COS 425:
Database and Information
Management Systems

Relational model:
Relational calculus

Tuple Relational Calculus

Queries are formulae, which define sets using:

1. Constants
2. Predicates (like *select* of algebra)
3. Boolean and, or, not
4. ∃ there exists
5. ∀ for all

- Variables range over tuples
- Attributes of a tuple T can be referred to in predicates using T.attribute_name

Example: { T | T ε *tp* and T.rank > 100 }
$\underbrace{\text{T ε tp and T.rank > 100}}_{\text{formula, T free}}$

tp: (name, rank); base relation of database

Formula defines relation

- Free variables in a formula take on the values of tuples
- A tuple is in the defined relation if and only if when substituted for a free variable, it satisfies (makes true) the formula

Free variable:

∃ x, ∀ x bind x – truth or falsehood no longer depends on a specific value of x

If x is not bound it is free

Quantifiers

There exists: $\exists x\ (f(x))$ for formula f with free variable x

- Is true if there is *some tuple* which when substituted for x makes f true

For all: $\forall x\ (f(x))$ for formula f with free variable x

- Is true if *any tuple* substituted for x makes f true
  i.e. all tuples when substituted for x make f true

Example

{T | ∃A ∃B (A ε *tp* and B ε *tp* and
A.name = T.name and A.rank = T.rank and B.rank =T.rank and T.name2= B.name) }

- T not constrained to be element of a named relation
- T has attributes defined by naming them in the formula: T.name, T.rank, T.name2
  - so schema for T is (name, rank, name2) *unordered*
- Tuples T in result have values for (name, rank, name2) that satisfy the formula
- *What is the resulting relation?*

Formal definition: formula

- A tuple relational calculus formula is
  - An atomic formula (uses predicate and constants):
    - T ε R where
      - T is a variable ranging over tuples
      - R is a named relation in the database – a ***base relation***
    - T.a **op** W.b where
      - a and b are names of attributes of T and W, respectively,
      - op is one of $< \ > \ = \ \neq \ \leq \ \geq$
    - T.a **op** constant
    - constant **op** T.a

Formal definition: formula cont.

- A tuple relational calculus formula is
  - An atomic formula
  - For any tuple relational calculus formulae f and g
    - not(f)
    - f and g
    - f or g
    - $\exists$ T(f (T)) for T free in f
    - $\forall$T(f (T)) for T free in f

Boolean operations

Quantified

Formal definition: query

A query in the relational calculus is a set definition

{T | f(T) }

where f is a relational calculus formula

T is the only variable free in f

The query defines the relation consisting of tuples T that satisfy f

The attributes of T are either defined by name in f or inherited from base relation R by a predicate T ε R

Some abbreviations for logic

- (p => q) equivalent to ((not p) or q)
- $\forall$x(f(x)) equiv. to not($\exists$x(not f(x)))
- $\exists$x(f(x)) equiv. to not($\forall$x(not f(x)))
- $\forall$x ε S (f) equiv. to $\forall$x ((x ε S) => f)
- $\exists$x ε S (f) equiv. to $\exists$x ((x ε S) and f)

Board examples

Board example 3 revisited: Recall for this example we working with relations
Acct: (bname, <u>acct#,</u> bal) Branch: (<u>bname</u>, bcity, assets)
Owner: (name, <u>acct#)</u> where "name" is name of customer owning acct#

Want to express in tuple relational calculus
"names of all customers who have accounts at all branches in Princeton"

Solution worked up on board (just reordered sequence of ands):

{T |∃O ∀B ((B ε Branch and B.bcity = 'Princeton') =>
∃A (A ε Acct and O ε Owners and A.acct# = O.acct# and
B.bname = A.bname and T.name=O.name)) }

says if "xxx" is an name in the result, some (xxx, nnn) ε Owner can be paired with (b1, Princeton, $$b1) ε Branch so is (b1, nnn, bal1) ε Acct and paired with (b2, Princeton, $$b2) ε Branch so is (b2, nnn, bal2) ε Acct

Is key of Acct => WRONG

CORRECT:

{T |∀B ∃O ((B ε Branch and B.city = 'Princeton') =>
∃A (A ε Acct and O ε Owners and A.acct# = O.acct# and
B.bname = A.bname and T.name=O.name)) }

Evaluating query in calculus

Declarative – how build new relation {x|f(x)}?

- Go through each candidate tuple value for x
- Is f(x) true when substitute candidate value for free variable x?
- If yes, candidate tuple is in new relation
- If no, candiate tuple is out

What are candidates?

- Do we know domain of x?
- Is domain finite?

Problem

- Consider {T | not (T ε *tp*) }
  - Wide open – what is schema for T?
- Consider {T | ∀S ((S ε *tp*) =>
  (not (T.name = S.name and
  T.rank = S.rank))) }
  - Now T:(name, rank) but universe is infinite

Don't want to consider infinite set of values

Constants of a database and query

Want consider only finite set of values

- What are constants in database and query?

Define:

- Let I be an instance of a database
  - A specific set of tuples (relation) for each base relational schema
- Let Q be a relational calculus query
- Domain (I,Q) is the set of all constants in Q or I

Safe query

- A query Q on a relational database with base schemas $\{R_i\}$ is safe if and only if for all instances I of $\{R_i\}$, any tuple in Q(I) – the relation resulting from applying Q to I – contains only values in Domain(I, Q)
- Means at worst candidates are all tuples can form from finite set of values in Domain(I, Q)

Text goes further

- Requires testing quantifiers has finite universe:
  - For each ∃T(p(T)) in the formula of Q, if p(*t*) is true for tuple *t*, then attributes of *t* are in Domain(I, Q)
  - For each ∀T(p(T)) in the formula of Q, if *t* is a tuple containing a constant not in Domain(I,Q), then p(*t*) is true

=> Only need to test tuples in Domain(I,Q)

The relational algebra and
the tuple relational calculus
over safe queries
are equivalent in expressiveness

Domain relational calculus

- Similar but variables range over domain values (i.e. attributes) not tuples
- Is equivalent to tuple relational calculus
- Example:

{<N, K, M> | (N, K) ε *tp* and (M, K) ε *tp* }

Summary

- The relational calculus provides an alternate way to express queries
- A formal model based on logical formulae and set theory
- Equivalence with algebra means can use either or both – but only one for formal proofs
- Next we will see that SQL borrows from both