Relational Calculus

Relational Calculus
- Comes in two flavors: Tuple relational calculus (TRC) and Domain relational calculus (DRC).
- Calculus has variables, constants, comparison ops, logical connectives and quantifiers.
  - TRC: Variables range over (i.e., get bound to) tuples.
  - DRC: Variables range over domain elements (= field values).
  - Like SQL
  - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called formulas.
- Answer tuple is an assignment of constants to variables that make the formula evaluate to true.

Tuple Relational Calculus
- Query has the form: \{T | p(T)\}
  - p(T) denotes a formula in which tuple variable T appears.
- Answer is the set of all tuples T for which the formula p(T) evaluates to true.
- Formula is recursively defined:
  - start with simple atomic formulas (get tuples from relations or make comparisons of values)
  - build bigger and better formulas using the logical connectives.

TRC Formulas
- An Atomic formula is one of the following:
  - R \in Rel
  - R.a op S.b
  - R.a op constant
  - op is one of: \{<,>,=,\leq,\geq\}
- A formula can be:
  - an atomic formula
  - \neg p, p \land q, p \lor q where p and q are formulas
  - \exists R(p(R)) where variable R is a tuple variable
  - \forall R(p(R)) where variable R is a tuple variable

Free and Bound Variables
- The use of quantifiers \exists X and \forall X in a formula is said to bind X in the formula.
  - A variable that is not bound is free.
- Let us revisit the definition of a query:
  - \{T | p(T)\}
- There is an important restriction
  - the variable T that appears to the left of `|` must be the only free variable in the formula p(T).
  - in other words, all other tuple variables must be bound using a quantifier.

Selection and Projection
- Find all sailors with rating above 7
  \{S \mid S \in\text{Sailors} \land S\.\text{rating} > 7\}
  - Modify this query to answer: Find sailors who are older than 18 or have a rating under 9, and are called 'Bob'.
- Find names and ages of sailors with rating above 7.
  \{S \mid \exists S1 \in\text{Sailors}(S1\.\text{rating} \geq 7 \land S\.\text{sname} = S1\.\text{sname} \land S\.\text{age} = S1\.\text{age})\}
  - Note, here S is a tuple variable of 2 fields (i.e. (S) is a projection of sailors), since only 2 fields are ever mentioned and S is never used to range over any relations in the query.
Find sailors rated > 7 who’ve reserved boat #103

\{S | S \in \text{Sailors} \land S\.rating > 7 \land \exists R (R \in \text{Reserves} \land R\.sid = S\.sid \land R\.bid = 103)}

Note the use of \( \exists \) to find a tuple in Reserves that `joins with' the Sailors tuple under consideration.

Joins (continued)

Find sailors rated > 7 who’ve reserved boat #103

\{S | S \in \text{Sailors} \land S\.rating > 7 \land \exists R (R \in \text{Reserves} \land R\.sid = S\.sid \land R\.bid = 103)}

Find sailors rated > 7 who’ve reserved a red boat

\{S | S \in \text{Sailors} \land S\.rating > 7 \land \exists R (R \in \text{Reserves} \land R\.sid = S\.sid \land \exists B (B \in \text{Boats} \land B\.bid = R\.bid \land B\.color = \text{‘red’})}\}

• Observe how the parentheses control the scope of each quantifier's binding. (Similar to SQL!)

Division (makes more sense here???)

Find sailors who’ve reserved all boats

(hint, use \( \forall \))

\{S | S \in \text{Sailors} \land \forall B (B \in \text{Boats} \land \exists R (R \in \text{Reserves} \land R\.sid = S\.sid \land R\.bid = B\.bid))}\}

• Find all sailors \( S \) such that for each tuple \( B \) in Boats there is a tuple in Reserves showing that sailor \( S \) has reserved it.

Division - a trickier example...

Find sailors who’ve reserved all red boats

\{S | S \in \text{Sailors} \land \forall B (B\.color = \text{‘red’} \Rightarrow \exists R (R \in \text{Reserves} \land R\.sid = S\.sid \land R\.bid = B\.bid))\}

Alternatively...

\{S | S \in \text{Sailors} \land \forall B (B\.color \neq \text{‘red’} \lor \exists R (R \in \text{Reserves} \land R\.sid = S\.sid \land R\.bid = B\.bid))\}

\( a \Rightarrow b \) is the same as \( \neg a \lor b \)

<table>
<thead>
<tr>
<th>a</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

• If \( a \) is true, \( b \) must be true for the implication to be true. If \( a \) is true and \( b \) is false, the implication evaluates to false.
• If \( a \) is not true, we don’t care about \( b \), the expression is always true.

Unsafe Queries, Expressive Power

• \( \exists \) syntactically correct calculus queries that have an infinite number of answers! Unsafe queries.
  - e.g., \( \exists S \mid \neg \{S \in \text{Sailors}\} \)
  - Solution???? Don’t do that!
• Expressive Power (Theorem due to Codd):
  - every query that can be expressed in relational algebra can be expressed as a safe query in DRC / TRC; the converse is also true.
• Relational Completeness: Query language (e.g., SQL) can express every query that is expressible in relational algebra/calculus. (actually, SQL is more powerful, as we will see...)
Summary

• The relational model has rigorously defined query languages — simple and powerful.
• Relational algebra is more operational
  - useful as internal representation for query evaluation plans.
• Relational calculus is non-operational
  - users define queries in terms of what they want, not in terms of how to compute it. (Declarative)
• Several ways of expressing a given query
  - a query optimizer should choose the most efficient version.
• Algebra and safe calculus have same expressive power
  - leads to the notion of relational completeness.

Midterm I - Info

• Remember - Lectures, Sections, Book & HW1
• 1 Cheat Sheet (2 sided, 8.5x11) - No electronics.
• Tues 2/21 in class
• Topics: next

Midterm I - Topics

• Ch 1 - Introduction - all sections
• Ch 3 - Relational Model - 3.1 thru 3.4
• Ch 9 - Disks and Files - all except 9.2 (RAID)
• Ch 8 - Storage & Indexing - all
• Ch 10 - Tree-based IXs - all
• Ch 11 - Hash-based IXs - all
• Ch 4 - Rel Alg & Calc - all (except DRC 4.3.2)

Addendum: Use of ∀

• ∀x (P(x)) - is only true if P(x) is true for every x in the universe
• Usually:
  ∀x ((x ∈ Boats) ⇒ (x.color = “Red”))
• ⇒ logical implication,
  a ⇒ b means that if a is true, b must be true
  a ⇒ b is the same as ¬a ∨ b

Find sailors who’ve reserved all boats

{S | S ∈ Sailors ∧
   ¬∀B (B ∈ Boats) ⇒
   ∃R(∃Reserves ∧ S.sid = R.sid ∧ B.bid = R.bid))

• Find all sailors S such that for each tuple B either it is not a tuple in Boats or there is a tuple in Reserves showing that sailor S has reserved it.

Find sailors who’ve reserved all red boats

{S | S ∈ Sailors ∧
   ¬∀B (¬B.Boats ∧ B.color = “Red”) ⇒
   ∃R(∃Reserves ∧ S.sid = R.sid ∧ B.bid = R.bid))

• Find all sailors S such that for each tuple B either it is not a tuple in Boats or there is a tuple in Reserves showing that sailor S has reserved it.

... reserved all red boats

{S | S ∈ Sailors ∧
   ∀B(¬(B ∈ Boats ∧ B.color ≠ “red”)) ⇒
   ∃R(∀Reserves ∧ S.sid = R.sid ∧ B.bid = R.bid))

• Find all sailors S such that for each tuple B either it is not a tuple in Boats or there is a tuple in Reserves showing that sailor S has reserved it.