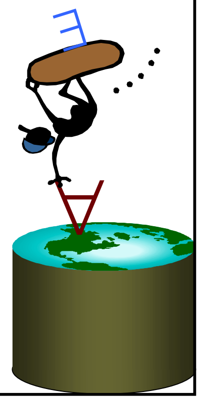
Relational Calculus

15-415, Spring 2003, Lecture 10
R&G, Chapter 4



> We will occasionally use this arrow notation unless there is danger of no confusion.
>
> Ronald Graham
> *Elements of Ramsey Theory*

TRC Formulas

- ***An Atomic formula is one of the following:***
  - *R* ∈ *Rel*
  - *R.a op S.b*
  - *R.a op constant*
    - *op* is one of $<, >, =, \le, \ge, \ne$
- **A *formula* can be*:***
  - an atomic formula
  - $\neg p, p \wedge q, p \vee q, p \Rightarrow q$ 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

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*.
    - Like SQL.
  - *DRC*: Variables range over *domain elements* (= field values).
    - Like Query-By-Example (QBE)
  - 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*.**

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.

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:**
  - v start with simple *atomic formulas* (get tuples from relations or make comparisons of values)
  - v build bigger and better formulas using the *logical connectives*.

Selection and Projection

- **Find all sailors with rating above 7**

$$\{S \mid S \in Sailors \wedge S.rating > 7\}$$

  - 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.**

$$\{S \mid \exists S1 \in Sailors(S1.rating > 7 \wedge S.sname = S1.sname \wedge S.age = S1.age)\}$$

  - 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.

Joins

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

{S | S∈Sailors ∧ S.rating > 7 ∧
∃R(R∈Reserves ∧ R.sid = S.sid
∧ R.bid = 103)}

Note the use of ∃ to find a tuple in Reserves that `joins with' the Sailors tuple under consideration.

Joins (continued)

{S | S∈Sailors ∧ S.rating > 7 ∧
∃R(R∈Reserves ∧ R.sid = S.sid
∧ ∃B(B∈Boats ∧ B.bid = R.bid
∧ B.color = 'red'))}

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

- **Observe how the parentheses control the scope of each quantifier's binding.**
- **This may look cumbersome, but it's not so different from SQL!**

Division (makes more sense here???)

Find sailors who've reserved all boats
(hint, use ∀)

{S | S∈Sailors ∧
∀B∈Boats (∃R∈Reserves
(S.sid = R.sid
∧ B.bid = R.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∈Sailors ∧
∀B ∈ Boats (B.color = **'red'** ⇒
∃R(R∈Reserves ∧ S.sid = R.sid
∧ B.bid = R.bid))}

Alternatively...

{S | S∈Sailors ∧
∀B ∈ Boats (B.color ≠ **'red'** ∨
∃R(R∈Reserves ∧ S.sid = R.sid
∧ B.bid = R.bid))}

a ⇒ b is the same as ¬a ∨ b

a \ b	T	F
T	T	F
F	T	T

- **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

- **∃ syntactically correct calculus queries that have an infinite number of answers! *Unsafe* queries.**
  - e.g., $\{S \mid \neg\{S \in Sailors\}\}$
  - 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*.

... reserved all red boats

$$\{S \mid S \in Sailors \wedge \forall B((B \in Boats \wedge B.color = \text{“red”}) \Rightarrow \exists R(R \in Reserves \wedge S.sid = R.sid \wedge B.bid = R.bid))\}$$

- **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.**

$$\{S \mid S \in Sailors \wedge \forall B(\neg(B \in Boats) \vee (B.color \neq \text{“red”}) \vee \exists R(R \in Reserves \wedge S.sid = R.sid \wedge B.bid = R.bid))\}$$

Addendum: Use of $\forall$

- $\forall x\ (P(x))$ - is only true if P(x) is true for *every* x in the universe
- Usually:

  $\forall x\ ((x \in Boats) \Rightarrow (x.color = \text{“Red”}))$
- $\Rightarrow$ logical implication,

  $a \Rightarrow b$ means that if a is true, b must be true

  $a \Rightarrow b$ is the same as $\neg a \vee b$

Find sailors who've reserved all boats

$$\{S \mid S \in Sailors \wedge \forall B((B \in Boats) \Rightarrow \exists R(R \in Reserves \wedge S.sid = R.sid \wedge B.bid = R.bid))\}$$

- **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.**

$$\{S \mid S \in Sailors \wedge \forall B(\neg(B \in Boats) \vee \exists R(R \in Reserves \wedge S.sid = R.sid \wedge B.bid = R.bid))\}$$