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

I.1 Provenance Models and Systems

Boris Glavic

October 3, 2012

Introduction and Nested Subqueries

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



Introduction and Nested Subqueries

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 ...);
- \Rightarrow Need provenance support for that



Introduction and Nested Subqueries

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 ...);
- \Rightarrow 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

GY

Introduction and Nested Subqueries

Excursion: Nested Subqueries in SQL

Idea

• Quantification (boolean result):

- Property holds for all results of query
- Query produces results



Introduction and Nested Subqueries

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



Introduction and Nested Subqueries

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



Introduction and Nested Subqueries

Excursion: Where can I use Nested Subqueries

- Nested subquery expression evaluates to constant \Rightarrow
- WHERE: filter out tuples
 - e.g., SELECT * FROM R WHERE EXISTS (SELECT * FROM S)
 - \Rightarrow 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



Introduction and Nested Subqueries

Excursion: Types of Nested Subqueries

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



Introduction and Nested Subqueries

Excursion: Existential Quantified Subqueries

Syntax • EXISTS (q) • q can be any query Semantics

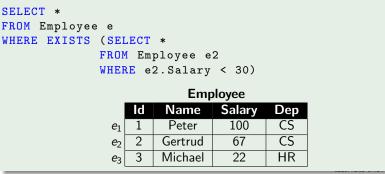
- Evaluates to . . .
 - true: q has non-empty result
 - false: q has empty result



Introduction and Nested Subqueries

Excursion: Existential Quantified Subqueries

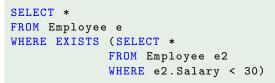
Example



Introduction and Nested Subqueries

Excursion: Existential Quantified Subqueries

Example





Introduction and Nested Subqueries

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

Introduction and Nested Subqueries

Excursion: Scalar Subqueries

Example



ILLINOIS INSTITUTE

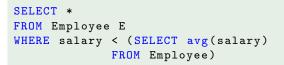
Slide 6 of 50

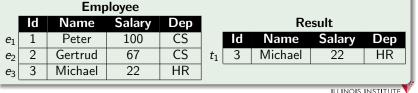
Boris Glavic

Introduction and Nested Subqueries

Excursion: Scalar Subqueries

Example





Introduction and Nested Subqueries

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 ∈ Q
 false: otherwise

Slide 7 of 50

Boris Glavic

CS 595 - Hot topics in database systems: Data Provenance

Introduction and Nested Subqueries

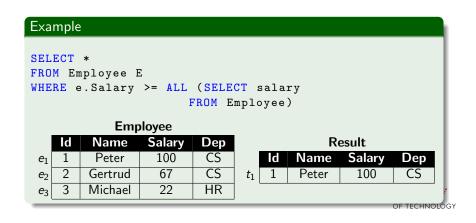
Excursion: Comparison Subqueries - Universal Quantification



Boris Glavic

Introduction and Nested Subqueries

Excursion: Comparison Subqueries - Universal Quantification



Introduction and Nested Subqueries

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

Slide 8 of 50

Boris Glavic

CS 595 - Hot topics in database systems: Data Provenance

GY

Introduction and Nested Subqueries

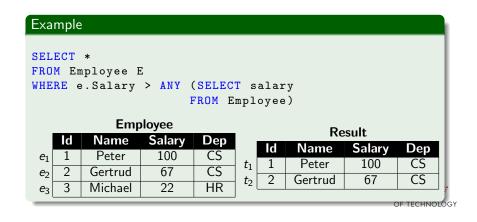
Excursion: Comparison Subqueries - Existential Quantification



Boris Glavic

Introduction and Nested Subqueries

Excursion: Comparison Subqueries - Existential Quantification



Boris Glavic

Introduction and Nested Subqueries

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)

Introduction and Nested Subqueries

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



Introduction and Nested Subqueries

Excursion - Correlated Subqueries

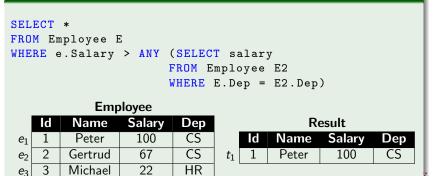
Example



Introduction and Nested Subqueries

Excursion - Correlated Subqueries

Example



Introduction and Nested Subqueries

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!

Introduction and Nested Subqueries

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



Introduction and Nested Subqueries

Excursion - Example EXISTS

$\overline{\mathsf{EXISTS}}$ with = correlation

- Subquery is **EXISTS** (q)
- q is SPJ query
- Correlation is $e_o = e_i$ between
 - outer query correlated attribute eo
 - inner query attribute or constant e_i
- Turn into Join

• FROM ..., (SELECT DISTINCT e_i FROM q) AS q WHERE $e_o = e_i$



Introduction and Nested Subqueries

Excursion - Example EXISTS

Example Original Query SELECT * FROM Employee E WHERE EXISTS (SELECT * FROM Employee E2 WHERE E.Dep = E2.Dep)



Slide 13 of 50

Boris Glavic

CS 595 - Hot topics in database systems: Data Provenance

Introduction and Nested Subqueries

Excursion - Example EXISTS

Example

Rewritten Query

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



Introduction and Nested Subqueries

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 eo
 - inner query attribute or constant e_i
- Add group-by and turn into join

• FROM ..., (SELECT DISTINCT e_i FROM q) AS q WHERE $e_o = e_i$



Introduction and Nested Subqueries

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)
```

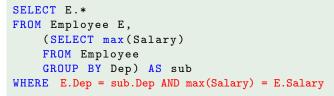


Introduction and Nested Subqueries

Excursion - Example Scalar Aggregation

Example

Rewritten Query





Introduction and Nested Subqueries

Un-nesting can be Hard (and Expensive)

Problems with Un-nesting and De-correlation

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

Example

GY

Introduction and Nested Subqueries

Un-nesting can be Hard (and Expensive)

Problems with Un-nesting and De-correlation

- Complex correlation expressions under aggregations
 - E.g., UDFs
 - \Rightarrow 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))
```

Slide 15 of 50

GY

Introduction and Nested Subqueries

Un-nesting can be Hard (and Expensive)

Solutions

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



Introduction and Nested Subqueries

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.
```

Slide 15 of 50

Boris Glavic

GY

Introduction and Nested Subqueries

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



Relational Algebra with Nested Subexpressions

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



Relational Algebra with Nested Subexpressions

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

Relational Algebra with Nested Subexpressions

C_{sub} Expressions

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



Relational Algebra with Nested Subexpressions

Example Nested Expressions

Example

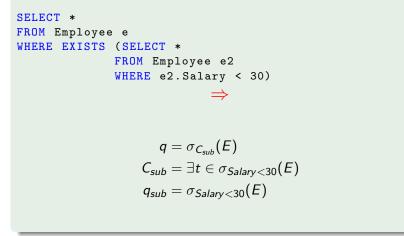
```
SELECT *
FROM Employee e
WHERE EXISTS (SELECT *
FROM Employee e2
WHERE e2.Salary < 30)
```



Relational Algebra with Nested Subexpressions

Example Nested Expressions

Example



Slide 19 of 50

Boris Glavic

CS 595 - Hot topics in database systems: Data Provenance

DGY

Relational Algebra with Nested Subexpressions

Example Nested Expressions

Example

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



CS 595 - Hot topics in database systems: Data Provenance

Slide 19 of 50

Boris Glavic

Relational Algebra with Nested Subexpressions

Example Nested Expressions

Example

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

$$\Rightarrow$$

 $q = \sigma_{C_{sub}}(E)$
 $C_{sub} = \forall t \in (\pi_{Salary}(E)) : t \ge Salary$
 $q_{sub} = \pi_{Salary}(E)$

Slide 19 of 50

Boris Glavic

Relational Algebra with Nested Subexpressions

C_{sub} Expression Semantics

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



Boris Glavic

Provenance of Subqueries and Compositional Rules

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



Provenance of Subqueries and Compositional Rules

Overview

Approach

1 Extend relational algebra with nested subquery expressions

2 Apply Perm declarative definition to determine provenance

3 Create rewrite rules



Slide 21 of 50

Provenance of Subqueries and Compositional Rules

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

Provenance of Subqueries and Compositional Rules

ANY-subqueries

Notation and Conventions

- $q = \sigma_C(q_1)$
- Nested expression $C_{Sub} = \exists t \in (q_{sub}) : t \text{ op } e$
- Tuple $t \in Q_1$

•
$$Q_{sub}^{true}(t) = \{t' \mid t' \in Q_{sub} \land t.e \text{ op } t'\}$$

• Tuples from the subquery for which the expression evaluates to true

•
$$Q_{sub}^{false}(t) = \{t' \mid t' \in Q_{sub} \land \neg(t.e \text{ op } t')\}$$

• Tuples from the subquery for which the expression evaluates to false

ILLINOIS INSTITUTE V OF TECHNOLOGY

Provenance of Subqueries and Compositional Rules

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 $Q_{Sub}^{true}(t)$



Provenance of Subqueries and Compositional Rules

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

$$\begin{array}{l} \textbf{3} \hspace{0.1 cm} w,w' \in \mathcal{W}(q,l): w \prec w' \land w \in \mathcal{PI}(q,t,l) \Rightarrow w' \notin \\ \mathcal{PI}(q,t,l) \end{array}$$

No ⊥ values ⇒holds!

• adding tuple from Q_{sub}^{false} causes condition (2) to fail

GY

Provenance of Subqueries and Compositional Rules

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
$$Q_{sub}^{false}(t) = Q_{sub}$$



Provenance of Subqueries and Compositional Rules

ANY-subqueries Provenance

Checking conditions

1
$$\begin{bmatrix} [op(\mathcal{PI}(op, t, l))] \end{bmatrix} = \{t^x\}$$

• Trivially holds $Q_{sub}^{false}(t) = Q_{sub}$
2 $\forall w \in \mathcal{PI}(op, t, l) : [[op(w)]] \neq \emptyset$
• Evaluates also to false on single tuples
3 $w, w' \in \mathcal{W}(q, l) : w \prec w' \land w \in \mathcal{PI}(q, t, l) \Rightarrow w' \notin \mathcal{PI}(q, t, l)$
• NO: $\langle t, \bot \rangle$ would fulfill conditions
4 $\mathcal{PP} \supset \mathcal{P}' \subseteq \mathcal{W}(q, l) : \mathcal{P}' \models (1), (2), (3)$
• Trivially holds $Q_{sub}^{false}(t) = Q_{sub}$

OF TECHNOLOGY

Provenance of Subqueries and Compositional Rules

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



Provenance of Subqueries and Compositional Rules

ANY-subqueries Compositional Rule

Rule

$$\begin{aligned} q &= \sigma_{\mathcal{C}}(q_1) \\ \mathcal{C}_{sub} &= \exists t \in (q_{sub}) : t \text{ op } e \\ \mathcal{PI}(q, t) &= \begin{cases} \{ < t, v >^{n \times m} \mid t^n \in Q_1 \land v^m \in Q_{sub}^{true}(t) \} & [[\mathcal{C}_{sub}(t)]] = true \\ \{ < t, v >^{n \times m} \mid t^n \in Q_1 \land v^m \in Q_{sub}(t) \} & \text{else} \end{cases} \end{aligned}$$



Slide 26 of 50 Boris Glavic

Provenance of Subqueries and Compositional Rules

ALL-subqueries Provenance

Case 1: C_{sub} evaluates to true

 Intuition: All tuples make the nested expression true for tuple t ⇒ provenance is all tuples

• Provenance is
$$Q_{Sub}^{true}(t) = Q_{Sub}(t)$$



Provenance of Subqueries and Compositional Rules

ALL-subqueries Provenance

Checking conditions

$$[[op(\mathcal{PI}(op, t, I))]] = \{t^x\}$$

• Trivially holds $Q_{sub}^{true}(t) = Q_{sub}$

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

$$\textbf{3} \quad w, w' \in \mathcal{W}(q, I) : w \prec_{reg} w' \land w \in \mathcal{PI}(q, t, I) \Rightarrow w' \notin \mathcal{PI}(q, t, I)$$

• < t, \perp > would be witness \Rightarrow provenance would be empty with \prec

• Trivially holds
$$Q_{sub}^{true}(t) = Q_{sub}$$

GY

Provenance of Subqueries and Compositional Rules

ALL-subqueries Provenance

Case 1: C_{sub} evaluates to false

• Intuition: Some tuple tuples make the expression false for tuple *t*

• Provenance is $Q_{sub}^{false}(t)$



Provenance of Subqueries and Compositional Rules

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

$$\begin{array}{l} \textbf{3} \hspace{0.1cm} w,w' \in \mathcal{W}(q,l) : w \prec w' \land w \in \mathcal{PI}(q,t,l) \Rightarrow w' \notin \\ \mathcal{PI}(q,t,l) \end{array}$$

• \perp not in provenance because ALL-subqueries evaluate to true over empty input

• Adding a tuple from Q_{sub}^{true} would break condition (2)

GY

Provenance of Subqueries and Compositional Rules

ALL-subqueries Compositional Rule

Rule

$$\begin{split} q &= \sigma_{C}(q_{1}) \\ C_{sub} &= \forall t \in (q_{sub}) : t \text{ op } e \\ \mathcal{PI}(q,t) &= \begin{cases} \{ < t, v >^{n \times m} | \ t^{n} \in Q_{1} \land v^{m} \in Q_{sub}(t) \} & [[C_{sub}(t)]] = true \\ \{ < t, v >^{n \times m} | \ t^{n} \in Q_{1} \land v^{m} \in Q_{sub}^{false}(t) \} & \text{else} \end{cases} \end{split}$$



Slide 29 of 50 Boris Glavic

Provenance of Subqueries and Compositional Rules

EXISTS- and Scalar-subqueries Provenance

C_{sub}

• Intuition: Every result tuple influences result of nested expression

• Provenance is
$$Q_{sub}(t) = Q_{sub}(t)$$



Provenance of Subqueries and Compositional Rules

EXISTS- and Scalar-subqueries Provenance

Checking conditions

$$[[op(\mathcal{PI}(op, t, I))]] = \{t^x\}$$

• 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

• $< t, \perp > \Rightarrow$ EXISTS evaluates to false

• Trivially holds provenance equals $Q_{sub}(t)$

Provenance of Subqueries and Compositional Rules

EXISTS- and Scalar-subqueries Compositional Rule

Rule

$$egin{aligned} q &= \sigma_{C}(q_{1}) \ C_{sub} &= orall t \in (q_{sub}): t \text{ op } e \ \mathcal{PI}(q,t) &= \{ < t, v >^{n imes m} | \ t^{n} \in Q_{1} \wedge v^{m} \in Q_{sub}(t) \} \end{aligned}$$



Slide 31 of 50 Boris Glavic

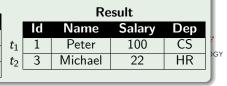
Provenance of Subqueries and Compositional Rules

Example

Example

Employee

	ld	Name	Salary	Dep
e_1	1	Peter	100	CS
e ₂	2	Gertrud	67	CS
e ₃	3	Michael	22	HR



Provenance of Subqueries and Compositional Rules

Example

Example

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

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

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

Employee

1	ld	Name	Salary	Dep	Result				
	lu					d	Name	Salary	Dep
e_1	1	Peter	100	CS	t_1	Iu	Ivanie	Salary	Бер
~I						1	Peter	100	CS
e_2	2	Gertrud	67	CS	-1	-		100	00
• <u>~</u>	_		•••		$\frac{1}{1}$ HR t_2	3	Michael	22	HR
e_3	- 3	Michael	22	∣ HR		Ŭ,			
-5	-				l				

GY

Decult

Provenance of Subqueries and Compositional Rules

Multiple Subqueries

Problem

• Ambiguous: more than one solution fulfills conditions



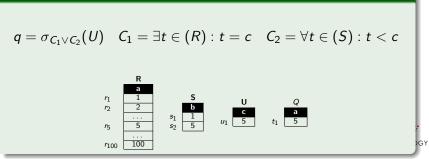
Provenance of Subqueries and Compositional Rules

Multiple Subqueries

Problem

Ambiguous: more than one solution fulfills conditions

Example



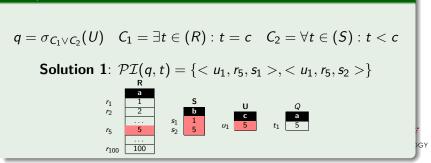
Provenance of Subqueries and Compositional Rules

Multiple Subqueries

Problem

• Ambiguous: more than one solution fulfills conditions

Example



Provenance of Subqueries and Compositional Rules

Multiple Subqueries

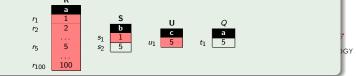
Problem

Ambiguous: more than one solution fulfills conditions

Example

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

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



Provenance of Subqueries and Compositional Rules

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: $C_{sub}(Q_{sub}, t) = C_{sub}(w, t)$



The Generic Rewrite Strategy

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

Rewrite Rules for Nested Subqueries

- 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

GY

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
 - \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

GY

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?
 - \Rightarrow Have to un-nest if uses correlation
 - \Rightarrow 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

ILLINOIS INSTITUTE

Gen strategy

Approach

- 1 Join outer query with all potential witness lists
 - \Rightarrow 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
- 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

GY

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

Potential Witness Lists

- Subquery q_{sub} that accessess relations R₁, ..., 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:

• $\pi_{\mathbf{R}_1 \to \mathcal{P}(R_1)}(R_1 \cup \{\bot\}) \times \ldots \times \pi_{\mathbf{R}_n \to \mathcal{P}(R_n)}(R_n \cup \{\bot\})$

Join with Potential Witness Lists

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

$$q^{P} = \sigma_{C_{sub}}(q_{1} \times \pi_{\mathbf{R}_{1} \to \mathcal{P}(R_{1})}(R_{1} \cup \{\bot\}) \times \ldots \times \pi_{\mathbf{R}_{n} \to \mathcal{P}(R_{n})}(R_{n} \cup \{\bot\}))$$

OF TECHNOLOGY

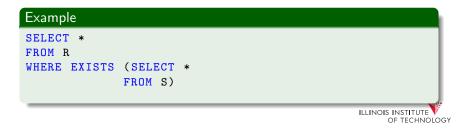
The Generic Rewrite Strategy

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

Join with Potential Witness Lists

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

$$q^{\mathcal{P}} = \sigma_{\mathcal{C}_{sub}}(q_1 \times \pi_{\mathbf{R}_1 \to \mathcal{P}(\mathcal{R}_1)}(\mathcal{R}_1 \cup \{\bot\}) \times \ldots \times \pi_{\mathbf{R}_n \to \mathcal{P}(\mathcal{R}_n)}(\mathcal{R}_n \cup \{\bot\}))$$



The Generic Rewrite Strategy

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

Join with Potential Witness Lists

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

$$q^{\mathcal{P}} = \sigma_{\mathcal{C}_{sub}}(q_1 \times \pi_{\mathbf{R}_1 \to \mathcal{P}(\mathcal{R}_1)}(\mathcal{R}_1 \cup \{\bot\}) \times \ldots \times \pi_{\mathbf{R}_n \to \mathcal{P}(\mathcal{R}_n)}(\mathcal{R}_n \cup \{\bot\}))$$

Example

```
SELECT *

FROM R

WHERE EXISTS (SELECT *

FROM S)

⇒

SELECT *

FROM R,

(SELECT b AS P(b) FROM S UNION SELECT NULL AS P(b)) AS wit

WHERE EXISTS (SELECT *

FROM S)
```

Slide 37 of 50

The Generic Rewrite Strategy

(2) - Rewrite Nested Queries

Approach

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

Example

SELECT * FROM S



Slide 38 of 50

Boris Glavic

The Generic Rewrite Strategy

(2) - Rewrite Nested Queries

Approach

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

Example

SELECT * FROM S



Slide 38 of 50

Boris Glavic

The Generic Rewrite Strategy

Approach

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

Example	
SELECT * FROM S	
SELECT b, b AS P(b) FROM S	
	OF TECHNOLOG

The Generic Rewrite Strategy

(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



The Generic Rewrite Strategy

(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 $\langle t, \bot, ..., \bot \rangle$
- $\mathcal{P}(q_{sub}) = X$: Simulated join
- J_{sub} : use result of C_{sub} to filter provenance

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

$$q^+ = \sigma_{C_{sub} \wedge C_{sub}^+}(q_1 \times \pi_{\mathbf{R}_1 \to \mathcal{P}(R_1)}(R_1 \cup \{\bot\}))$$

$$\times \ldots \times \pi_{\mathbf{R}_n \to \mathcal{P}(R_n)}(R_n \cup \{\bot\}))$$

$$C_{sub}^+ = \left[\exists t \in \sigma_{J_{sub} \wedge \mathcal{P}(q_{sub}) = X}(\pi_{\mathcal{P}(q_{sub}) \to X}(q_{sub}^+))\right]$$

$$\vee \left[\neg \exists t \in q_{sub} \land \mathcal{P}(q_{sub}) \text{ is } \varepsilon\right)\right]$$

Slide 40 of 50

Boris Glavic

CS 595 - Hot topics in database systems: Data Provenance

The Generic Rewrite Strategy

(3) - Filtering Provenance using J_{sub}

ANY-subqueries

• ANY-subquery $C_{sub} = \exists t \in (q_{sub}) : t \text{ op } e$

Recall

- if C_{sub} is true then provenance is $Q_{sub}^{true}(t)$
- if C_{sub} is false then provenance is $Q_{sub}(t)$

• Filter tuples from $Q_{sub}^{true}(t)$: $\sigma_{C'_{sub}}(q_{sub}(t))$

$$\bullet \Rightarrow J_{sub} = C_{sub} \land C'_{sub} \lor \neg C_{sub} = C'_{sub} \lor \neg C_{sub}$$



The Generic Rewrite Strategy

(3) - Filtering Provenance using J_{sub}

ALL-subqueries

- ALL-subquery $C_{sub} = \forall t \in (q_{sub}) : t \text{ op } e$
- 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:
$$C'_{sub} = e \ op \ t$$

•
$$\Rightarrow J_{sub} = C_{sub} \lor (\neg C_{sub} \land \neg C'_{sub}) = C_{sub} \lor \neg C'_{sub}$$



The Generic Rewrite Strategy

(3) - Filtering Provenance using
$$J_{sub}$$

EXISTS-subqueries

• ALL-subquery
$$C_{sub} = \exists t \in q_{sub}$$

• provenance contains all tuples form $Q_{sub}(t)$

•
$$\Rightarrow J_{sub} = true$$



The Generic Rewrite Strategy

Gen Strategy Example

Example

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



The Generic Rewrite Strategy

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
          UNTON 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
                                                                       GΥ
               (NOT EXISTS (SELECT * FROM S) AND P(b) IS NULL)
          )
Slide 42 of 50
              Boris Glavic
                                       CS 595 - Hot topics in database systems: Data Provenance
```

The Generic Rewrite Strategy

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



Alternative Rewrite Strategies

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



Alternative Rewrite Strategies

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?



Alternative Rewrite Strategies

Left strategy

Preconditions

No correlations

Rationale

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

Rules

$$egin{aligned} q &= \sigma_{\mathcal{C}_{sub}}(q_1) \ q^+ &= \pi_{\mathbf{Q}_1,\mathcal{P}(q)}(\sigma_{\mathcal{C}_{sub}}(q_1 ext{ } \mathcal{I}_{sub} ext{ } q_{sub}^+) \end{aligned}$$

Boris Glavic

Alternative Rewrite Strategies

Unn strategy

Preconditions

- No correlations
- 2 ANY- or EXISTS-subquery

Rationale

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

Rules

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

$$q^+ = \pi_{\mathbf{Q}_1, \mathcal{P}(q)}(q_1 \bowtie_{C'_{sub}} q_{sub}^+)$$

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

Alternative Rewrite Strategies

Example - Applying Unn-strategy

Example

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

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



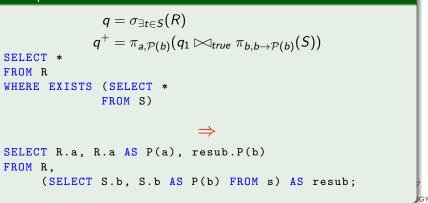
Slide 47 of 50

Boris Glavic

Alternative Rewrite Strategies

Example - Applying Unn-strategy

Example



Boris Glavic

Alternative Rewrite Strategies

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.



Recap

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



Recap

Recap

Provenance for Nested Subqueries

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



Recap

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

Recap

Un-nesting and De-correlation strategies

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



Recap

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.



Slide 50 of 50

Boris Glavic

CS 595 - Hot topics in database systems: Data Provenance