What is a Query Language?

Universality of Data Retrieval Languages, Aho and Ullman, POPL 1979

Raghu Ramakrishnan

What is a Query Language?

- A language that allows retrieval and manipulation of data from a database.
- What is A Database?
  - A large collection of DATA
  - The data can be grouped into sets whose elements have similar structure.
- What Kind of Structure Can the Data Have?
- What Kind of Manipulation Should Be Allowed?

Some Ideas

- Relations should be treated as sets of tuples.
- The query language must have a simple, non-operational meaning that is independent of physical data representation.
- There must be efficient ways to process queries over (large) sets of similarly structured facts.

We will focus on the relational model

Principles for A Relational Query Language*

*Proposed by Aho & Ullman

1) Relation = Set of Tuples.
   Ordering & other storage details should not be visible.
2) Data Values should not be ‘Interpreted’.  
   Def: Let \( \mu = D \rightarrow D \) be a Bijection.  
   A Function \( f \) is Allowable if \( \mu(f(a_1, \ldots, a_k)) = f(\mu(a_1), \ldots, \mu(a_k)) \) 

Note: (2) Says that no special meaning should be attached to data values (as far as the query language is concerned); thus, Arithmetic is Disallowed! 

\[ 5 + 6 = 11, 8 < 9, \ldots \]

Allowable Fns – Transitive Closure

- Aho & Ullman’s notation of allowable function is rather restrictive. However:
  1. All Relational Algebra queries are allowable.
  2. Transitive Closure is allowable.
- And they prove that:
  * There is no Relational Algebra query that computes the Transitive Closure of a Relation.

Any R.A expression has a fixed size, say \( n \). Choose Relation R:

\[
\begin{align*}
\alpha_1 & \quad \alpha_2 & \quad \ldots & \quad \alpha_k \quad \text{such that} \quad k > n
\end{align*}
\]

The relational algebra expression cannot deal with \( \alpha_n, \alpha_k \).
Proposal

- We should extend RA to support a least fixpoint operator.
  - Leads to recursive queries
  - Some systems (e.g., Oracle) support limited forms of recursion like transitive closure. Others (DB2) support linear recursion, following SQL:1999.

Least Fixpoints

- The LFP operator is defined as follows:
  \[ \text{LFP}(R = f(R)) = r, \]
  where:
  1. \( r = f(r) \)
  2. if \( r' = f(r) \) then \( r \subseteq r' \)

- Theorem (Tarski):
  There is a least fixpoint satisfying \( \text{LFP}(R = f(R)) \) if \( f \) is monotone.
  \[ r_1 \subseteq r_2 \Rightarrow f(r_1) \subseteq f(r_2) \]
  Note: If \( f \) is a relation algebra expression without `-' (set diff.), then it is monotone.

Least Fixpoint – Cont.

- Theorem (Kleene)
  If \( f \) is continuous & over a complete lattice,
  \[ \text{LFP}(R = f(R)) = \lim_{n \to \infty} f^n(\emptyset) \]

- Example: Transitive Closure
  \[ R = R \cup r, \]
  \[ f(\emptyset) = r, \]
  \[ f(f(\emptyset)) = f(r) = r \cup r, \]
  \[ f^*(\emptyset) = \bigcup_{n \geq 1} r \cup r \cdots \cup r \]

Proof

Consider a set of \( l \) arbitrary symbols:
\[ \Sigma = \{a_1, a_2, \ldots, a_l\} \]
We consider a family of relations
\[ R'_l = \{(a_1, a_1), (a_1, a_2), \ldots, (a_{l-1}, a_l)\} \]

We show that NO relational algebra expression computes exactly the tuples in \( R'_l \) for all \( l \)

LFP - Cont.

- Claim:
  The LFP operator satisfies principles 1&2

- Theorem (Aho-Ullman):
  There is no relational algebra expression \( E(R) \) that computes the transitive closure of an arbitrary input relation \( R \).

We will prove that every R.A. expr. \( E(R, j) \) can be expressed as:
\[ \{b_1, b_2, \ldots, b_k | \Psi(b_1, b_2, \ldots, b_k)\} \]
Where
\[ \Psi \] is of the form: clause1 \( \lor \) clause2 \( \lor \cdots \) Each clause is of the form: atom1 \( \land \) atom2 \( \land \cdots \) Each atom is of the form:
\[ b_j = a_j, b_j \neq a_j, b_j = b_j + c, b_j \neq b_j + c \]
The \( b_j \)'s are variables taking values from \( \Sigma \), and the \( c \)'s are constants \( (0 \leq c \leq l) \)

Note: Here \((b_1 + c) \Rightarrow a_j \) s.t. \( b_j = a_j \)
Lemma: If \( E \) is any R.A. expr.
\[ E(R) = [b_1, \ldots, b_n | \Psi(b_1, \ldots, b_n)] \]
Suppose the lemma is true, we can then prove the theorem as follows:
Suppose \( E(R) = R' \), for some \( E \), for all \( R \), then \( R'' = [b_1, \ldots, b_n | \Psi(b_1, \ldots, b_n)] \)

Case 1: Every clause in \( \Psi \) has an atom of the form:
\[ b_i = a_j, b_i = a_k, \text{ or } b_i = b_i + c \]
Consider \( (b_i, b_j) = (a_{a_{a_{a_{\ldots}}}}) \) where
\( m > i \) s.t. \( b_i = a_k \) or \( b_i = a_1 \) is an atom;
\( d > i \) s.t. \( b_i = b_i + c \) is an atom
\[ \vdash (a_{a_{a_{a_{\ldots}}}}) \] is not computed, but is in \( R'' \)

Case 2: Some clause in \( \Psi \) has ONLY atoms with \( \neq \)
Consider \( (b_i, b_j) = (a_{a_{a_{a_{\ldots}}}}) \)
Where no atom
\[ b_i \neq a_k, a_k \neq a_{a_{a_{a_{\ldots}}}} \]
appears in \( \Psi \), and
\( d > c, \) for all \( c \) s.t. \( b_j = b_i + c \) or \( b_j = b_i + c \)
appears in \( \Psi \),
\[ \vdash (a_{a_{a_{a_{\ldots}}}}) \] is computed, but is not in \( R'' \)

\[
\begin{array}{c}
\vdash (a_{a_{a_{a_{\ldots}}}}) \text{ is not computed, but is in } R'' \\
\end{array}
\]

\[
\begin{array}{c}
\vdash (a_{a_{a_{a_{\ldots}}}}) \text{ is computed, but is not in } R'' \\
\end{array}
\]

Proof of lemma

Base: 0 operators. \( \vdash E(R) = R \) or constant relation.
\[ R = [b_1, b_2, \ldots, b_n | b_i = b_i + 1] \]
\[ [c_1, c_2, \ldots, c_m] = [b_1, c_1 = c_2 = \ldots = ] \]

Induction:
\[ E = E_1 \cup E_2, E_1 \cup E_2, \text{ or } E_1 \times E_2 \]
\[ E = [b_1, \ldots, b_n | \Psi(b_1, \ldots, b_n)] \]
\[ E_i = [b_i, \ldots, b_n | \Psi(b_i, \ldots, b_n)] \]
\[ E_i = [b_i, \ldots, b_n | \Psi(b_i, \ldots, b_n) \cup \Psi(b_i, \ldots, b_n)] \]
\[ E = \sigma_{(E_1 \times E_2)} F \text{ has only } \vdash \]
\[ E = [b_1, \ldots, b_n | \Psi(b_1, \ldots, b_n) \times F(b_1, \ldots, b_n)] \]
\[ E = \sigma_{(E_1 \times E_2)} ; \text{ proceeding similarly} \]

Transitive closure - more

\[ R_i = \{(a_1, a_2), (a_2, a_3), \ldots, (a_{n-1}, a_n)\} \]
\[ \sigma_{(E_1 \times E_2)} \]
Does this relational algebra expr. computes \( R'' \)?

YES! But it is NOT a relation algebra expression!

\[
\begin{array}{c}
\vdash (a_{a_{a_{a_{\ldots}}}}) \text{ is not computed, but is in } R'' \\
\end{array}
\]

BP-Completeness

\[ A \text{ query language is BP-complete if:} \]
\[ \bullet \] All functions that can be expressed in the language are allowable.
\[ \bullet \] Let \( r_1 \) and \( r_2 \) be two relations (instances), such that for all renamings \( \mu \)
\[ r_1 = \mu(r_1) \Rightarrow r_2 = \mu(r_2) \]
Then there is a function \( f \) in the language such that
\[ r_2 = f(r_1) \]

Transitive closure - more

\[ R_i = \{(a_1, a_2), (a_2, a_3), \ldots, (a_{n-1}, a_n)\} \]
\[ \sigma_{(E_1 \times E_2)} \]
Does this relational algebra expr. computes \( R'' \)?

YES! But it is NOT a relation algebra expression!

\[
\begin{array}{c}
\vdash (a_{a_{a_{a_{\ldots}}}}) \text{ is not computed, but is in } R'' \\
\end{array}
\]
Example of BP-Complete

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>10</td>
<td>11</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

1. If ‘A’ is used as ‘r1’ in previous slide, which of the others qualifies as ‘r1’?

2. For each such relation, find relational algebra function $f$. 