Relational calculus

Iztok Savnik, FAMNIT

Slides

- *Raghu Ramakrishnan, Johannes Gehrke, Database Management Systems, McGraw-Hill, 3rd ed., 2007.*
- *Slides from „Cow Book“: R.Ramakrishnan, http://pages.cs.wisc.edu/~dbbook/*

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*.
  - *DRC*: Variables range over *domain elements* (= field values).
  - Both TRC and DRC 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*.

Tuple Relational Calculus

- *Query* has the form: $\{ T \mid p(T) \}$ where T is the only free variable.
- *Answer* includes all tuples T that make the *formula* $p(T)$ be *true*.
- *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

- *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

  $\neg p$, $p \wedge q$, $p \vee q$, where p and q are formulas, or

  $\exists R(p(R))$, where tuple variable R is *free* in p(R), or

  $\forall R(p(R))$, where tuple variable R is *free* in p(R)

- The use of quantifiers $\exists R$ and $\forall R$ is said to *bind* R.

  A variable that is not bound is free.

Find the names and ages of saliors with a rating >7.

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

- P is considered to be a tuple variable with exactly two attributes.
  - P has attributes *name* and *age* since these were the only two attributes mentioned in the query.
  - Result of the query is the relation with two fields.
  - Formulas P.name=S.sname and P.age=S.age give value to P's fields.

Find the sailor name, boat id and reservation date for each reservation.

$$\{ P \mid \exists R \in Reserves\ \exists S \in Sailors\ (R.sid = S.sid \land P.name = S.sname \land P.bid = R.bid \land P.day = R.day) \}$$

- *Combinig values from more relations into answer tuples.*
- *For each reserves tuple, we look for the sailor with the same sid.*
  - The answer tuple P is constructed from the two tuples.

Find the names of sailors who have reserved boat #103.

$$\{ P \mid \exists S \in Sailors\ \exists R \in Reserves\ (R.sid = S.sid \wedge R.bid = 103 \wedge P.name = S.sname) \}$$

- *For each sailor tuple we find the corresponding reservation tuples for the boat #103.*
  - *The condition R.sid=S.sid defines a join!*
  - *The predicate R.bid=103 selects the reservations of a boat #103.*
  - *The names of selected sailors are copied to the result tuple P.*

Find the names of sailors who have reserved a red boat.

$$\{ P \mid \exists S \in Sailors\ \exists R \in Reserves\ (R.sid = S.sid \wedge P.name = S.sname \wedge \exists B \in Boats(B.bid = R.bid \wedge B.color = \text{'red'})) \}$$

- *Retrieve all sailors tuples for which there exists a reservation of a red boat.*
- *Alternative way of expressing the query.*

$$\{ P \mid \exists S \in Sailors\ \exists R \in Reserves\ \exists B \in Boats\ (R.sid = S.sid \wedge B.bid = R.bid \wedge P.name = S.sname \wedge B.color = \text{'red'}) \}$$

Find the names of sailors who have reserved at least two boats.

$$\{ P \mid \exists S{\in}Sailors\ \exists R1{\in}Reserves\ \exists R2{\in}Reserves\ (R1.sid{=}S.sid \wedge R2.sid{=}S.sid \wedge R1.bid{\neq}R2.bid \wedge P.name{=}S.sname) \}$$

- *We need two tuples from Reserves both joined with a tuple from Sailors.*
- *Tuples from Reserves are verified to be different ($R1.bid{\neq}R2.bid$).*

Find the names of sailors who have reserved all boats.

$$\{ P \mid \exists S \in Sailors\ \forall B \in Boats\ (\exists R \in Reserves(S.sid = R.sid \land R.bid = B.bid \land P.name = S.sname) \}$$

- *The query is very intuitive in TRC.*
- *English version: »find the sailors S such that for all boats B there exists a reservation such that sailor S has reserved boat B.«*
- *This query was expressed using the division operator in algebra.*

Domain Relational Calculus

- *Query* has the form:

$$\{\langle x1,x2,\ldots,xn\rangle \mid p(\langle x1,x2,\ldots,xn\rangle)\}$$

- *Answer* includes all tuples $\langle x1,x2,\ldots,xn\rangle$ that make the *formula* $p(\langle x1,x2,\ldots,xn\rangle)$ be *true*.

- *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*.

DRC Formulas

- *Atomic formula:*

  $\langle x1, x2, \ldots, xn \rangle \in Rname$, or X *op* Y, or X *op* constant

  op is one of $<, >, \leq, \geq, =, \neq$.
- *Formula:*

  an atomic formula, or

  $\neg p$, $p \wedge q$, $p \vee q$, where p and q are formulas, or

  $\exists X(p(X))$, where variable X is *free* in p(X), or

  $\forall X(p(X))$, where variable X is *free* in p(X)
- 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 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:

$$\{\langle x1,x2,\ldots,xn\rangle \mid p(\langle x1,x2,\ldots,xn\rangle)\}$$

- There is an important restriction: the variables x1, ..., xn that appear to the left of `|' must be the *only* free variables in the formula p(...).

Find all sailors with a rating above 7

$$\{\langle I,N,T,A\rangle \mid \langle I,N,T,A\rangle \in Sailors \wedge T>7\}$$

- The condition $\langle I,N,T,A\rangle \in Sailors$ ensures that the domain variables *I, N, T* and *A* are bound to fields of the same Sailors tuple.
- The term $\langle I,N,T,A\rangle$ to the left of `|' (which should be read as *such that*) says that every tuple $\langle I,N,T,A\rangle$ that satisfies $T>7$ is in the answer.
- Modify this query to answer:

  Find sailors who are older than 18 or have a rating under 9, and are called 'Joe'.

Find sailors rated > 7 who have reserved boat #103

$$\{\langle I,N,T,A\rangle \mid \langle I,N,T,A\rangle \in Sailors \wedge T>7 \wedge \exists Ir,Br,D(\langle Ir,Br,D\rangle \in Reserves \wedge Ir=I \wedge Br=103\}$$

- We have used $\exists Ir,Br,D$ as a shorthand for $\exists Ir(\exists Br(\exists D(\ldots)))$.
- Note the use of $\exists$ to find a tuple in Reserves that \`joins with' the Sailors tuple under consideration.

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

$$\{ \langle I,N,T,A \rangle \mid \langle I,N,T,A \rangle \in Sailors \land T>7 \land$$
$$\exists Ir,Br,D(\langle Ir,Br,D \rangle \in Reserves \land Ir=I \land$$
$$\exists B,BN,C(\langle B,BN,C \rangle \in Boats \land B=Br \land C='red' \}$$

- Observe how the parentheses control the scope of each quantifier's binding.
- This may look cumbersome, but with a good user interface, it is very intuitive. (MS Access, QBE)

Find sailors who've reserved all boats

$$\{ \langle I,N,T,A\rangle \mid \langle I,N,T,A\rangle \in Sailors \wedge$$
$$\forall B,BN,C(\neg(\langle B,BN,C\rangle \in Boats) \vee$$
$$(\exists Ir,Br,D(\langle Ir,Br,D\rangle \in Reserves \wedge I=Ir \wedge Br=B)))\}$$

- Find all sailors *I* such that for each 3-tuple ⟨B,BN,C⟩ either it is not a tuple in Boats or there is a tuple in Reserves showing that sailor *I* has reserved it.

Find sailors who've reserved all boats (again!)

$$\{ \langle I,N,T,A \rangle \mid \langle I,N,T,A \rangle \in Sailors \wedge$$
$$\forall \langle B,BN,C \rangle \in Boats$$
$$(\exists \langle Ir,Br,D \rangle \in Reserves(I=Ir \wedge Br=B))\}$$

- Simpler notation, same query. (Much clearer!)
- To find sailors who've reserved all red boats:

$$\ldots\ (C \neq \text{'red'} \vee \exists \langle Ir,Br,D \rangle \in Reserves(I=Ir \wedge Br=B))\}$$

Unsafe Queries, Expressive Power

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

Summary

- Relational calculus is non-operational, and users define queries in terms of what they want, not in terms of how to compute it. (Declarativeness.)
- Algebra and safe calculus have same expressive power, leading to the notion of relational completeness.