Optimizing Multiset Relational Algebra Queries using Weak-Equivalent Rewrite Rules

Jelle Hellings[1] Yuqing Wu[2] Dirk Van Gucht[3] Marc Gyssens[4]

[1]McMaster University, 1280 Main St. W., Hamilton, ON L8S 4L7, Canada

[2]Pomona College, 185 E 6th St., Claremont, CA 91711, USA

[3]Indiana University, 919 E 10th St., Bloomington, IN 47408, USA

[4]Hasselt University, Martelarenlaan 42, 3500, Hasselt, Belgium

Historical Context: Relation Algebra

Study of expressive power of graph and tree query languages.

Historical Context: Relation Algebra

Study of expressive power of graph and tree query languages.

<table>
<tr><td>id</td><td>$\cup$</td><td>$\circ$</td><td>$+$</td><td>$\smallfrown$</td><td>π</td><td>$\overline{\pi}$</td><td>$\cap$</td><td>$-$</td><td>di</td></tr>
<tr><td colspan="4">RPQs</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="5">2RPQs</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="6">Nested RPQs</td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="9">Navigational XPath, Graph XPath</td><td></td></tr>
<tr><td colspan="10">FO[3] + transitive closure</td></tr>
</table>

Historical Context: Relation Algebra

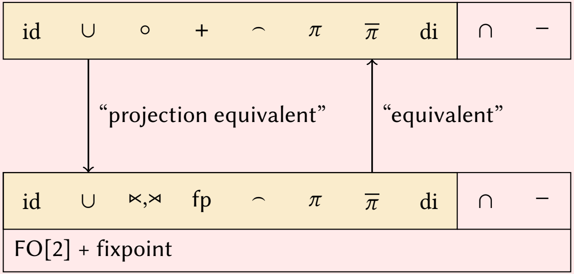
Study of expressive power of graph and tree query languages.

<table>
<tr><td>id</td><td>$\cup$</td><td>$\circ$</td><td>$+$</td><td>$\frown$</td><td>π</td><td>$\overline{\pi}$</td><td>$\cap$</td><td>$-$</td><td>di</td></tr>
<tr><td colspan="4">RPQs</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="5">2RPQs</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="6">Nested RPQs</td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="9">Navigational XPath, Graph XPath</td><td></td></tr>
<tr><td colspan="10">FO[3] + transitive closure</td></tr>
</table>

Trees JLAMP 2022, IS 2020, FoIKS 2018, DBPL 2015, ...

Graphs CJ 2020, DBPL 2017, AMAI 2015, FoIKS 2012, ICDT 2011, ...

Historical Context: Relation Algebra with Semi-Joins (CJ 2020, DBPL 2017)



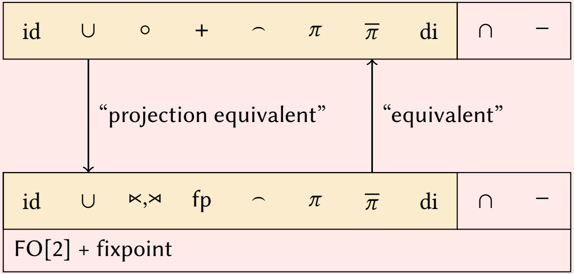
Main Result

If a relation algebra query is

- either a Boolean query or a node query; and
- does not use intersection and difference,

then the query can be evaluated *without* joins (composition, $\circ$) and transitive closures ($+$).

Historical Context: Relation Algebra with Semi-Joins (CJ 2020, DBPL 2017)



Consequence

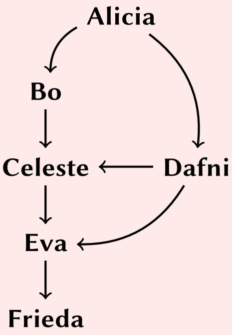
As there is a complexity difference between

- joins and semi-joins; and
- transitive closure and fixpoints,

we can *optimize* many (parts of) relation algebra query evaluation.

Graph Query Evaluation and SQL

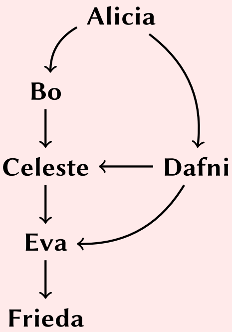
edges	
nfrom	nto



$$\pi_1[Edges \circ Edges \circ Edges \circ Edges].$$

Graph Query Evaluation and SQL

edges	
nfrom	nto



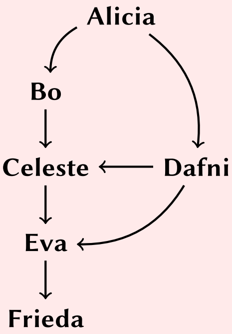
$$\pi_1[Edges \circ Edges \circ Edges \circ Edges].$$

```
SELECT DISTINCT S.nfrom
FROM edges S, edges R, edges T, edges U
WHERE S.nto = R.nfrom AND
      R.nto = T.nfrom AND
      T.nto = U.nfrom;
```

Graph Query Evaluation and SQL

edges

nfrom	nto



$$\pi_1[Edges \circ Edges \circ Edges \circ Edges].$$

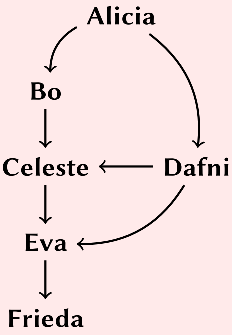
```
SELECT DISTINCT S.nfrom
FROM edges S, edges R, edges T, edges U
WHERE S.nto = R.nfrom AND
      R.nto = T.nfrom AND
      T.nto = U.nfrom;
```

Cost of query plan

Depending on system $\sim \mathcal{O}(|\text{edges}|^3)$–$\mathcal{O}(|\text{edges}|^4)$:
Queries do not complete.

Graph Query Evaluation and SQL

edges	
nfrom	nto



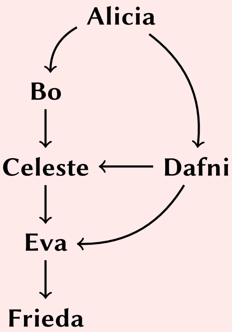
$$\pi_1[Edges \ltimes (Edges \ltimes (Edges \ltimes Edges))].$$

```
SELECT DISTINCT nfrom FROM edges
WHERE nto IN (
  SELECT nfrom FROM edges
  WHERE nto IN (
    SELECT nfrom FROM edges
    WHERE nto IN (
      SELECT nfrom FROM edges)));
```

Graph Query Evaluation and SQL

edges

nfrom	nto



$$\pi_1[Edges \ltimes (Edges \ltimes (Edges \ltimes Edges))].$$

```
SELECT DISTINCT nfrom FROM edges
WHERE nto IN (
  SELECT nfrom FROM edges
  WHERE nto IN (
    SELECT nfrom FROM edges
    WHERE nto IN (
      SELECT nfrom FROM edges)));
```

Cost of query plan

$\sim \mathcal{O}(|\text{edges}|)$: 90 ms with 75,000 edges.

Semi-Join Rewriting for SQL

Challenges

- From binary to *n*-ary relations.
- From set semantics to multiset semantics.
- More operations (e.g., aggregation).

Opportunities

- Constants and selections.
- Keys and primary keys.

An SQL-Based Example

Original query

```
SELECT DISTINCT C.cname, P.type
FROM customer C, bought B, product P
WHERE C.cname = B.cname AND B.pname = P.pname AND
      P.type = 'food';
```

An SQL-Based Example

Original query

```
SELECT DISTINCT C.cname, P.type
FROM customer C, bought B, product P
WHERE C.cname = B.cname AND B.pname = P.pname AND
        P.type = 'food';
```

Rewritten query

```
SELECT cname, 'food' AS type
FROM customer WHERE cname IN (
    SELECT cname FROM bought WHERE pname IN (
        SELECT pname FROM product WHERE type = 'food'));
```

An SQL-Based Example

Original query

```
SELECT DISTINCT C.cname, P.type
FROM customer C, bought B, product P
WHERE C.cname = B.cname AND B.pname = P.pname AND
      P.type = 'food';
```

Rewritten query

```
SELECT cname, 'food' AS type
FROM customer WHERE cname IN (
    SELECT cname FROM bought WHERE pname IN (
        SELECT pname FROM product WHERE type = 'food'));
```

Rewrite result

Performance gain of at-least 15%.

(Instance with 500 customers, 24 077 products, and 100 000 records in *Bought*).

A Formalization of the Multiset Relational Algebra

Customer

cname	*age*
Alice	19
Bob	20
Eve	21

Product

pname	*type*
apple	food
apple	fruit
banana	fruit
car	non-food

Bought

cname	*pname*	*price*
Alice	apple	0.35
Alice	apple	0.35
Bob	apple	0.45
Bob	banana	0.50
Eve	car	10000

- A *tuple t over names* $\mathcal{A}$ is a function mapping names to values:

$$t' = \{cname \mapsto \texttt{Alice}, pname \mapsto \texttt{apple}, price \mapsto \texttt{0.35}\}.$$

- A *relation (over* $\mathcal{A}$*)* is a set of tuples (over $\mathcal{A}$).
- A *multiset relation* $\tau_{\mathscr{R}}$ is a function mapping each tuple in relation $\mathscr{R}$ to a count:

$$\tau_{Bought}(t') = 2.$$

- A *database instance* $\mathfrak{I}$ maps each relation name into a multiset relation.

The Standard Multiset Relational Algebra

We write $[\![e]\!]_{\mathfrak{I}}$ to denote the evaluation of e on instance $\mathfrak{I}$.

Multiset relation $[\![R]\!]_{\mathfrak{I}} = \mathfrak{I}(R)$ with R a relation name.

Selection $[\![\dot{\sigma}_E(e)]\!]_{\mathfrak{I}} = \{(\mathrm{t} : n) \mid (\mathrm{t} : n) \in [\![e]\!]_{\mathfrak{I}} \land (\mathrm{t} \text{ satisfies } E)\}$.

Projection $[\![\dot{\pi}_{\mathrm{B}}(e)]\!]_{\mathfrak{I}} = \{(\mathrm{t}|_{\mathrm{B}} : \mathrm{count}(\mathrm{t}|_{\mathrm{B}}, e)) \mid \mathrm{t} \in [\![e]\!]_{\mathfrak{I}}\}$ with $\mathrm{count}(\mathrm{t}|_{\mathrm{B}}, e) = \sum_{((\mathrm{s}:m) \in [\![e]\!]_{\mathfrak{I}}) \land (\mathrm{s} \equiv_{\mathrm{B}} \mathrm{t})} m$.

Renaming $[\![\dot{\rho}_f(e)]\!]_{\mathfrak{I}} = \{(\mathrm{rename}(\mathrm{t}, f) : m) \mid (\mathrm{t} : m) \in [\![e]\!]_{\mathfrak{I}}\}$ with $\mathrm{rename}(\mathrm{t}, f) = \{f(a) \mapsto \mathrm{t}(a) \mid a \in \mathcal{S}(e; \mathfrak{D})\}$.

Deduplication $[\![\dot{\delta}(e)]\!]_{\mathfrak{I}} = \{(\mathrm{t} : 1) \mid (\mathrm{t} : n) \in [\![e]\!]_{\mathfrak{I}}\}$.

The Standard Multiset Relational Algebra

We write $[\![e]\!]_{\mathfrak{I}}$ to denote the evaluation of e on instance $\mathfrak{I}$.

Union, intersection, and difference

$$\begin{aligned}
[\![e_1 \mathbin{\dot\cup} e_2]\!]_{\mathfrak{I}} &= \{(\mathrm{t} : n_1 + n_2) \mid (\mathrm{t} : n_1) \in [\![e_1]\!]_{\mathfrak{I}} \land (\mathrm{t} : n_2) \in [\![e_2]\!]_{\mathfrak{I}}\} \cup \\
&\qquad \{(\mathrm{t} : n) \mid (\mathrm{t} : n) \in [\![e_1]\!]_{\mathfrak{I}} \land \mathrm{t} \notin [\![e_2]\!]_{\mathfrak{I}}\} \cup \\
&\qquad \{(\mathrm{t} : n) \mid \mathrm{t} \notin [\![e_1]\!]_{\mathfrak{I}} \land (\mathrm{t} : n) \in [\![e_2]\!]_{\mathfrak{I}}\}; \\
[\![e_1 \mathbin{\dot\cap} e_2]\!]_{\mathfrak{I}} &= \{(\mathrm{t} : \min(n_1, n_2)) \mid (\mathrm{t} : n_1) \in [\![e_1]\!]_{\mathfrak{I}} \land (\mathrm{t} : n_2) \in [\![e_2]\!]_{\mathfrak{I}}\}; \\
[\![e_1 \mathbin{\dot-} e_2]\!]_{\mathfrak{I}} &= \{(\mathrm{t} : n) \mid (\mathrm{t} : n) \in [\![e_1]\!]_{\mathfrak{I}} \land \mathrm{t} \notin [\![e_2]\!]_{\mathfrak{I}}\} \cup \\
&\qquad \{(\mathrm{t} : n_1 - n_2) \mid (n_1 > n_2) \land (\mathrm{t} : n_1) \in [\![e_1]\!]_{\mathfrak{I}} \land (\mathrm{t} : n_2) \in [\![e_2]\!]_{\mathfrak{I}}\}.
\end{aligned}$$

θ-join and natural join

$$\begin{aligned}
[\![e_1 \mathbin{\dot\bowtie}_E e_2]\!]_{\mathfrak{I}} &= \{(\mathrm{t}_1 \cdot \mathrm{t}_2 : n_1 \cdot n_2) \mid (\mathrm{t}_1 : n_1) \in [\![e_1]\!]_{\mathfrak{I}} \land (\mathrm{t}_2 : n_2) \in [\![e_2]\!]_{\mathfrak{I}} \land \\
&\qquad \mathrm{t}_1 \equiv_{\mathcal{S}(e_1;\mathfrak{D}) \cap \mathcal{S}(e_2;\mathfrak{D})} \mathrm{t}_2 \land (\mathrm{t}_1 \cdot \mathrm{t}_2 \text{ satisfies } E)\}; \\
[\![e_1 \mathbin{\dot\bowtie} e_2]\!]_{\mathfrak{I}} &= [\![e_1 \mathbin{\dot\bowtie}_{\emptyset} e_2]\!]_{\mathfrak{I}}
\end{aligned}$$

The Standard Multiset Relational Algebra – Example

Customer

cname	*age*
Alice	19
Bob	20
Eve	21

Product

pname	*type*
apple	food
apple	fruit
banana	fruit
car	non-food

Bought

cname	*pname*	*price*
Alice	apple	0.35
Alice	apple	0.35
Bob	apple	0.45
Bob	banana	0.50
Eve	car	10000

The query

$$e = \dot{\pi}_{age}(\dot{\sigma}_{type=\texttt{non-food}}(Customer \mathbin{\dot{\bowtie}} Bought \mathbin{\dot{\bowtie}} Product))$$

returns the ages of people that bought non-food products. We have:

$$[\![e]\!]_{\mathfrak{I}} = \{(age \mapsto 21 : 1)\}.$$

The Standard Multiset Relational Algebra – Example

Customer

cname	*age*
Alice	19
Bob	20
Eve	21

Product

pname	*type*
apple	food
apple	fruit
banana	fruit
car	non-food

Bought

cname	*pname*	*price*
Alice	apple	0.35
Alice	apple	0.35
Bob	apple	0.45
Bob	banana	0.50
Eve	car	10000

The query

$$e = \dot{\pi}_{age}(\dot{\sigma}_{type=\texttt{food}}(Customer \dot{\bowtie} Bought \dot{\bowtie} Product))$$

returns the ages of people that bought non-food products. We have:

$$[\![e']\!]_{\mathfrak{I}} = \{(age \mapsto \texttt{19} : 2), (age \mapsto \texttt{20} : 1)\}.$$

The Extended Multiset Relational Algebra

θ-semi-join and semi-join

$$\begin{aligned}\llbracket e_1 \dot{\ltimes}_E e_2 \rrbracket_{\mathfrak{I}} = \{(\mathrm{t}_1 : n_1) \mid (\mathrm{t}_1 : n_1) \in \llbracket e_1 \rrbracket_{\mathfrak{I}} \wedge \exists \mathrm{t}_2\, (\mathrm{t}_2 \in \llbracket e_2 \rrbracket_{\mathfrak{I}} \wedge \\ \mathrm{t}_1 \equiv_{\mathcal{S}(e_1;\mathfrak{D}) \cap \mathcal{S}(e_2;\mathfrak{D})} \mathrm{t}_2 \wedge (\mathrm{t}_1 \cdot \mathrm{t}_2 \text{ satisfies } E))\}; \\ \llbracket e_1 \dot{\ltimes} e_2 \rrbracket_{\mathfrak{I}} = \llbracket e_1 \dot{\ltimes}_{\emptyset} e_2 \rrbracket_{\mathfrak{I}}.\end{aligned}$$

Attribute introduction

$\llbracket i_f(e) \rrbracket_{\mathfrak{I}} = \{(\mathrm{t} \cdot \{\mathrm{B} \mapsto \mathrm{value}(\mathrm{t}, x) \mid (b := x) \in f\} : m) \mid (\mathrm{t} : m) \in \llbracket e \rrbracket_{\mathfrak{I}}\}$ with $\mathrm{value}(\mathrm{t}, x) = \mathrm{t}(x)$ if x is an attribute and $\mathrm{value}(\mathrm{t}, x) = x$ otherwise.

Max-union

$$\begin{aligned}\llbracket e_1 \dot{\sqcup} e_2 \rrbracket_{\mathfrak{I}} = \{(\mathrm{t} : \max(n_1, n_2)) \mid (\mathrm{t} : n_1) \in \llbracket e_1 \rrbracket_{\mathfrak{I}} \wedge (\mathrm{t} : n_2) \in \llbracket e_2 \rrbracket_{\mathfrak{I}}\} \cup \\ \{(\mathrm{t} : n) \mid (\mathrm{t} : n) \in \llbracket e_1 \rrbracket_{\mathfrak{I}} \wedge \mathrm{t} \notin \llbracket e_2 \rrbracket_{\mathfrak{I}}\} \cup \\ \{(\mathrm{t} : n) \mid \mathrm{t} \notin \llbracket e_1 \rrbracket_{\mathfrak{I}} \wedge (\mathrm{t} : n) \in \llbracket e_2 \rrbracket_{\mathfrak{I}}\}.\end{aligned}$$

The Extended Multiset Relational Algebra – Example

Customer

cname	*age*
Alice	19
Bob	20
Eve	21

Product

pname	*type*
apple	food
apple	fruit
banana	fruit
car	non-food

Bought

cname	*pname*	*price*
Alice	apple	0.35
Alice	apple	0.35
Bob	apple	0.45
Bob	banana	0.50
Eve	car	10000

The query

$$e = \dot{\delta}(\dot{\pi}_{age}(\dot{\sigma}_{type=\texttt{food}}(Customer \mathbin{\dot{\bowtie}} Bought \mathbin{\dot{\bowtie}} Product)))$$

is equivalent to

$$\dot{\delta}(\dot{\pi}_{age}(Customer \mathbin{\dot{\ltimes}} (Bought \mathbin{\dot{\ltimes}}_{type=\texttt{food}} Product))).$$

The Extended Multiset Relational Algebra – Example

Customer

cname	*age*
Alice	19
Bob	20
Eve	21

Product

pname	*type*
apple	food
apple	fruit
banana	fruit
car	non-food

Bought

cname	*pname*	*price*
Alice	apple	0.35
Alice	apple	0.35
Bob	apple	0.45
Bob	banana	0.50
Eve	car	10000

The query

$$e = \dot{\delta}(\dot{\pi}_{cname,type}(\dot{\sigma}_{type=\texttt{food}}(Customer \mathbin{\dot{\bowtie}} Bought \mathbin{\dot{\bowtie}} Product)))$$

is equivalent to

$$i_{type:=\texttt{food}}(\dot{\pi}_{cname}(Customer \mathbin{\dot{\ltimes}} (Bought \mathbin{\dot{\ltimes}}_{type=\texttt{food}} Product))).$$

Query Rewriting

Question

Can we rewrite

$$\dot{\delta}(\dot{\pi}_{age}(\dot{\sigma}_{type=\texttt{food}}(Customer \mathbin{\dot{\bowtie}} Bought \mathbin{\dot{\bowtie}} Product)))$$

into

$$\dot{\delta}(\dot{\pi}_{age}(Customer \mathbin{\dot{\ltimes}} (Bought \mathbin{\dot{\ltimes}}_{type=\texttt{food}} Product)))?$$

Query Rewriting

Question

Can we rewrite

$$\dot{\delta}(\dot{\pi}_{age}(\dot{\sigma}_{type=\texttt{food}}(Customer \dot{\bowtie} Bought \dot{\bowtie} Product)))$$

into

$$\dot{\delta}(\dot{\pi}_{age}(Customer \dot{\ltimes} (Bought \dot{\ltimes}_{type=\texttt{food}} Product)))?$$

Consider $Bought \dot{\bowtie}_{type=\texttt{food}} Product$

The query

$$Bought \dot{\bowtie}_{type=\texttt{food}} Product$$

is not equivalent to

$$Bought \dot{\ltimes}_{type=\texttt{food}} Product$$

but they do produce the same values for the attributes from *Bought*.

Notions of Query Equivalence

Definition

We say that e_1 and e_2 are

strong-equivalent (denoted by $e_1 \doteq e_2$), if we have $[\![e_1]\!]_{\mathfrak{I}} = [\![e_2]\!]_{\mathfrak{I}}$ for all instances $\mathfrak{I}$;

Notions of Query Equivalence

Definition

We say that e_1 and e_2 are

strong-equivalent (denoted by $e_1 \doteq e_2$), if we have $[\![e_1]\!]_{\mathfrak{I}} = [\![e_2]\!]_{\mathfrak{I}}$ for all instances $\mathfrak{I}$;

strong-B-equivalent (denoted by $e_1 \doteq_{\text{B}} e_2$) if $\dot{\pi}_{\text{B}}(e_1) \doteq \dot{\pi}_{\text{B}}(e_2)$;

Notions of Query Equivalence

Definition

We say that e_1 and e_2 are

- strong-equivalent (denoted by $e_1 \doteq e_2$), if we have $[\![e_1]\!]_{\mathfrak{I}} = [\![e_2]\!]_{\mathfrak{I}}$ for all instances $\mathfrak{I}$;
- strong-B-equivalent (denoted by $e_1 \doteq_{\mathrm{B}} e_2$) if $\dot{\pi}_{\mathrm{B}}(e_1) \doteq \dot{\pi}_{\mathrm{B}}(e_2)$;
- weak-equivalent (denoted by $e_1 \mathrel{\tilde{\doteq}} e_2$) if $\dot{\delta}(e_1) \doteq \dot{\delta}(e_2)$; and
- weak-B-equivalent (denoted by $e_1 \mathrel{\tilde{\doteq}}_{\mathrm{B}} e_2$) if $\dot{\pi}_{\mathrm{B}}(e_1) \mathrel{\tilde{\doteq}} \dot{\pi}_{\mathrm{B}}(e_2)$.

Notions of Query Equivalence

Definition

We say that e_1 and e_2 are

strong-equivalent (denoted by $e_1 \doteq e_2$), if we have $[\![e_1]\!]_{\mathfrak{I}} = [\![e_2]\!]_{\mathfrak{I}}$ for all instances $\mathfrak{I}$;

strong-B-equivalent (denoted by $e_1 \doteq_{\mathrm{B}} e_2$) if $\dot{\pi}_{\mathrm{B}}(e_1) \doteq \dot{\pi}_{\mathrm{B}}(e_2)$;

weak-equivalent (denoted by $e_1 \tilde{\doteq} e_2$) if $\dot{\delta}(e_1) \doteq \dot{\delta}(e_2)$; and

weak-B-equivalent (denoted by $e_1 \tilde{\doteq}_{\mathrm{B}} e_2$) if $\dot{\pi}_{\mathrm{B}}(e_1) \tilde{\doteq} \dot{\pi}_{\mathrm{B}}(e_2)$.

Example

$$\dot{\sigma}_{type=\mathsf{food}}(Bought \mathbin{\dot{\bowtie}} Product) \doteq Bought \mathbin{\dot{\bowtie}}_{type=\mathsf{food}} Product.$$

$$Bought \mathbin{\dot{\bowtie}}_{type=\mathsf{food}} Product \mathrel{\tilde{\doteq}}_{\{cname\}} Bought \mathbin{\dot{\ltimes}}_{type=\mathsf{food}} Product.$$

Conditions and Selections

If $e_1 \mathrel{\tilde{\dot{=}}}_{\{a\}} e_2$, then $\dot{\sigma}_{a=a',b=\mathsf{u}}(e_1) \mathrel{\tilde{\dot{=}}}_{\{a,a',b\}} \dot{\sigma}_{a=a',b=\mathsf{u}}(e_2)$.

Conditions and Selections

If $e_1 \mathrel{\dot{\tilde{=}}}_{\{a\}} e_2$, then $\dot{\sigma}_{a=a',b=\mathsf{u}}(e_1) \mathrel{\dot{\tilde{=}}}_{\{a,a',b\}} \dot{\sigma}_{a=a',b=\mathsf{u}}(e_2)$.

Definition

The *closure $\mathcal{C}(\textsc{b}; E)$ of attributes $\textsc{b}$ under equalities E* is the smallest superset of $\textsc{b}$ such that, for every condition $(v = w) \in E$ or $(w = v) \in E$, $v \in \mathcal{C}(\textsc{b}; E)$ if and only if $w \in \mathcal{C}(\textsc{b}; E)$.

Conditions and Selections

$$\text{If } e_1 \,\tilde{\dot{=}}_{\{a\}}\, e_2, \text{ then } \dot{\sigma}_{a=a',b=\mathsf{u}}(e_1) \,\tilde{\dot{=}}_{\{a,a',b\}}\, \dot{\sigma}_{a=a',b=\mathsf{u}}(e_2).$$

Definition

The *closure $C(\textsc{b}; E)$ of attributes $\textsc{b}$ under equalities E* is the smallest superset of $\textsc{b}$ such that, for every condition $(v = w) \in E$ or $(w = v) \in E$, $v \in C(\textsc{b}; E)$ if and only if $w \in C(\textsc{b}; E)$.

Lemma

If two tuples agree on attributes $\textsc{b}$ and satisfy equalities E, then they also agree on

- *all attributes in $C(\textsc{b}; E)$; and*
- *all attributes a that are constants due to E ($C(\{a\}; E)$ contains a constant).*

We write $\mathrm{det}(\textsc{b}; E)$ *to denote the above set of attributes.*

From Selection to Attribute Introduction

Theorem

Let g and h be expressions. We have

- *if* $\dot{\sigma}_E(g) \mathrel{\tilde{=}}_{\mathrm{A}} h$*, then* $\dot{\sigma}_E(g) \mathrel{\tilde{=}}_{\mathrm{A} \cup \mathrm{B}} i_f(h)$*; and*
- *if* $\dot{\sigma}_E(g) \doteq_{\mathrm{A}} h$*, then* $\dot{\sigma}_E(g) \doteq_{\mathrm{A} \cup \mathrm{B}} i_f(h)$

with f *a set of assignment-pairs such that, for each* $b \in \mathrm{B}$*, there exists* $(b := x) \in f$ *with* $x \in C(b; E)$ *and either* $x \in \mathrm{A}$ *or* x *is a constant.*

Example

We have

$$\dot{\sigma}_{type=\mathtt{food}}(Customer \mathbin{\dot{\bowtie}} Bought \mathbin{\dot{\bowtie}} Product) \mathrel{\tilde{=}}_{\{cname\}} \dot{\pi}_{cname}(C \mathbin{\dot{\ltimes}} (B \mathbin{\dot{\ltimes}}_{type=\mathtt{food}} P)).$$

Hence, we have

$$\dot{\sigma}_{type=\mathtt{food}}(Customer \mathbin{\dot{\bowtie}} Bought \mathbin{\dot{\bowtie}} Product) \mathrel{\tilde{=}}_{\{cname,type\}} i_{type:=\mathtt{food}}(\dot{\pi}_{cname}(C \mathbin{\dot{\ltimes}} (B \mathbin{\dot{\ltimes}}_{type=\mathtt{food}} P))).$$

Rewrite Rules

- Rules to work with strong-equivalences and weak-equivalences. E.g.,

  if $e_1 \tilde{\doteq}_{\textsc{b}} e_2$, e_1 and e_2 are set relations, and B is a key of e_1 and e_2, then $e_1 \doteq_{\textsc{b}} e_2$.

- Rules to pull attribute introduction up. E.g.,

  $$\dot{\sigma}_E(i_f(e)) \doteq i_f(\dot{\sigma}_E(e)) \text{ if, for all } (b := x) \in f,\ b \text{ is not used in } E.$$

- Rules to interact between attribute introduction and renaming.

Rewrite Rules

- Rules to work with strong-equivalences and weak-equivalences. E.g.,

  if $e_1 \mathrel{\tilde{\doteq}}_{\text{B}} e_2$, e_1 and e_2 are set relations, and B is a key of e_1 and e_2, then $e_1 \doteq_{\text{B}} e_2$.

- Rules to pull attribute introduction up. E.g.,

  $$\dot{\sigma}_E(i_f(e)) \doteq i_f(\dot{\sigma}_E(e)) \text{ if, for all } (b := x) \in f,\ b \text{ is not used in } E.$$

- Rules to interact between attribute introduction and renaming.

Tools that we use to introduce extra cases in which we can rewrite joins.

From Join to Semi-Join (Simplified)

Theorem

Let g_1 and h_1 be expressions with schema $\mathcal{A}_1$ and let g_2 and h_2 be expressions with schema $\mathcal{A}_2$. We have

1. $g_1 \dot{\bowtie}_E g_2 \mathrel{\tilde{\dot{=}}}_{\mathcal{A}_1} h_1 \dot{\ltimes}_E h_2$ *if* $g_1 \mathrel{\tilde{\dot{=}}}_{\mathcal{A}_1} h_1$ *and* $g_2 \mathrel{\tilde{\dot{=}}}_{\mathcal{A}_2} h_2$*; and*
2. $g_1 \dot{\bowtie}_E g_2 \doteq_{\mathcal{A}_1} h_1 \dot{\ltimes}_E h_2$ *if* $g_1 \doteq_{\mathcal{A}_1} h_1$, $g_2 \mathrel{\tilde{\dot{=}}}_{\mathcal{A}_2} h_2$, g_2 *is a set relation, and* g_2 *has a key* $\mathcal{C}$ *with* $\mathcal{C} \subseteq \mathrm{det}(\mathcal{A}_1 \cap \mathcal{A}_2; E)$.

From Join to Semi-Join (Simplified)

Theorem

Let g_1 and h_1 be expressions with schema $\mathtt{A}_1$ and let g_2 and h_2 be expressions with schema $\mathtt{A}_2$. We have

1. $g_1 \dot{\bowtie}_E g_2 \doteqdot_{\mathtt{A}_1} h_1 \dot{\ltimes}_E h_2$ *if* $g_1 \doteqdot_{\mathtt{A}_1} h_1$ *and* $g_2 \doteqdot_{\mathtt{A}_2} h_2$*; and*
2. $g_1 \dot{\bowtie}_E g_2 \doteq_{\mathtt{A}_1} h_1 \dot{\ltimes}_E h_2$ *if* $g_1 \doteq_{\mathtt{A}_1} h_1$, $g_2 \doteqdot_{\mathtt{A}_2} h_2$, g_2 *is a set relation, and* g_2 *has a key* $\mathtt{C}$ *with* $\mathtt{C} \subseteq \operatorname{det}(\mathtt{A}_1 \cap \mathtt{A}_2; E)$.

Let $\mathtt{B} \subseteq (\mathtt{A}_2 - \mathtt{A}_1)$ *and let* f *be a set of assignment-pairs such that, for each* $b \in \mathtt{B}$, *there exists* $(b := x) \in f$ *with* $x \in C(b; E)$ *and either* $x \in \mathtt{A}_1$ *or* x *is a constant. We also have*

3. $g_1 \dot{\bowtie}_E g_2 \doteqdot_{\mathtt{A}_1 \cup \mathtt{B}} i_f(h_1 \dot{\ltimes}_E h_2)$ *if* $g_1 \doteqdot_{\mathtt{A}_1} h_1$ *and* $g_2 \doteqdot_{\mathtt{A}_2} h_2$*; and*
4. $g_1 \dot{\bowtie}_E g_2 \doteq_{\mathtt{A}_1 \cup \mathtt{B}} i_f(h_1 \dot{\ltimes}_E h_2)$ *if* $g_1 \doteq_{\mathtt{A}_1} h_1$, $g_2 \doteqdot_{\mathtt{A}_2} h_2$, g_2 *is a set relation, and* g_2 *has a key* $\mathtt{C}$ *with* $\mathtt{C} \subseteq \operatorname{det}(\mathtt{A}_1 \cap \mathtt{A}_2; E)$.

Further Ingredients

- Rules to push down and eliminating deduplication. E.g.,

$$\dot{\delta}(e_1 \cup e_2) \doteq \dot{\delta}(e_1) \,\dot{\sqcup}\, \dot{\delta}(e_2).$$

- Rules on when the remaining operators can interact with strong-B-equivalences.
- Rules on when the remaining operators can interact with weak-B-equivalences.
- Best-effort derivation rules to derive the keys a query result can have.
- Best-effort derivation rules to determine whether a query result can have duplicates.

A Detailed Example

Consider the query

$$e = \dot{\sigma}_{type=\mathtt{food}}(Bought \,\dot{\bowtie}\, Product)$$

A Detailed Example

Consider the query

$$e = \dot{\sigma}_{type=\texttt{food}}(Bought \dot{\bowtie} Product)$$

First, we push down selections in e and obtain:

$$e \doteq Bought \dot{\bowtie}_{type=\texttt{food}} Product.$$

A Detailed Example

Consider the query

$$e = \dot{\sigma}_{type=\mathtt{food}}(Bought \dot{\bowtie} Product)$$

First, we push down selections in e and obtain:

$$e \doteq Bought \dot{\bowtie}_{type=\mathtt{food}} Product.$$

As *pname* is a key of *Product* and *Product* is a set relation, we have:

$$Bought \dot{\bowtie}_{type=\mathtt{food}} Product \doteq_{\{cname,pname\}} Bought \dot{\ltimes}_{type=\mathtt{food}} Product.$$

A Detailed Example

Consider the query

$$e = \dot{\sigma}_{type=\texttt{food}}(Bought \mathbin{\dot{\bowtie}} Product)$$

First, we push down selections in e and obtain:

$$e \doteq Bought \mathbin{\dot{\bowtie}}_{type=\texttt{food}} Product.$$

As *pname* is a key of *Product* and *Product* is a set relation, we have:

$$Bought \mathbin{\dot{\bowtie}}_{type=\texttt{food}} Product \doteq_{\{cname,pname\}} Bought \mathbin{\dot{\ltimes}}_{type=\texttt{food}} Product.$$

Let $f = \{type := \texttt{food}\}$. We have

$$Bought \mathbin{\dot{\bowtie}}_{type=\texttt{food}} Product \doteq_{\{cname,pname,type\}} i_f(Bought \mathbin{\dot{\ltimes}}_{type=\texttt{food}} Product).$$

Toward Semi-Join Rewritings in Practice

Current experience

- Limited rewriting happens in most databases.
  E.g., A single semi-join in an edge-reachability query.
- In distributed settings: semi-joins are used to reduce communication.

Toward Semi-Join Rewritings in Practice

Current experience

- Limited rewriting happens in most databases.
  E.g., A single semi-join in an edge-reachability query.
- In distributed settings: semi-joins are used to reduce communication.

Practical limitations

- Many systems have a low *upper bound* on the cost of optimizing query evaluation.
- Integration with other query optimization steps.
  E.g., index selection, selection of join algorithms, ...

Toward Semi-Join Rewritings in Practice

Current experience

- Limited rewriting happens in most databases.
  E.g., A single semi-join in an edge-reachability query.
- In distributed settings: semi-joins are used to reduce communication.

Practical limitations

- Many systems have a low *upper bound* on the cost of optimizing query evaluation.
- Integration with other query optimization steps.
  E.g., index selection, selection of join algorithms, ...

Future direction

Many applications have static (up to parameters) queries:
These could be partially pre-compiled (heavier optimization path).

Conclusion

We provide a formal framework for optimizing SQL queries with

- a main focus on rewriting joins to semi-joins;
- that considers multiset semantics;
- that uses conditions and selections; and
- that incorporates key and duplicate information.