

# 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!

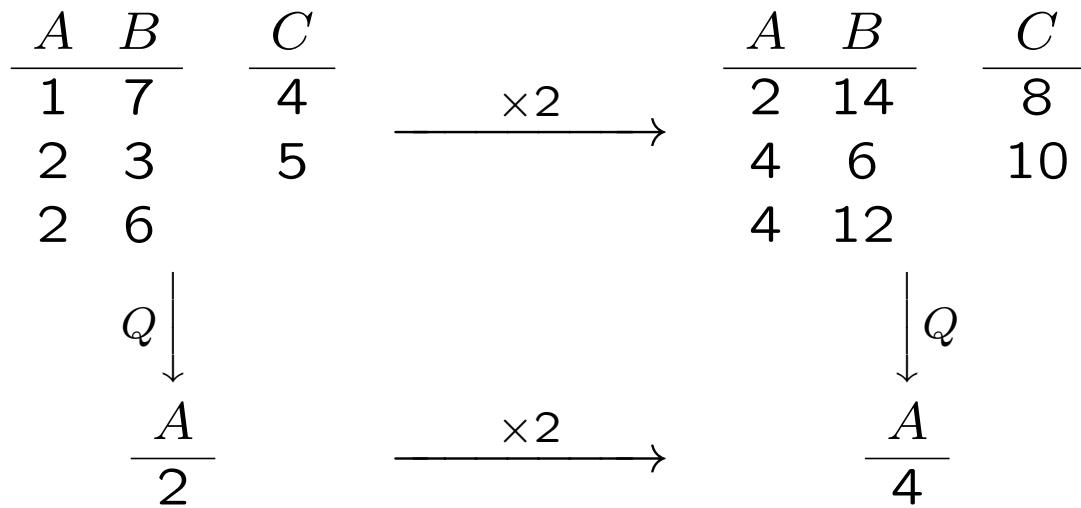
Consider query  $Q$  on relations  $R(A,B)$  and  $S(C)$ :

```
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  $B < C$



## 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

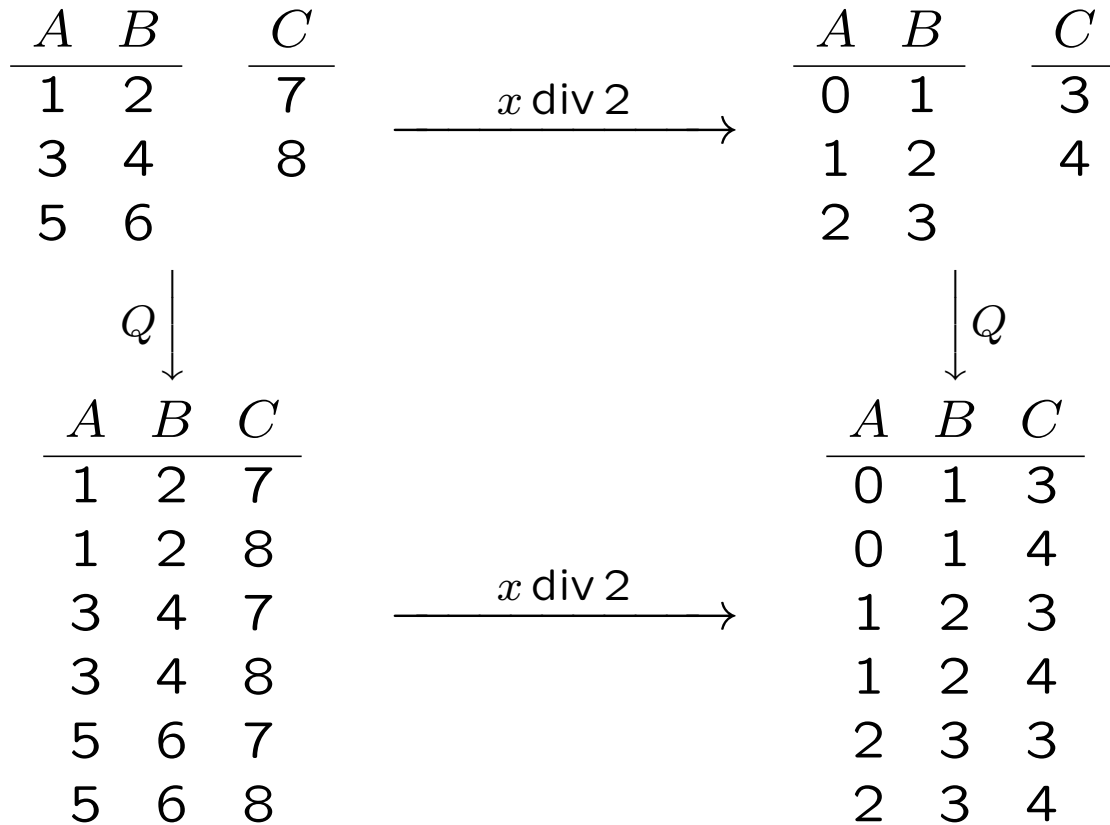
$$Q(f(D)) = f(Q(D))$$

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

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

# Cartesian product is fully generic

$Q: R \times S$

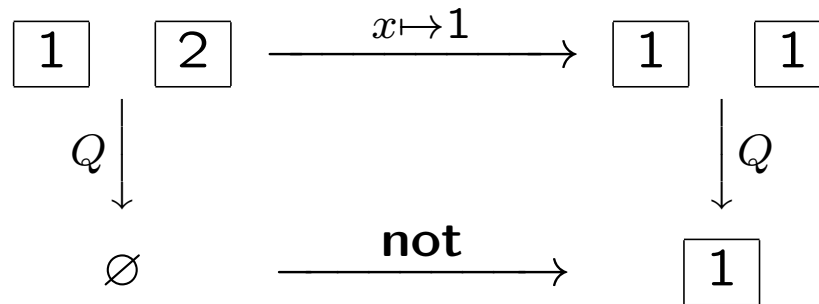


Also fully generic: projection, union



# Intersection **not** fully generic

$Q: R \cap S$



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

$A$	$B$		$C$	$D$	$\Rightarrow$	$A$	$B$	$C$	$D$	
1	2	$a$	2	3	$c$	1	2	2	3	$(a, c)$
1	4	$b$	4	3	$d$	1	2	4	3	$(a, d)$
						1	4	2	3	$(b, c)$
						1	4	4	3	$(b, d)$
					$\Rightarrow$	$A$	$B$	$C$	$D$	
						1	2	2	3	$(a, c)$
						1	4	4	3	$(b, d)$
					$\Rightarrow$	$A$	$C$			
						1	3	$(a, c) + (b, d)$		

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

⇒ object of type  $\{[\{d\}, \{d\}]\}$

                  ↓        ↓

                  French    English

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

⇒ object of type  $\{[d, \{d\}]\}$

                  ↓        ↓

                  language    synonyms

OO programming languages, collection types, Spark, JSON

# Fully generic complex-object queries

**Nested relational algebra** without equality selection:

identity:  $o \mapsto o$

projection:  $[o_1, \dots, o_k] \mapsto o_i$

singleton:  $o \mapsto \{o\}$

union:  $[o_1, o_2] \mapsto o_1 \cup o_2$

unit:  $o \mapsto []$

empty:  $o \mapsto \emptyset$

flatten:  $\{o_1, \dots, o_n\} \mapsto o_1 \cup \dots \cup o_n$

cart.prod:  $[o_1, o_2] \mapsto o_1 \times o_2$

emptiness test

composition of queries

map:  $\{o_1, \dots, o_n\} \mapsto \{Q(o_1), \dots, Q(o_n)\}$

tupling:  $o \mapsto [Q_1(o), \dots, Q_k(o)]$

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

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

Intriguing operator, fully generic

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

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

Can express powerset operator:

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

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

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

# 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  $enc : \mathbf{dom} \rightarrow \{0, 1\}^*$

Encode objects as strings over  $\{0, 1\}$  and punctuation 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

⇒ 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?