What is a Query Language?

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

Raghu Ramakrishnan
What is ...?

- 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:

$$\mu(f(r_1,\ldots,r_n)) = f(\mu(r_1),\ldots,\mu(r_n))$$

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!

$5+6 = 11$, $8<9$, …
Principles – Refinement

❖ Principle (2) is too restrictive.
❖ Relax it slightly:

Let $P$ be a special set of predicates. (e.g. $<, =$)

$\mu$ Preserves $P$ if $\forall p \in P$

$\mu(p(x_1, \ldots, x_n))$ is true $\iff p(\mu(x_1), \ldots, \mu(x_n))$ is true.

Relaxing Principle (2): We require that:

$\mu(f(r_1, \ldots, r_n)) = f(\mu(r_1), \ldots, \mu(r_n))$

only for Bijections $\mu$ that preserve $P$.

Note: If we include $+, \times$, etc. to $P$, soon only the identity function will preserve $P$!
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:

\[
\begin{align*}
a_1 & \quad a_2 & \quad \cdots & \quad a_k & \quad k > n
\end{align*}
\]

The relational algebra expression cannot deal with \((a_1, a_k)\).
Proposal

- We should extent 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:

\[ LFP(R = f(R)) = r, \] \text{where:} 

1. \( r = f(r) \)  
2. if \( r' = f(r') \) then \( r \subseteq r' \)

- **Theorem (Tarski):**
  There is a least fixpoint satisfying \( LFP(R=f(R)) \) if \( f \) is **monotone**.

**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

  \[ R = R \circ r \cup r; \]
  \[ \therefore f(R) \text{ is } R \circ r \cup r \]
  \[ f(\emptyset) = r; \]
  \[ f(f(\emptyset)) = f(r) = r \circ r \cup r \]
  \[ \therefore f^n(\emptyset) = \bigcup_{i=1}^{n} r \circ r \circ \ldots \circ r \]
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.
**Proof**

Consider a set of $l$ arbitrary symbols:

$$\Sigma_l = \{a_1, a_2, \ldots, a_l\}$$

We consider a family of relations

$$R_l = \{(a_1, a_2), (a_2, a_3), \ldots, (a_{l-1}, a_l)\}$$

We show that NO relational algebra expression computes exactly the tuples in $R_l^+$ for all $l$
We will prove that every R.A. expr. \( E(R_1) \) can be expressed as:
\[
\{ b_1 b_2 \cdots b_k \mid \Psi(b_1, b_2, \cdots b_k) \}
\]
Where
\( \Psi \) is of the form: \( \text{clause 1} \lor \text{clause 2} \lor \cdots \)
Each \( \text{clause} \) is of the form: \( \text{atom 1} \land \text{atom 2} \land \cdots \)
Each \( \text{atom} \) is of the form:
\[
b_i = a_c, b_i \neq a_c, b_i = b_j + c, b_i \neq b_j + c
\]
The \( b \)'s are variables taking values from \( \Sigma_1 \), and the \( c \)'s are constants \( (0 \leq c \leq l) \)

Note: Here \((b_j + c) \equiv a_m \text{ s.t. } b_j = a_{m-c}\)
Lemma: If $E$ is any R.A. expr.
$E(R_l) = \{b_1 b_2 \cdots b_k \mid \Psi(b_1, b_2, \ldots b_k)\}$

Suppose the lemma is true, we can then prove the theorem as follows:
Suppose $E(R) = R^+$, for some $E$, for all $R$, then $R_l^+ = \{b_1 b_2 \mid \Psi(b_1, b_2)\}$

Case 1: Every clause in $\Psi$ has an atom of the form:
$b_1 = a_i, b_2 = a_i$, or $b_1 = b_2 + c$
Consider $(b_1, b_2) = (a_m, a_{m+d})$ where
$m > \forall i \text{ s.t. } b_1 = a_i$ or $b_2 = a_i$ is an atom;
$d > \forall c \text{ s.t. } b_1 = b_2 + c$ is an atom
$\therefore (a_m, a_{m+d})$ is not computed, but is in $R_l^+$
**Case 2**: Some clause in $\Psi$ has ONLY atoms with $\neq$

Consider $(b_1, b_2) = (a_{m+d}, a_m)$

Where no atom

$b_i \neq a_m$ or $b_i \neq a_{m+d}$

appears in $\Psi$, and

$d > c$, for all $c$ s.t. $b_1 \neq b_2 + c$ or $b_2 \neq b_1 + c$

appears in $\Psi$.

$\therefore (a_{m+d}, a_m)$ is computed, but is not in $R_l^+$
Proof of lemma

Basis: 0 operators. \( \therefore E(R) \) is \( R \) or constant relation.
\[
R = \{ b_1 b_2 | b_2 = b_1 + 1 \};
\]
\[
\{ c_1, c_2, \ldots c_m \} = \{ b_1 | b_1 = c_1 \lor b_1 = c_2 \lor \cdots \}
\]

Induction:
\[
E = E_1 \cup E_2, \quad E_1 \cdot E_2 \quad \text{or} \quad E_1 \times E_2
\]
\[
E_1 = \{ b_1 \cdots b_k | \Psi_1(b_1 \cdots b_k) \}
\]
\[
E_2 = \{ b_1' \cdots b_k' | \Psi_2(b_1' \cdots b_k') \}
\]
\[
E_1 \cup E_2 = \{ b_1 \cdots b_k | \Psi_1(b_1 \cdots b_k) \lor \Psi_2(b_1 \cdots b_k) \}
\]
\[
E = \sigma_F(E_1), \quad F \text{ has only } =, \neq
\]
\[
\therefore E = \{ b_1 \cdots b_k | \Psi_1(b_1 \cdots b_k) \land F(b_1 \cdots b_k) \}
\]
\[
E = \pi_S(E_1), \quad \text{proceeding similarly} \ldots
\]
Transitive closure - more

\[ R_l = \{(a_1, a_2), (a_2, a_3) \cdots (a_{l-1}, a_l)\} \]

\[ \sigma_{1<2} (\pi_1 (R_l) \times \pi_2 (R_l)) \]

Does this relational algebra expr. computes \( R_l^+ \) ?
Transitive closure - more

\[ R_l = \{(a_1, a_2), (a_2, a_3) \ldots (a_{l-1}, a_l)\} \]
\[ \sigma_{1<2}(\pi_1(R_l) \times \pi_2(R_l)) \]

Does this relational algebra expr. computes \( R_l^+ \)?

YES! But it is NOT a relation algebra expression!

What does “\( a_i < a_j \)” mean now?!
**BP-Completeness**

- A query language is BP-complete if:
  - All functions that can be expressed in the language are allowable.
  - 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
    
    \[ r_2 = f(r_1) \]
**Example of BP-Complete**

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

1. If ‘A’ is used as ‘$r_j$’ in previous slide, which of the others qualifies as ‘$r_2$’?

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