Fully Generic Queries:

Open Problems and Some Partial Answers

Jan Van den Bussche
(Hasselt University, Belgium)

joint work with Dimitri Surinx, Jonni Virtema

What is a Query?

An SQL select statement?

A Python program?

A Google query?

Theory of database queries [Chandra, Harel]

Fix some input database schema S_{in}

Fix some output database schema S_{out}

A query is a mapping from instances of S_{in} to instances of S_{out}

- SQL: input relational database, output relation
- Python: input objects containing data, output object
- Google: input documents, output documents

Computability, genericity

Two requirements on mapping Q:

1. Computable

2. Generic: treat atomic data entries as atomic!

Consider query Q on relations $\mathtt{R}(\mathtt{A},\mathtt{B})$ and $\mathtt{S}(\mathtt{C})$:

```
select A
from R, S
where B < C
```

Permute the entries in the relations, preserving order
$\Rightarrow$ query answer is preserved

Commutative diagram

Q: `select A from R,S where B<C`

$$
\begin{tikzcd}
\begin{array}{cc} A & B \\ \hline 1 & 7 \\ 2 & 3 \\ 2 & 6 \end{array} \quad \begin{array}{c} C \\ \hline 4 \\ 5 \end{array} \arrow[r, "\times 2"] \arrow[d, "Q"'] & \begin{array}{cc} A & B \\ \hline 2 & 14 \\ 4 & 6 \\ 4 & 12 \end{array} \quad \begin{array}{c} C \\ \hline 8 \\ 10 \end{array} \arrow[d, "Q"] \\
\begin{array}{c} A \\ \hline 2 \end{array} \arrow[r, "\times 2"] & \begin{array}{c} A \\ \hline 4 \end{array}
\end{tikzcd}
$$

Formal definition of genericity

Let **dom** be the universe of atomic data values

Query Q is "generic" if

$$Q(f(D)) = f(Q(D))$$

for every D and every permutation f of **dom**

- All reasonable queries are generic.
- Some queries are even more...

Full genericity [Beeri, Milo, Ta-Shma]

- Query Q is "generic" if

$$Q(f(D)) = f(Q(D))$$

  for every D and every permutation f of **dom**

- Query Q is "**fully** generic" if

$$Q(f(D)) = f(Q(D))$$

  for every D and every **function** f : **dom** $\rightarrow$ **dom**

$\Rightarrow$ f may map distinct values to the same

Cartesian product is fully generic

Q: $R \times S$

A	B
1	2
3	4
5	6

C
7
8

$\xrightarrow{x \,\mathsf{div}\, 2}$

A	B
0	1
1	2
2	3

C
3
4

$Q \big\downarrow$ $\qquad$ $\big\downarrow Q$

A	B	C
1	2	7
1	2	8
3	4	7
3	4	8
5	6	7
5	6	8

$\xrightarrow{x \,\mathsf{div}\, 2}$

A	B	C
0	1	3
0	1	4
1	2	3
1	2	4
2	3	3
2	3	4

Also fully generic: projection, union

Intersection **not** fully generic

Q: $R \cap S$

$$
\begin{tikzcd}
\boxed{1} \quad \boxed{2} \arrow[r, "x \mapsto 1"] \arrow[d, "Q"'] & \boxed{1} \quad \boxed{1} \arrow[d, "Q"] \\
\varnothing \arrow[r, "\textbf{not}"] & \boxed{1}
\end{tikzcd}
$$

Neither fully generic: difference, selection

Intuition for fully generic:

- Output can be computed without comparing values

Application: Provenance tracking

Relations $R(A, B)$, $S(C, D)$ with provenance tokens

Query Q: $\pi_{A,D}(\sigma_{B=C}(R \times S))$

$$
\begin{array}{ccc}
A & B & \\
\hline
1 & 2 & a \\
1 & 4 & b
\end{array}
\qquad
\begin{array}{ccc}
C & D & \\
\hline
2 & 3 & c \\
4 & 3 & d
\end{array}
\Rightarrow
\begin{array}{ccccc}
A & B & C & D & \\
\hline
1 & 2 & 2 & 3 & (a, c) \\
1 & 2 & 4 & 3 & (a, d) \\
1 & 4 & 2 & 3 & (b, c) \\
1 & 4 & 4 & 3 & (b, d)
\end{array}
$$

$$
\Rightarrow
\begin{array}{ccccc}
A & B & C & D & \\
\hline
1 & 2 & 2 & 3 & (a, c) \\
1 & 4 & 4 & 3 & (b, d)
\end{array}
$$

$$
\Rightarrow
\begin{array}{ccc}
A & C & \\
\hline
1 & 3 & (a, c) + (b, d)
\end{array}
$$

No need to compare provenance tokens

Complex objects

"Flat" relational model: only fully generic queries are **unions** of **projections** of **cartesian products**.

Complex objects:

d atomic data values

[ ] tuple constructor

$\{\ \}$ set constructor

- arbitrary combinations allowed
- typed

E.g. type $\{[\{d\},\{d\}]\}$

Complex objects example

Store, for each concept, the set of synonyms in French, and in English

$\Rightarrow$ object of type $\{[\underset{\substack{\downarrow \\ \text{French}}}{\{d\}}, \underset{\substack{\downarrow \\ \text{English}}}{\{d\}}]\}$

Store, for each concept, and each available language, the set of synonyms

$\Rightarrow$ object of type $\{\{[\underset{\substack{\downarrow \\ \text{language}}}{d}, \underset{\substack{\downarrow \\ \text{synonyms}}}{\{d\}}]\}\}$

OO programming languages, collection types, Spark, JSON

Fully generic complex-object queries

Nested relational algebra without equality selection:

identity: $o \mapsto o$
projection: $[o_1, \ldots, o_k] \mapsto o_i$
singleton: $o \mapsto \{o\}$
union: $[o_1, o_2] \mapsto o_1 \cup o_2$

unit: $o \mapsto [\,]$
empty: $o \mapsto \varnothing$
flatten: $\{o_1, \ldots, o_n\} \mapsto o_1 \cup \cdots \cup o_n$
cart.prod: $[o_1, o_2] \mapsto o_1 \times o_2$

emptiness test
composition of queries
map: $\{o_1, \ldots, o_n\} \mapsto \{Q(o_1), \ldots, Q(o_n)\}$
tupling: $o \mapsto [Q_1(o), \ldots, Q_k(o)]$

We denote this language by $\mathcal{L}$

One-each [Beeri, Milo, Ta-Shma]

Intriguing operator, fully generic

For any type τ, define one-each : $\{\{\tau\}\} \to \{\{\tau\}\}$:

$$\{s_1, \ldots, s_n\} \mapsto \{s'_1 \cup \cdots \cup s'_n \mid \emptyset \neq s'_i \subseteq s_i \text{ for } i = 1, \ldots, n\}$$

Can express powerset operator:

$$2^s = \text{one-each}(\{s\}) \cup \{\emptyset\}$$

Open question: Can one-each be expressed in $\mathcal{L}$ + powerset?

Open question: Can every fully generic query be expressed in $\mathcal{L}$ + one-each?

The equivalence problem

Equivalence problem:

Input: Query expressions E_1, E_2

Decide: Do E_1 and E_2 express the same query?

Fundamental problem, reasoning,
automated query optimization

Undecidable for relational algebra, or nested relational algebra

Decidable for nested relational algebra, with **atomic** equality selection, but without emptiness test

Open question: What about $\mathcal{L}$?

Computability and definability

Open question: Is every fully generic query computable?

Open question: Is the definability problem decidable?

Input: Two objects A and B

Decide: Does there exist fully generic Q such that $Q(A) = B$?

Lacking answers to all these questions (some partial answers in proceedings paper)

We can at least formalize the intuition: *fully generic* $=$ *computable without comparisons, not sensitive to duplicates*

Formalizing computability of queries

Classical computability works with strings over finite alphabet

Bijection *enc* : **dom** $\to \{0,1\}^*$

Encode objects as strings over $\{0,1\}$ and puncutation symbols (comma, square brackets, set brackets)

Query Q is "computable under *enc*" if there exists Turing machine M:

Input: An encoding of some instance D

Output: An encoding of $Q(D)$

If Q is generic, this is independent of chosen *enc*

Domain Turing machines

Hull and Su proposed **domain** Turing machine:

- works directly over strings with **dom**-elements
- copy current symbol in register
- compare register value with current symbol
- write register value to current tape cell

Every computable, generic query is computable by a domain Turing machine

Oblivious domain Turing machines

- works directly over strings with **dom**-elements
- copy current symbol in register

no comparing register value with current symbol

- write register value to current tape cell

$\Rightarrow$ computing without comparing values: fully generic

Theorem: Every computable, fully generic query is computable by a oblivious domain Turing machine

Conclusion

Fully generic queries

Fascinating class of queries

Many open questions; poorly understood

Can be processed without comparing values;
output not sensitive to duplicates in input

Allows fast reorganisation of data

Linear in output size?

What about bag data model?