COS 597A:
Principles of
Database and Information 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

Value of an attribute of a tuple T can be referred to in predicates using **T[attribute_name]**

Example: { T | T ε Winners and T[year] > 2006 }
|__formula, T free ______|

Winners: (name, tournament, year); 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: ∃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: ∀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 Winners and B ε Winners and
A[name] = T[name] and A[tournament] = T[tournament] and
B[tournament] =T[tournament] and T[name2] = B[name]) }

- T not constrained to be element of a named relation
- Result has attributes defined by naming them in the formula: T[name], T[tournament], T[name2]
  - so schema for result: (name, tournament, name2) *unordered*
- Tuples T in result have values for (name, tournament, 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 < > = ≠ ≤ ≥
    - 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
    - (f)
    - not(f)
    - f and g
    - f or g
    - ∃T(f (T)) for T free in f
    - ∀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 ***Result*** consisting of tuples T that satisfy f

The attributes of ***Result*** 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)
- ∀x(f(x)) equiv. to not(∃x(not f(x)))
- ∃x(f(x)) equiv. to not(∀x(not f(x)))
- ∀x ε S (f) equiv. to ∀x ((x ε S) => f)
- ∃x ε S (f) equiv. to ∃x ((x ε S) and f)

Example: relating to algebra

- How do projection in calculus?

$\pi_{name,year}$ (Winners)

becomes

{ T | ∃W (W ε Winners ∧
T[name] = W[name] ∧
T[year] = W[year]) }

∧ denotes AND

Board Examples

Database:

students: (<u>SS#</u>, name, PUaddr, homeAddr, classYr)
employees: (<u>SS#</u>, name, addr, startYr)
jobs: (<u>position</u>, division, SS#, managerSS#)
division *foreign key referencing* PUdivision
study: (<u>SS#,</u> academic_dept., adviser)
SS# *foreign key referencing* students
PUdivision: (<u>division_name</u>, address, director)

Board Example 1

find SS#, name, and classYr of all student employees

Board Example 2

find (student, manager) pairs where both are students - report SS#s

Board Example 3

find *names* of all CS students working for the library (library a division)

Board Example 4

Find academic departments that have students working in all divisions

Clarifying issue with ∀

Consider simpler example than Board Example 4

Relations: for_sale:(house, town)
showing:(client, house)
house foreign key references for_sale

Query: clients who have seen all houses for sale

Try:
{T | ∀F ε for_sale ∃W ε showing
(T[client] = W[client] and W[house]=F[house]) }

If no houses for sale at this instant? i.e. F empty

Relations: for_sale:(house, town)
showing:(client, house)
house foreign key references for_sale

Query: clients who have seen all houses for sale

If for_sale empty, "∀F ε for_sale (…)" is true
Then ***any*** tuple T over domain of client satisfies => infinite
Fix: Adding leading, independent ∃ :

{T | ∃S ε showing (T[client]=S[client]) and
∀F ε for_sale ∃W ε showing
(T[client] = W[client] and W[house]=F[house]) }

Now what is result if for_sale is empty?

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, candidate tuple is out

What are candidates?
- Do we know domain of x?
- Is domain finite?

Problem

- Consider {T | not (T ε *Winners*) }
  - Wide open – what is schema for Result?
- Consider {T | ∀S ((S ε *Winners*) =>
(not (T[name] = S[name] and
T[year] = S[year]))) }
  - Now Result:(name, year) 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
- Let Q(I) denote the relation resulting from applying Q to I

Safe query

A query Q on a relational database with base schemas $\{R_i\}$ is safe if and only if:

1. for all instances I of $\{R_i\}$, any tuple in Q(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)

Safe query: need more

Require testing quantifiers has finite universe:

2. For each $\exists T(p(T))$ in the formula of Q, if p(*t*) is true for tuple *t*, then attributes of *t* are in Domain(I, Q)

3. For each $\forall T(p(T))$ 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)

Safe query: all conditions

A query Q on a relational database with base schemas $\{R_i\}$ is safe if and only if:

1. for all instances I of $\{R_i\}$, any tuple in Q(I) contains only values in Domain(I, Q)
2. For each $\exists T(p(T))$ in the formula of Q, if p(*t*) is true for tuple *t*, then attributes of *t* are in Domain(I, Q)
3. For each $\forall T(p(T))$ in the formula of Q, if *t* is a tuple containing a constant not in Domain(I,Q), then p(*t*) is true

Equivalence Algebra and Calculus

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. attribute values) not tuples
- Is equivalent to tuple relational calculus when both restricted to safe expressions

Example:
{<N, K, M> | ∃Y ∃Z (<N, K,Y> ε *Winners* and
<M, K, Z> ε *Winners*) }

N, M range over Winners.name
K ranges over Winners.tournament
Y, Z range over Winners.year

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