true}(t)$: $\sigma_{C'_{sub}}(q_{sub}(t))$
- $\Rightarrow J_{sub} = C_{sub} \wedge C'_{sub} \vee \neg C_{sub} = C'_{sub} \vee \neg C_{sub}$

ILLINOIS INSTITUTE OF TECHNOLOGY

(3) - Filtering Provenance using J_{sub}

ALL-subqueries

- ALL-subquery $C_{sub} = \forall t \in (q_{sub}) : t\ op\ e$
- Recall
  - if C_{sub} is true then provenance is $Q_{sub}(t)$
  - if C_{sub} is false then provenance is $Q_{sub}^{false}(t)$
- Use: $C'_{sub} = e\ op\ t$
- $\Rightarrow J_{sub} = C_{sub} \vee (\neg C_{sub} \wedge \neg C'_{sub}) = C_{sub} \vee \neg C'_{sub}$

ILLINOIS INSTITUTE OF TECHNOLOGY

(3) - Filtering Provenance using J_{sub}

EXISTS-subqueries

- ALL-subquery $C_{sub} = \exists t \in q_{sub}$
- Recall
  - provenance contains all tuples form $Q_{sub}(t)$
- $\Rightarrow J_{sub} = true$

ILLINOIS INSTITUTE OF TECHNOLOGY

Gen Strategy Example

Example

```
SELECT *
FROM R
WHERE EXISTS (SELECT *
              FROM S)
```

Gen Strategy Example

Example

```
SELECT R.a, R.a AS P(a), wit.P(b)
FROM
    R,
    (SELECT S.b AS prov_S_b
    FROM S
    UNION ALL
    SELECT NULL AS b) AS wit
WHERE
    EXISTS ( SELECT * FROM S)) AND
    (
        (EXISTS (SELECT S.b, S.b AS P(b)_X
                FROM s
                WHERE NOT S.b IS DISTINCT FROM P(b))
        OR
        (NOT EXISTS (SELECT * FROM S) AND P(b) IS NULL)
    )
```

Recap Gen Strategy

Advantages

- Works for all nested subqueries
- Single rewrite rule

Disadvantages

- Blows up size of query
- Simulated join using correlations is hard to optimize
- . . . and if not optimized will cause crossproduct in outer query

ILLINOIS INSTITUTE OF TECHNOLOGY

Outline

Overview

Rationale

- Adapt un-nesting and de-correlation rewrite for provenance computation

Concerns

- Rewritten nested subquery can still be joined?
- Can evaluate J_{sub} like expression to filter provenance?

ILLINOIS INSTITUTE OF TECHNOLOGY

Left strategy

Preconditions

1. No correlations

Rationale

- Rewritten subquery can be executed as non-nested query
- $\Rightarrow$Evaluate J_{sub} as join condition

Rules

$$q = \sigma_{C_{sub}}(q_1)$$
$$q^{+} = \pi_{\mathbf{Q_1},\mathcal{P}(q)}(\sigma_{C_{sub}}(q_1 ⟕_{J_{sub}} {q_{sub}}^{+})$$

Unn strategy

Preconditions

1. No correlations
2. ANY- or EXISTS-subquery

Rationale

- Join with provenance and evaluation of nested expression can be done in one step

Rules

$$q = \sigma_{C_{sub}}(q_1)$$

$$q^+ = \pi_{\mathbf{Q_1}, \mathcal{P}(q)}(q_1 \bowtie_{C'_{sub}} {q_{sub}}^+)$$

$$C'_{sub} = \begin{cases} (e \ op \ t) & \text{if} \\ true & \text{else (EXISTS)} \end{cases}$$

Example - Applying Unn-strategy

Example

$$q = \sigma_{\exists t \in S}(R)$$

```
SELECT *
FROM R
WHERE EXISTS (SELECT *
              FROM S)
```

Example - Applying Unn-strategy

Example

$$q = \sigma_{\exists t \in S}(R)$$

$$q^{+} = \pi_{a,\mathcal{P}(b)}(q_1 \bowtie_{true} \pi_{b,b \rightarrow \mathcal{P}(b)}(S))$$

```
SELECT *
FROM R
WHERE EXISTS (SELECT *
              FROM S)
```

$\Rightarrow$

```
SELECT R.a, R.a AS P(a), resub.P(b)
FROM R,
     (SELECT S.b, S.b AS P(b) FROM s) AS resub;
```

More Strategies

- **Move**: move subqueries to projections to be able to avoid repeated evaluation of subexpressions by **Left**
- **Unn-Not**: For negated uncorrelated `EXISTS`- or `ANY`-subqueries. Rewrite by using outer join and to model non-existence.
- **JA**: For correlated `ANY`- and scalar subqueries with aggregations. Joins with rewritten query using group-by.
- **EXISTS**: For correlated `EXISTS`-subqueries. Turns correlation into join.

ILLINOIS INSTITUTE OF TECHNOLOGY

Outline

Recap

Provenance for Nested Subqueries

- Nested subqueries important but provenance computation is hard
- Quantification!
- Ambiguity for multiple nested subqueries
- $\Rightarrow$Extended definition

Recap

Gen-strategy

- Do not unnest
- Create all potential witness lists
- Standard rewrite for nested query
- Simulate join between provenance and potential witness lists using correlation
- Selection condition to filter out provenance according to nested expression result (J_{sub})

Recap

Un-nesting and De-correlation strategies

- Based on un-nesting in query optimization
- More efficient than Gen-strategy
- Only applicable for certain subqueries

ILLINOIS INSTITUTE OF TECHNOLOGY

Literature

Boris Glavic and Gustavo Alonso.
Provenance for Nested Subqueries.
In Proceedings of the 12th International Conference on Extending Database Technology (EDBT), 982-993, 2009.