Towards a theory of search queries

Jan Van den Bussche
(Hasselt University)

Joint work with
George Fletcher, Dirk Van Gucht,
Stijn Vansummeren

ACM TODS (November 2010)
Outline

1. Theory of database queries
2. Relational algebra
3. Semijoin algebra
4. Search queries
5. Dataspaces
6. Structured querying versus searching
7. Research problems
Computational problems

• Classically, any computational problem is a function (mapping) from inputs to outputs

• E.g., route planning:
  – Input: a map (graph), source, target
  – Output: shortest route in graph from source to target

• Deal with nondeterminism
Database queries

• A **query** is a function from databases to databases

• E.g., Employee query
  – Input: history of employee hirings
  – Output: list of all employees who have been hired at least twice

• Also route planning!
Relational algebra

• Language in which queries over relational databases can be expressed

• Every expression denotes a query
  – compare arithmetic: $\text{avg}(x,y) = (x+y)/2$

• Expression is a combination of operators
  – union, intersection, difference
  – cartesian product (join)
  – selection
  – projection
  – renaming
Employee query

relation History(emp_id, hire_date)

\[ \pi_{H1.emp_id} \sigma_{H1.emp_id=H2.emp_id \text{ and } H1.hire_date\neq H2.hire_date} (\rho_{H1}(History) \times \rho_{H2}(History)) \]

equivalently:

\[ \pi_{H1.emp_id} (\rho_{H1}(History) \bowtie \rho_{H2}(History)) \]

\[ H1.emp_id=H2.emp_id \]
\[ H1.hire_date\neq H2.hire_date \]
Another example

• Extreme elements query:
  – Input: a total order relation $R(x,y)$
  – Output: the minimum and maximum element
    
    $$(\pi_x(R) \setminus \pi_y(R)) \cup (\pi_y(R) \setminus \pi_x(R))$$
Expressibility

• Not all queries are expressible in relational algebra
• E.g., route planning
• Not surprising
  – $\text{avg}(x,y)$ versus $\sin(x)$
The first-order queries

• Relational algebra forms an important core query language
  – SQL select-statements = rel.alg. + aggregates
  – even XPath 2.0 = relational algebra!
  – also SPARQL = relational algebra

• Queries expressible in relational algebra are called the first-order queries
  – relational calculus (first-order logic)
Semijoin

- Recall Employee query:

\[ \pi_{H1.\text{emp}_\text{id}} (\rho_{H1}(\text{History}) \bowtie \rho_{H2}(\text{History})) \]

\[ \begin{align*}
H1.\text{emp}_\text{id} &= H2.\text{emp}_\text{id} \\
H1.\text{hire}_\text{date} &\neq H2.\text{hire}_\text{date}
\end{align*} \]

- We don’t need attributes of H2 after join

- Semijoin:

\[ \pi_{H1.\text{emp}_\text{id}} (\rho_{H1}(\text{History}) \bowtie \rho_{H2}(\text{History})) \]

\[ \begin{align*}
H1.\text{emp}_\text{id} &= H2.\text{emp}_\text{id} \\
H1.\text{hire}_\text{date} &\neq H2.\text{hire}_\text{date}
\end{align*} \]
The semijoin algebra (SA)

• Same as relational algebra, except:
  \( \times \) and \( \bowtie \) are replaced by \( \Join \)

• SA queries...
  – always return subset of the relations (possibly \( \pi \))
  – can be efficiently processed
    • sorting
    • one-pass query processing
    • linear

• SA with only equalities in join conditions
  = the linear fragment of relational algebra
Searching versus Querying

• Users of information systems do not use SQL
  – Google
  – Library catalog

• Programs built over information retrieval (full text) engine cannot call SQL
  – Websites

• They can **search:**
  – ti=databases AND NOT au=ullman
  – pyrrhula OR bullfinch
Pyrrhula pyrrhula (Eurasian Bullfinch)
Abstract Dataspaces

• An abstract **dataspace** is a set of **objects**
• Each **object** is a set of **items**
• E.g., set of webpages
  – each webpage = set of strings
• E.g., classical relation is set of tuples
  – each tuple = set of attribute–value pairs
Attribute–value pairs

- Tuple

<table>
<thead>
<tr>
<th>emp_id</th>
<th>hire_date</th>
<th>job</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>20091021</td>
<td>programmer</td>
</tr>
</tbody>
</table>

- Set of attribute–value pairs

<table>
<thead>
<tr>
<th>att</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp_id</td>
<td>1234</td>
</tr>
<tr>
<td>hire_date</td>
<td>20091021</td>
</tr>
<tr>
<td>job</td>
<td>programmer</td>
</tr>
</tbody>
</table>
Attribute–value dataspaces

• Objects are arbitrary sets of AV-pairs

<table>
<thead>
<tr>
<th>attribute</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>John</td>
</tr>
<tr>
<td>paper p1</td>
<td></td>
</tr>
<tr>
<td>paper p2</td>
<td></td>
</tr>
<tr>
<td>location</td>
<td>Namur</td>
</tr>
<tr>
<td>likes</td>
<td>Orval</td>
</tr>
<tr>
<td>paper_id</td>
<td>p1</td>
</tr>
<tr>
<td>title</td>
<td>SQL</td>
</tr>
<tr>
<td>proceedings</td>
<td>VLDB</td>
</tr>
<tr>
<td>drink_type</td>
<td>beer</td>
</tr>
<tr>
<td>name</td>
<td>Orval</td>
</tr>
<tr>
<td>kind</td>
<td>Trappist</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>attribute</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Anne</td>
</tr>
<tr>
<td>paper p1</td>
<td></td>
</tr>
<tr>
<td>location Brussels</td>
<td></td>
</tr>
<tr>
<td>phone</td>
<td>022222785</td>
</tr>
<tr>
<td>name</td>
<td>Mary</td>
</tr>
<tr>
<td>paper p2</td>
<td></td>
</tr>
<tr>
<td>paper p3</td>
<td></td>
</tr>
<tr>
<td>location Brussels</td>
<td></td>
</tr>
<tr>
<td>location Antwerp</td>
<td></td>
</tr>
<tr>
<td>hobby</td>
<td>birdwatching</td>
</tr>
<tr>
<td>paper_id p2</td>
<td></td>
</tr>
<tr>
<td>title XQuery</td>
<td></td>
</tr>
<tr>
<td>proceedings VLDB</td>
<td></td>
</tr>
<tr>
<td>citations</td>
<td>55</td>
</tr>
<tr>
<td>paper_id p3</td>
<td></td>
</tr>
<tr>
<td>title Pyrrhula song</td>
<td></td>
</tr>
<tr>
<td>journal Ornithology</td>
<td></td>
</tr>
<tr>
<td>drink_type beer</td>
<td></td>
</tr>
<tr>
<td>name Orval</td>
<td></td>
</tr>
<tr>
<td>kind Trappist</td>
<td></td>
</tr>
</tbody>
</table>
Orval
“Database of everything”

- Alon Halevy
- Very similar to Semantic Web
  - RDF
  - Linked Data
- Personal Information Management
- NoSQL databases
A–V dataspace as RDF store

- RDF store: set of triples
  - (subject, predicate, object)
- view A–V dataspace $D$ as set of triples:
  - $\{(oid,att,val) : oid \in D \& (att,val) \in D\}$
RDF triple store as A–V dataspace

• Use 3 special attributes
  – subject
  – predicate
  – object

• RDF triple store is just a relation over the scheme \{subj,pred,obj\}

• Already know a relation is a dataspace!

• No RDFS
Searching Dataspaces

• Abstract Dataspace
  – set of objects
  – object: set of items

• Abstract keyword
  – predicate on items

• E.g., when items are strings:
  – string contains “Brussel”
Boolean Search Language (BSL)

• Every keyword $k$ is an expression
• Meaning:
  – Retrieve all objects containing some item satisfying $k$
• If $e_1$ and $e_2$ are expressions then so are:
  – $e_1$ OR $e_2$
  – $e_1$ AND $e_2$
  – $e_1$ AND NOT $e_2$
• Meaning: union, intersection, set difference
• Bruxelles AND NOT (Orval OR Chimay)
Dataspace search queries

• Database query:
  – mapping from databases to databases

• Dataspace query:
  – mapping $q$ from dataspaces to dataspaces

• Dataspace search query:
  – such that $q(D) \subset D$ for each $D$

• Bit like semijoin queries...
Which dataspace search queries...

• ...are expressible in BSL?
• BSL queries are safe
  – Only returns objects containing some item satisfying some keyword that we used
• BSL queries are additive
  \[ q(D) = \text{union of all } q(o) \text{ for } o \in D \]
BSL queries are **finitely distinguishing**

- Only distinguish objects using some finite set $K$ of keywords
- $o_1$ and $o_2$ are “$K$-equivalent” if for each $k$ in $K$, $o_1$ matches $k \iff o_2$ matches $k$
- When $o_1$ and $o_2$ from $D$ are $K$-equivalent then $o_1 \in q(D) \iff o_2 \in q(D)$
Characterisation of BSL

• A dataspace query \( q \) is expressible in BSL if (and only if) \( q \) is additive, and for some finite set \( K \) of keywords,
  – \( q \) is \( K \)-safe and
  – \( q \) is \( K \)-distinguishing
Application to relational selection queries

- Recall: relation = set of tuples = set of objects
- Object = set of attribute–value pairs
- Keywords: $A=c$
  - $A$: attribute from the given relation scheme
  - $c$: arbitrary constant
- Also wildcard keyword: *
- Example BSL query:
  * AND NOT (job=programmer OR emp_id=1234)
- Same as rel.alg. using only $\cup, \setminus, \sigma_{A=c}$
Characterising relational selection queries

• A relational selection query is expressible in the relational algebra using only $\cup$, $\setminus$, $\sigma_{A=c}$ if and only if it is additive and commutes with any $C$-epimorphism, for some finite set $C$ of constants.

• $C$-epimorphism: function $f$ from values to values such that $f$ and $f^{-1}$ are the identity on $C$.

• $q$ commutes with $f$:

$$q(f(D)) = f(q(D))$$

• In line with known “genericity” properties [Aho&Ullman, Chandra&Harel, Hull&Yap, Abiteboul&Vianu]
Characterisation of BSL (repeated)

• A dataspace query $q$ is expressible in BSL if (and only if) $q$ is additive, and for some finite set $K$ of keywords,
  – $q$ is $K$-safe and
  – $q$ is $K$-distinguishing
Not expressible in BSL

• Negated keywords (if you don’t have them)
  – retrieve all objects containing an item not matching “Brussel”
  – not finitely distinguishing over positive keywords

• Normally will use boolean-closed repertoire of keywords
Neither expressible in BSL

- Retrieve all objects sharing an item with an object matching “Brussel”
- Retrieve all co-authors of Mary
- Not additive
- We cannot do joins or even semijoins
- Want to do such “associative search”
Similarity relations (simrels)

• How to link two objects?
  – hardwire links between objects in the dataspace
  – not necessary
  – not flexible

• Better: use simrels between items
  – a simrel is a binary predicate on items
Examples of simrels

• Equality

• Translation on city names:
  – Bruxelles trans Brussel
  – Anvers trans Antwerpen
  – Namur trans Namen

• Equal-value on A–V pairs:
  – (likes, Orval) eqval (name, Orval)

• Equal-attribute on A–V pairs:
  – (name, John) eqatt (name, Orval)
Simlinks

• If $k$ and $k'$ are keywords, and $\approx$ is a simrel, then $k \approx k'$ is a simlink.

• Meaning: binary predicate on items
  – will be used to link objects

• $i_1 \ [ k \approx k' \ ] \ i_2$ if
  – $i_1$ satisfies $k$
  – $i_2$ satisfies $k'$
  – $i_1 \approx i_2$

• Example on string items, with substring and wildcard keywords and translation simrel:
  “Grand Place” [ Grand trans * ] “Grote Markt”
Linking objects using simlinks

• For objects $o1$ and $o2$, $o1 \ [ k \approx k' ] \ o2$ if
  – $o1$ contains some item $i1$
  – $o2$ contains some item $i2$
  – $i1 \ [ k \approx k' ] \ i2$

• New associative search operator on dataspaces: $\text{LINK} \ [ k \approx k' ] \ (S)$
  – retrieves all objects in the dataspace that are linked by $[ k \approx k' ]$ to some object in $S$

  $\text{LINK} \ [ \text{Grand trans } * \ ] \ (\text{Markt})$
Associative Search Language (ASL)

• BSL extended with link operator
• Parameterised by choice of:
  – keywords (already for BSL)
  – simrels (for link operator)
• What is the expressiveness of ASL?
• Link operator is like semijoin...
  \[ e_1 \text{ AND } \text{LINK} [ \theta ] (e_2) \]
  \[ e_1 \Theta e_2 \]
ASL on A–V dataspaces

• Keywords:
  – literals & wildcards
    (name: John)  (name: *)  (*: John)
  – negation on values
    (likes: ¬(Heineken,Budweiser))
  – negation on attributes
    (¬(paper_id,title): Orval)
  – negation on both values and attributes
    (¬(paper_id,title): ¬(Heineken,Budweiser))

• Simrels:
  – eq, eq_val, eq_att
Example query

- Retrieve all people located in Antwerp who have published a paper in *Ornithology*:
  (location: Antwerp) AND
  \[
  \text{LINK} \ [ \ (\text{paper: } \ast) \ \text{eq_val} \ (\text{paper_id: } \ast) \ ]
  \]
  (journal: Ornithology)

- Which queries can we express?
A–V dataspace as relation

• We saw this already: set of triples (oid, att, val)

• How does ASL compare to querying this relation using relational algebra?
ASL translated into semijoin algebra

(location: Antwerp) AND

\[ \text{LINK} \left[ \left( \text{paper: } * \right) \text{eq_val} \left( \text{paper_id: } * \right) \right] \]

(journal: Ornithology)

\[ \pi_{\text{oid}} \sigma_{\text{att='location'}}(T) \Join \]

\[ \pi_{\text{oid}} (\sigma_{\text{att='paper'}}(T) \Join \]

\[ \pi_{\text{val}} \sigma_{\text{att='paper_id'}}(T \Join \pi_{\text{oid}} \sigma_{\text{att='journal'}}(T))) \]

- Only natural semijoins are used
SA queries not expressible in ASL

- “Retrieve all people who have the same value for a boss and a friend attribute”
- “Retrieve all people who like some beer that nobody else likes”
- Can prove that these are not expressible using invariance under bisimulations
Bisimilarity of Dataspaces

• Dataspace $D$ and object $o$ in $D$, also $D’$ and $o’$
• Natural number $n$
• We say that $(D,o) \leftrightarrow_n (D’,o’)$ if
  – $o$ and $o’$ match precisely the same keywords
  – moreover for $n>0$:
  – for each $\text{simrel} \approx$ and for each object $p$ in $D$ such that $o \approx p$, there exists $p’$ in $D’$ such that $o’ \approx p’$ and $(D,p) \leftrightarrow_{n-1} (D’,p’)$
  – vice versa (from $D’$ to $D$)
Invariance under bisimilarity

- Let $q$ be an ASL query using at most $n$ nested link operators
- Let $(D,o) \leftrightarrow^n_n (D',o')$
- Then $(D,o)$ is indistinguishable from $(D',o')$: 
  - $o$ in $q(D)$ if and only if $o'$ in $q(D')$
- (Converse holds as well: if indistinguishable, then bisimilar)
SA queries not expressible in ASL (repeated)

• “Retrieve all people who have the same value for a boss and a friend attribute”
• “Retrieve all people who like some beer that nobody else likes”
• Can prove that these are not expressible using invariance under bisimulations
The “search” fragment of SA

\[ E ::= T \]
\[ \mid \sigma_{\text{att}=c}(E) \]
\[ \mid \sigma_{\text{val}=c}(E) \]
\[ \mid E \cup E \]
\[ \mid E \setminus E \]
\[ \mid \pi_\alpha(E) \]
\[ \mid E \Join \pi_{\text{oid}}(E) \]
\[ \mid \pi_{\text{oid}}(E \Join \pi_\beta(E)) \]

- \( c \): constant
- \( \alpha \): \{oid\}, \{oid,att\}, or \{oid,val\}
- \( \beta \): \{att\}, \{val\}, or \{att,val\}
What have we learned?

- Searching unstructured information motivates to investigate new query languages
  - but the classical theory is still very useful:
    - relational databases
    - relational algebra
    - genericity
    - semijoin algebra
    - bisimilarity
- Querying RDF triple stores
Open research problems

• Algorithms, data structures for query processing
• Are BSL and ASL sufficient? Other search primitives?
• User interface: search should be easier than full querying in SQL
• How to represent relational databases as dataspaces (or RDF) such that querying can be done by searching?
  – Querying the Deep Web [Halevy]
Orval
Computability

• Of course a query $q$ must be computable
• So, there must exist:
  – representation of databases into strings
  – algorithm A
Genericity: motivation

• Not just any crazy function is a “reasonable” database query

• E.g., random choice:
  – input: a list of names
  – output: one name from the list

• Better: minimum element query:
  – input: a list of names, and a total order over it
  – output: the minimum according to given order
Genericity: definition

• A query $q$ is **generic** if it is invariant under isomorphisms
  – formally, for any permutation $f$ of data values,
    
    $$q(f(D)) = f(q(D))$$
Not generic

• Random