#### COS 597A: Principles of Database and Information Systems

Relational model: Relational calculus

#### Modeling access

- Have looked at modeling information as data + structure
- Now: how model access to data in relational model?
- · Formal specification of access provides:
  - Unambiguous queries
  - Correctness of results
  - Expressiveness of query languages

## Queries

A query is a mapping from a set of relations to a relation

Query: relations  $\rightarrow$  relation

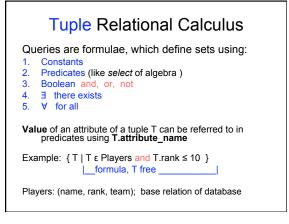
- Can derive schema of result from schemas of input relations
- Can deduce constraints on resulting relation that
   must hold for any input relations
- · Can identify properties of result relation

## Relational query languages

- Two formal relational languages to describe mapping
   Relational calculus
  - Declarative describes results of query
  - Relational algebra
    - Procedural lists operations to form query result
- · Equivalent expressiveness
- Each has strong points for usefulness
- DB system query languages (e.g. SQL) take best of both

### begin with Relational Calculus

- Two forms
  - Tuple relational calculus: variables of formulae range over tuples
  - Domain relational calculus: variables of formulae range over attributes



#### 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: ∀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: there exists

{T |3A 3B (A ε Players and B ε Players and A.name = T.name and A.rank > B.rank and B.name = T.name2)}

- · T not constrained to be element of a named relation
- Result has attributes defined by naming them in the formula: T.name, T.name2

 so schema for result: (name, name2) unordered

- Tuples T in result have values for (name, name2) that satisfy the formula
- What is the resulting relation?

# Example: for all 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 ( F ɛ for\_sale => ∃W (W ɛ showing and T.client = W.client and W.house=F.house)) } Shorthand: {T | ∀F ɛ for\_sale ∃W ɛ showing (T.client = W.client and W.house=F.house) }

Relations: for\_sale:(<u>house</u>, town) showing:(client, house) house foreign key references for\_sale

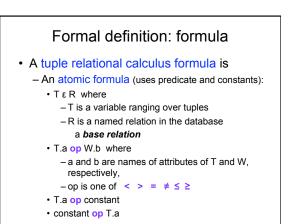
Query: clients who have seen all houses for sale

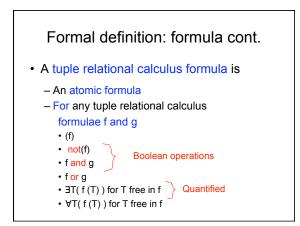
If for\_sale empty, " $\forall$ F ( F  $\epsilon$  for\_sale => ...)" is true Then *any* tuple T satisfies and result is infinite set

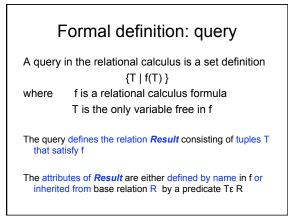
Fix: Adding leading, independent 3 :

{T | 3S ε showing (T.client=S.client) and ∀F ε for\_sale 3W ε showing (T.client = W.client and W.house=F.house) }

Now what is result if for\_sale is empty?







### Some abbreviations for logic

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

## Board examples

#### Board Example 1

students: (<u>SS#</u>, name, PUaddr, homeAddr, Yr) employees: (<u>SS#</u>, name, addr, startYr) jobs: (<u>position</u>, division, SS#, managerSS#) study: (<u>SS#</u>, academic\_dept., adviser)

find SS#, name, and Yr of all students employees

#### Board Example 2

students: (<u>SS#</u>, name, PUaddr, homeAddr, Yr) employees: (<u>SS#</u>, name, addr, startYr) jobs: (<u>position</u>, division, SS#, managerSS#) study: (<u>SS#</u>, academic\_dept., adviser)

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

#### Board Example 2

students: (<u>SS#</u>, name, PUaddr, homeAddr, Yr) employees: (<u>SS#</u>, name, addr, startYr) jobs: (<u>position</u>, division, SS#, managerSS#) study: (<u>SS#</u>, academic\_dept., adviser)

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

#### Board Example 3

students: (<u>SS#</u>, name, PUaddr, homeAddr, Yr) employees: (<u>SS#</u>, name, addr, startYr) jobs: (<u>position</u>, division, SS#, managerSS\$) study: (SS#, academic dept., adviser)

# Find divisions that have students from all departments working in them

Interpret "all departments" to be all departments that appear in jobs.academic\_dept.

#### 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 guery?

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<sub>i</sub>} , 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:

- 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)

#### 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  $Domain(I,\,Q)$
- 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

#### 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:

A, B range over Players.name

R, S range over Players.rank

#### Summary

- The relational calculus provides a formal model based on logical formulae and set theory
- Schema of result is explicit in expression
- Language of queries is same language that we use to prove properties: first order logic.