Relational Calculus

Chapter 4

Relational Calculus

- The logic underlying SQL and relational databases
- Declarative (non-procedural) – Describes answers without saying how to compute them
- Comes in two flavors (see textbook):
  - TRC: Tuple relational calculus and
  - DRC: Domain relational calculus
- Both are simple subsets of first-order logic
- Expressions in the calculus are called formulas. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to *true*.

$$\{T \mid T \in Loan \wedge T.amount > 1400\}$$

tuples.

Tuple Relational Calculus

- *Query* has the form: predicate of T

$$\{ T \mid p(T) \}$$

- *Answer* includes all tuples t for which the formula $p(T)$ evaluates to true when $T = t$

- *Formula* is recursively defined, starting with simple *atomic formulas* (getting tuples from relations or making comparisons of values), and building bigger and better formulas using the *logical connectives*.

TRC Formulas

- *Tuple variable:* takes on tuples of a relation as values
- *Atomic formula:*
  - $R \in Rname$, or $R.a$ *op* $S.b$, or $R.a$ *op* constant
  - *op* is one of $<, >, =, \leq, \geq, \neq$
- *Formula:*
  - an atomic formula, or
  - $\neg p, p \wedge q, p \vee q, p \Rightarrow q$, where p and q are formulas, or
  - $\exists X(p(X))$, where X is a tuple variable
  - $\forall X(p(X))$, where X is a tuple variable
  - The use of quantifiers $\exists X$ and $\forall X$ is said to *bind* X.
    - A variable that is not bound is free.

Free and Bound Variables

- The use of quantifiers $\exists X$ and $\forall X$ in a formula is said to *bind* X.
  - A variable that is not bound is free.
- Let us revisit the definition of a query:

$$\{ T \mid p(T) \}$$

- There is an important restriction: variable *T* that appears to the left of | must be the only free variable in the formulae p(…)

Setup

- Following Relations:
  - Loan(lid integer, amount number, bname string, due DATE);
  - Borrowers: Bw(cid integer, lid integer);
  - Depositors: Dp(cid integer, did integer);
  - DepositAccount(did integer, amount number);
  - Customer: Customer(cid integer, name STRING, …);

Find Loans larger than 1400

RA: $\sigma_{amount>1400}(Loan)$

RC: $\{\, T \mid \underbrace{T \in Loan}_{\text{set of tuples}} \wedge T.amount>1400 \,\}$

- Loan has an attribute amount.
- The condition *T* ∈ *Loan* ensures that the tuple variable *T* is bound to some tuple of *Loan*.
- The term *T* to the left of \`|' (which should be read as *such that*) says that every tuple that satisfies *T.amount*>1400 is in the answer.
- Modify this query to answer (assume bname field):
  - Find loans larger than 1400 that originated at branch "Redwood"

Find branch names with loans >1400

(only schema bname).

$$\pi_{bname}(\sigma_{amount>1400}(Loan))$$

T: result table (not from database).

goes away because it's bound.

P: from loan $\{ T \mid \exists P \in Loan\ (T.bname = P.bname \wedge P.amount>1400)\}$

tuples to return

tuples working with

- Because the only field of *T* that is mentioned is *bname* and *T* doesn't range over any of the relations in the query, *T* is a tuple with exactly one field: *bname*

Question??

$$\{ T \mid \exists P \in Loan\ (T.bname = P.bname \wedge P.amount > 1400)\}$$

C

In the formula above, T and P are respectively:

A. ~~Bound; Bound (I.e. both P and T and Bound)~~
B. ~~Bound; Unbound (I.e. T is Bound and P is Unbound)~~
C. Unbound; Bound (I.e. T is Unbound and P is Bound)
D. Unbound; Unbound (I.e. both P and T and Unbound)

Find all customers (cid) with an account and a loan

result table (with cid as only scheme)

$$\pi_{cid}\left(Dp \bowtie Bw\right)$$

$$\{ T \mid \exists R \in Dp\ \exists S \in Bw\ (T.cid = R.cid \wedge T.cid = S.cid) \}$$

- Note the use of $\exists$ to find a tuple in *Borrower* that `joins with' the *Depositor* tuple under consideration.

Find all customers (cid) with an account or a loan

$$\pi_{cid}(Dp) \cup \pi_{cid}(Bw)$$

$$\{ T \mid \exists R \in Dp\ (T.cid = R.cid) \vee \exists S \in Bw\ (T.cid = S.cid) \}$$

- Note that now we use two independent $\exists$ connected by an $\vee$. The tuple of *Depositor* is not linked to the tuple of *Borrower*

Unsafe Queries

- It is possible to write syntactically correct calculus queries that have an infinite number of answers! Such queries are called *unsafe*.
  - e.g., $\{T \mid \neg(T \in Loan)\}$

Expressive Power

- Codd's Theorem: Every RA query can be expressed as a safe query in relational calculus; the converse is also true.
- *Relational Completeness*: A "relationally complete" query language (e.g., SQL) can express every query that is expressible in relational algebra or safe query in relational calculus.

Summary

delarative ,

- The relational model: Two formal models (RA and RC)
- Relational algebra is more operational
  - Several ways of expressing a given query
  - a query optimizer should choose the most efficient version
- Relational calculus is non-operational
  - Users define queries in terms of what they want, not in terms of how to compute it (declarative)
- Algebra and safe relational calculus have same expressive power, leading to the notion of relational completeness