A Crash Course

in Database Queries

and how to treat queries as data

Jan Van den Bussche
Hasselt University

joint work with Stijn Vansummeren, Dirk Van Gucht

Query languages are programming languages!

[Atkinson–Buneman–Cardelli–Maier–Ohori–Sheard–Stemple
–Stonebraker–Tannen]

$\Rightarrow$ Database queries. . .

- are programs
- can crash
- can be ill-typed
- are polymorphic
- can be treated as data (metadata, reflection)

Motivation

Lowell self-assessment:

> "We recommend that database researchers increase their focus on the integration of text, data, code, and streams."

Asilomar Report:

> "Ever more complex application environments have increased the need to integrate programs and data."

What are the basic theoretical questions concerning flexible operation of queries "out of the box"? Of treating queries as data?

How to apply/adapt programming-language ideas to query languages?

Query languages

Relational algebra

Nested relational calculus

XQuery

Relational algebra: syntax

$$
\begin{array}{rcll}
e & ::= & x & \text{(relation variable)} \\
 & | & r & \text{(constant relation)} \\
 & | & e \cup e & \\
 & | & e - e & \\
 & | & e \times e & \\
 & | & \sigma_{A=B}(e) & \\
 & | & \pi_{A,\ldots,B}(e) & \\
 & | & \rho_{A/B}(e) &
\end{array}
$$

Example expression:

$$x - (\pi_A(x) \times \rho_{C/B}(y))$$

Heterogeneous relations

A relation is a finite set of tuples

A tuple is a mapping t from some relation scheme to $\mathbb{V}$

- $\mathbb{V}$: universe of atomic values
- relation scheme: finite set of attributes
- call relation scheme of t, the <u>type</u> of t

$$\boxed{\begin{array}{l}(A\colon 1, B\colon 2)\\(A\colon 3, C\colon 4)\\(D\colon 4)\end{array}}$$

Relational algebra: semantics

Expression $e(x, \ldots, y)$ can be applied to relations $r, \ldots, s$

Operational semantics: rewrite ground expression $e' = e(r, \ldots, s)$ until you end up with a relation r'

Rewriting $e' \to r'$ defined by inference rules

Inference rule for union

if $e_1 \rightarrow r_1$
and $e_2 \rightarrow r_2$
then $e_1 \cup e_2 \rightarrow r_1 \cup r_2$

Written like fraction:

$$\frac{e_1 \rightarrow r_1 \qquad e_2 \rightarrow r_2}{e_1 \cup e_2 \rightarrow r_1 \cup r_2}$$

Difference:

$$\frac{e_1 \rightarrow r_1 \qquad e_2 \rightarrow r_2}{e_1 - e_2 \rightarrow r_1 \setminus r_2}$$

Cartesian product

$$\frac{e_1 \longrightarrow r_1 \qquad e_2 \longrightarrow r_2 \qquad \forall t_1 \in r_1 : \forall t_2 \in r_2 : \mathsf{type}(t_1) \cap \mathsf{type}(t_2) = \varnothing}{e_1 \times e_2 \longrightarrow \{t_1 \cup t_2 \mid t_1 \in r_1 \ \& \ t_2 \in r_2\}}$$

Selection, projection, renaming

$$\frac{e \to r' \qquad \forall t \in r' : A, B \in \mathsf{type}(t)}{\sigma_{A=B}(e) \to \{t \in r' \mid t(A) = t(B)\}}$$

$$\frac{e \to r' \qquad \forall t \in r' : A, \ldots, B \in \mathsf{type}(t)}{\pi_{A,\ldots,B}(e) \to \{\pi_{A,\ldots,B}(t) \mid t \in r'\}}$$

$$\frac{e \to r' \qquad \forall t \in r' : A \in \mathsf{type}(t) \ \& \ B \notin \mathsf{type}(t)}{\rho_{A/B}(e) \to \{\rho_{A/B}(t) \mid t \in r'\}}$$

Example derivation

Evaluate $x - (\pi_A(x) \times \rho_{C/B}(y))$

$$\dfrac{r \to r \qquad \dfrac{\dfrac{r \to \boxed{\begin{array}{c}(A\colon 1, B\colon 2)\\(A\colon 1, B\colon 3)\\(A\colon 2, B\colon 4)\\(A\colon 2, B\colon 5)\end{array}}}{\pi_A(r) \to \boxed{\begin{array}{c}(A\colon 1)\\(A\colon 2)\end{array}}} \qquad \dfrac{s \to \boxed{\begin{array}{c}(C\colon 2)\\(C\colon 3)\\(C\colon 4)\end{array}}}{\rho_{C/B}(s) \to \boxed{\begin{array}{c}(B\colon 2)\\(B\colon 3)\\(B\colon 4)\end{array}}}}{\pi_A(r) \times \rho_{C/B}(s) \to \boxed{\begin{array}{c}(A\colon 1, B\colon 2)\\(A\colon 1, B\colon 3)\\(A\colon 1, B\colon 4)\\(A\colon 2, B\colon 2)\\(A\colon 2, B\colon 3)\\(A\colon 2, B\colon 4)\end{array}}}}{r - (\pi_A(r) \times \rho_{C/B}(s)) \to \boxed{(A\colon 2, B\colon 5)}}$$

Nested relations (complex objects)

A complex object is:

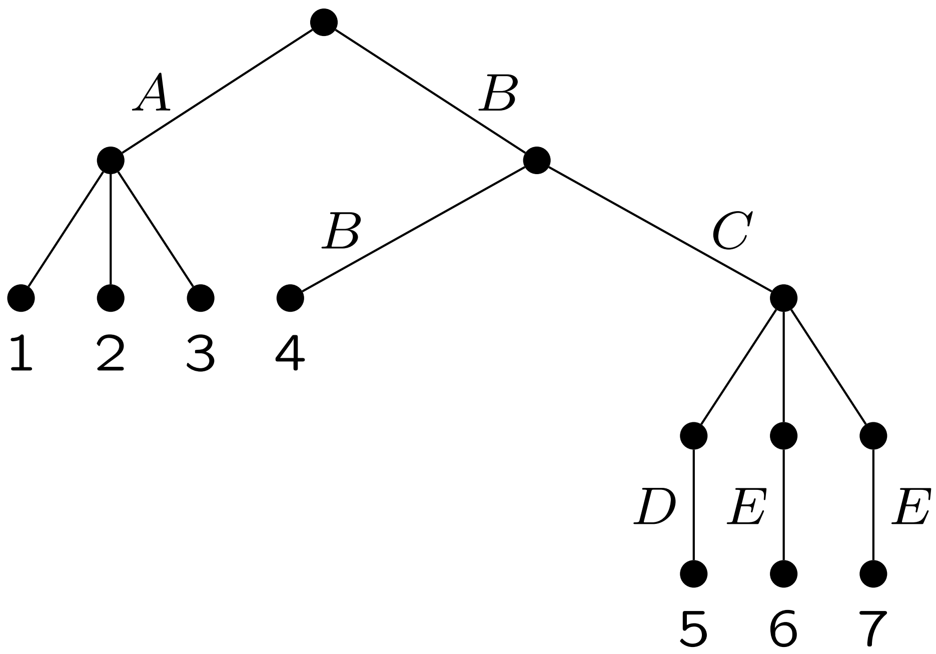
- an atomic value

- a tuple of complex objects

- a finite set of complex objects

$$(A\colon \boxed{\begin{array}{c} 1 \\ 2 \\ 3 \end{array}}, B\colon (B\colon 4, C\colon \boxed{\begin{array}{c} (D\colon 5) \\ (E\colon 6) \\ (E\colon 7) \end{array}}))$$

Complex object tree



$$\left(A\colon \boxed{\begin{array}{c}1\\2\\3\end{array}}, B\colon \left(B\colon 4, C\colon \boxed{\begin{array}{c}(D\colon 5)\\(E\colon 6)\\(E\colon 7)\end{array}}\right)\right)$$

Nested relational calculus [BNTW]

The relational algebra of complex objects

Syntax:

$$
\begin{array}{rcll}
e & ::= & x & \text{(variable)} \\
 & | & r & \text{(constant object)} \\
 & | & \varnothing & \\
 & | & \{e\} & \\
 & | & e \cup e & \\
 & | & \bigcup e & \\
 & | & e.A & \\
 & | & (A\colon e, \ldots, B\colon e) & \\
 & | & \text{for } z \in e \text{ return } e & \\
 & | & \text{if } e \text{ eq } e \text{ then } e \text{ else } e & \\
 & | & \text{if } e = e \text{ then } e \text{ else } e &
\end{array}
$$

Example NRC expression

$$
\begin{array}{l}
\bigcup \text{for } u \in x \text{ return} \\
\quad\quad \bigcup \text{for } v \in y \text{ return} \\
\quad\quad\quad\quad \text{if } u.B \text{ eq } v.B \\
\quad\quad\quad\quad\quad\quad \text{then } \{(A\colon u.A, B\colon u.B, C\colon v.C)\} \\
\quad\quad\quad\quad\quad\quad \text{else } \emptyset
\end{array}
$$

Operational semantics of for-loop

$$\frac{e_1 \to r \qquad r \text{ is a set} \qquad \forall t \in r : e_2(t) \to s_t}{\text{for } z \in e_1 \text{ return } e_2(z) \to \{s_t \mid t \in r\}}$$

Equality test

$$\frac{e_1 \to r_1 \qquad e_2 \to r_2 \qquad e_3 \to r_3 \qquad r_1 = r_2}{\text{if } e_1 = e_2 \text{ then } e_3 \text{ else } e_4 \to r_3}$$

$$\frac{e_1 \to r_1 \qquad e_2 \to r_2 \qquad e_4 \to r_4 \qquad r_1 \neq r_2}{\text{if } e_1 = e_2 \text{ then } e_3 \text{ else } e_4 \to r_4}$$

e_1 eq e_2: similar, but r_1 and r_2 must be atomic values

<u>Positive</u> NRC: has only e_1 eq e_2

Queries can crash

$\pi_A(x)$ crashes on $r = \boxed{\begin{array}{l}(A\colon 1, B\colon 2)\\(B\colon 3)\end{array}}$

$x \times y$ crashes on $r = \boxed{\begin{array}{l}(A\colon 1)\\(B\colon 2)\end{array}}$ and $s = \boxed{(B\colon 3, C\colon 4)}$

Similar for NRC

"Well-defined" = "does not crash"

Cannot expect an expression to be well-defined on all inputs

$\Rightarrow$ consider types

The well-definedness problem

(Relational algebra version)

Recall: type = relation scheme = finite set of attributes

A relation r has type R if all its tuples do

- notation $r : R$

The well-definedness problem:

Input: expression $e(x, \ldots, y)$ and types $R, \ldots, S$

Decide: is $e(r, \ldots, s)$ well-defined on all inputs $r : R$, ..., $s : S$?

Some immediate observations

Well-definedness is undecidable for relational algebra:

- take $R = \{A, B\}$
- for well-defined e:

$$e(x) \text{ satisfiable} \quad \Leftrightarrow \quad \pi_A \pi_B(x \times \pi_\emptyset(e)) \text{ ill-defined}$$

Decidable for <u>positive</u> relational algebra (without difference):

- just keep track of possible types that can occur
- monotonicity
- can add $\sigma_{A \neq B}$

The NRC case

Types:

- atom

- $(A\colon R, \ldots, B\colon S)$

- $\{T\}$

Well-definedness still undecidable for full NRC

Well-definedness for positive NRC

Still decidable

- small model property for ill-definedness
- monotonicity
- coNEXPTIME-hard (satisfiability, [Koch])

Singleton extraction

$$\frac{e \to \{t\}}{\text{extract}(e) \to t}$$

OQL, XQuery, but also SQL:

`select ..., (`Q`), ... from ... where ...`

or

`select ... from ... where `A` = (`Q`)`

crashes unless subquery returns a singleton ("scalar subquery")

Well-definedness for positive NRC + extract

Is undecidable

- extract($\{e_1, e_2\}$) well-defined iff e_1, e_2 equivalent
- equivalence of positive NRC is undecidable!

(Satisfiability still decidable.)

Decidable for lists, bags (see also XQuery)

XQuery (w/o recursion)

Value = list of atoms and tree nodes

- underlying store of XML trees (node-labeled by atoms)

Language:

$$
\begin{array}{rcll}
e & ::= & x & \text{(variable)} \\
  & | & a & \text{(constant atom)} \\
  & | & () & \\
  & | & \text{for } z \in e \text{ return } e & \\
  & | & \text{if } e \text{ then } e \text{ else } e & \\
  & | & \text{let } z := e \text{ in } e & \\
  & | & f(e, \ldots, e) & \text{(operators)}
\end{array}
$$

XQuery operators

Operators: Element construction, list functions, axes, tests, . . .

– can crash!

e.g. element$\{e_1\}\{e_2\} \Rightarrow e_1$ must be singleton

XQuery well-definedness

Types: bounded-depth regular expression types

– XML Schema (extended DTD) w/o recursion

For <u>well-behaved</u> (generic, monotonic, local) operators, XQuery well-definedness is <u>decidable</u>

Caveat: automatic coercions

– atomization: tree $\rightarrow$ atom

– difference between NRC(=) and NRC(eq) is blurred

– undecidability

Semantic type-checking

Input: expression $e(x, \ldots, y)$, well-defined under types $R, \ldots, S$; additional type T

Decide: is $e(r, \ldots, s)$ always of type T, for all inputs $r : R$, ..., $s : S$?

- RA, NRC: same story as well-definedness
- positive NRC + extract: still decidable!
- XQuery: undecidable!

Static type checking

Since input types are known, can try to derive statically:

- can expression crash?

- if not, what is output type?

$\Rightarrow$ Inference rules that derive $\Gamma \vdash e : T$

– Γ: the given input types

- relational algebra

- NRC

- XQuery [XQuery formal semantics; Ghelli et al., JFP 2006]

Static type system of relational algebra

$$\frac{\Gamma \vdash e_1 : T \qquad \Gamma \vdash e_2 : T}{\Gamma \vdash e_1 \cup e_2 : T} \qquad \frac{\Gamma \vdash e_1 : T \qquad \Gamma \vdash e_2 : T}{\Gamma \vdash e_1 - e_2 : T}$$

$$\frac{\Gamma \vdash e_1 : R \qquad \Gamma \vdash e_2 : S \qquad R \cap S = \varnothing}{\Gamma \vdash e_1 \times e_2 : R \cup S} \qquad \frac{\Gamma \vdash e : T \qquad A, B \in T}{\Gamma \vdash \sigma_{A=B}(e) : T}$$

$$\frac{\Gamma \vdash e : T \qquad A, \ldots, B \in T}{\Gamma \vdash \pi_{A,\ldots,B}(e) : \{A, \ldots, B\}} \qquad \frac{\Gamma \vdash e : T \qquad A \in T \qquad B \notin T}{\Gamma \vdash \rho_{A/B}(e) : (R \setminus \{A\}) \cup \{B\}}$$

Soundness and completeness of type systems

If e is well-typed ($e : T$ can be derived) then e is well-defined, and output type is always T

Not vice versa: type checking is only sound, not complete

Nevertheless we have <u>expressive completeness</u>:

- a well-defined expression with known output type can always be equivalently rewritten into a well-typed expression

[Vansummeren]

Expressive completeness of static type checking

- given types $R, \ldots, S$
- given type T
- given expression $e(x, \ldots, y)$
  - e well-defined under $R, \ldots, S$
  - e's output type is always T

Then there always exists e' equivalent to e on all inputs $r : R, \ldots, s : S$, and

$$\boxed{x : R, \ldots, y : S \vdash e' : T}$$

Holds for RA, NRC — for XQuery?

Simple example of expressive completeness

Consider $x : \{A, B\}$ and $y : \{A, C\}$

Then

$$\pi_A(x \cup y)$$

is equivalent to

$$\pi_A(x) \cup \pi_A(y)$$

Less trivial for queries involving difference

– Encode, in RA/NRC, heterogeneous relations/objects by homogeneous ones.

Note

Expressive completeness is immediate for well-typed languages that are Turing-complete

– Java, Haskell, . . .

– Simply-typed lambda calculus: not expressive complete

Polymorphism

$\widehat{\pi}_A$: project out attribute A

$\bowtie$: natural join

Typing rules:

$$\frac{e : R \qquad A \in R}{\widehat{\pi}_A(e) : R \setminus \{A\}} \qquad\qquad \frac{e_1 : R_1 \qquad e_2 : R_2}{e_1 \bowtie e_2 : R_1 \cup R_2}$$

For any given input types, $\widehat{\pi}$ and $\bowtie$ can be expressed using the other RA operators

But not by one expression that works for all possible input types!

The ultimate polymorphic, typed, query language?

Type inference

Input: expression e

Output: all Γ and T for which $\Gamma \vdash e : T$

Output is usually infinite; need some simple kind of finite representation

Output can be empty as well (untypeable expression, e.g., $\pi_A \pi_B(x)$)

$\Rightarrow$ type formulas

Polytypes with kinding [Ohori–Buneman]

Polytype = type with type variables (ML, Hindley–Milner)

NRC:

for $u \in x$ return
  for $v \in y$ return
    $(C\colon u.A, D\colon v.B)$

Type formula with type variables and kinding

$$\left.\begin{array}{l} x : \{\alpha\} \\ y : \{\beta\} \\ \mathsf{kind}(\alpha) = (A\colon \gamma) \\ \mathsf{kind}(\beta) = (B\colon \delta) \end{array}\right. \mapsto (C\colon \gamma, D\colon \delta)$$

Insufficient to represent type inference of $\widehat{\pi}$ or ρ

Row variables [Rémy]

$$\widehat{\pi}_A(x) \cup y$$

Type formula with row variables and forbidden attributes:

$$\begin{array}{l} x : \{A\} \cup \alpha \\ y : \alpha \\ \text{forbidden}(\alpha) = \{A\} \end{array} \mapsto \alpha$$

Insufficient to represent type inference of $\times$

Type inference for full relational algebra

1. Use multiple row variables, stand for disjoint types

2. Generalize required, forbidden attributes to boolean constraints

$$e = \sigma_{B=C}(\rho_{A/B}(x) \times y)$$

$$\begin{array}{l} x : \alpha \\ y : \beta \\ A : x \\ B : \neg x \wedge \neg y \\ C : x \vee y \end{array} \quad \mapsto \quad \begin{array}{l} e : \alpha \cup \beta \\ A : y \\ B : \text{true} \\ C : \text{true} \end{array}$$

Type formulas can become exponentially long, but <u>typeability</u> is in NP (complete)

General approach to type inference

1. Universe of all types, with appropriate constraints and operations on types, forms a logical model $\mathcal{M}$

2. Formulate quantifier-free formula over type variables, stating constraints on input types of e, for e to be well-typed

   - Type formula!

3. Existential theory of $\mathcal{M}$ is decidable (typeability)

Can play this game for full NRC with $\widehat{\pi}$, ρ, $\bowtie$, $\times$, ...

Complexity for NRC does not become worse than for relational algebra (NP-complete)

Type reflection

java.lang.reflection package:

- inspect type (class) of an object
- type information becomes data!

Schema querying ('90s)

Semistructured data, XML: distinction between type information and data is blurred

- can validate XML Schema using (recursive) XQuery

Downside: is XPath expression `x/a=5` false because x has no a-child? Or because x has an a-child, but it is not 5?

Example: transposition of a relation [GL,WR]

$$
\begin{array}{ccc} A & B & C \\ \hline \multicolumn{1}{|c}{1} & d & \multicolumn{1}{c|}{e} \\ \multicolumn{1}{|c}{2} & f & \multicolumn{1}{c|}{g} \\ \hline \end{array}
\quad \leftrightarrow \quad
\begin{array}{ccc} A & 1 & 2 \\ \hline \multicolumn{1}{|c}{B} & d & \multicolumn{1}{c|}{f} \\ \multicolumn{1}{|c}{C} & e & \multicolumn{1}{c|}{g} \\ \hline \end{array}
$$

$\Rightarrow$ Attributes as data values

Expressions as data

E.g., workload log:

user	datetime	query
john	20070612T1030	select . . . from . . . where . . .
mary	20070611T2316	. . .
	⋮	

Integration of program logic and data

- System catalog: VIEWS table
- QUEL as a data type
- Oracle EXPRESSION type
- workload monitoring
- publish-subscribe
- workflow management
- software engineering

Querying a database containing queries

- Hotspots: which subqueries are often used?
  - syntactic
- Semantic: which queries return no answer?
  - semantic
- View maintenance: how do the query answers change under this update?
  - syntactic & semantic

Two approaches: (i) decomposition; (ii) ADT

Decomposition-based approach

Use standard query language for syntactic manipulations

$\Rightarrow$ Stored expressions cannot be represented as atomic values

– must be decomposed

Many ways to do this:

- XML: syntax tree
  - XQueryX
- Relational: decomposition

MetaSQL

SQL/XML

- queries are stored in XML columns

Add EVAL function to SQL

- analogy with Lisp, Scheme

E.g. workloads: $\text{Log}(\text{user}, \text{datetime}, \text{query})$

```
select query, A, B
from Log
where (A,B) in EVAL(rewrite(query,update)) MINUS EVAL(query)
```

Similarly can add EVAL to RA, or XQuery

Does EVAL add power?

Data complexity of EVAL = evaluation complexity of relational algebra evaluation:

Input: database D and expression e

Output: $e(D)$

PSPACE-complete $\gg$ LOGSPACE (plain relational algebra)

But what about standard generic queries?

- expressions not in input

- dynamic generation and evaluation of expressions only as auxiliary querying tool

Transitive closure in XQuery + EVAL

Table $R(A,B)$ in XML: list D of $R(A,B)$-elements

TC(D) = EVAL(<u>construct</u>(D)):

construct(D) $= E_1, \ldots, E_n$ with $n =$ `count`$(D//T)$

with E_j:

for t_1 in D//T, . . . , t_j in D//T return
if every z in ((t_1/B=t_2/A),. . . ,(t_{j-1}/B=t_j/A))
satisfies z=fn:true() then
element(T){t_1/A,t_j/B} else ()

In relational algebra, on databases w/o stored expressions,
EVAL = for-loops

Type-safe reflection

EVAL: fragile, can crash easily

Idea: two-level type system [MetaML], e.g.:

$$(A\colon \mathsf{atom}, B\colon \mathsf{atom}, C\colon \langle\{(D\colon \mathsf{atom}, E\colon \mathsf{atom})\}\rangle)$$

$\Rightarrow$ EVAL can be typed

ADT approach: also provide typed repertoire of syntactic manipulation operators

e.g. substitute all occurrences of relation variable $x : T$ by the expression $e : T$

Less powerful $\Rightarrow$ add polymorphism?

Conclusions

Flexible operation of queries "out of the box"

- Well-definedness (better algorithms?)
- Expressive completeness of type systems
- Polymorphism (design of query languages?)
- Meta-querying: querying queries (design of languages?)

Integration of programs (queries) and data