Relational Algebra and Calculus

Linda Wu

(CMPT 354 • 2004-2)

Topics

- Formal query languages
- Preliminaries
- Relational algebra
- Relational calculus
- Expressive power of algebra and calculus

Relational Query Languages

- Relational model supports simple, powerful query languages
  - Allow manipulation and retrieval of data from a database
  - Allow for much optimization
  - Strong formal foundation based on logic
- Query Languages ≠ programming languages
  - Query languages are not expected to be "Turing complete"
  - Query languages are not intended to be used for complex calculations
  - Query languages support easy, efficient access to large data sets

Formal Relational Query Languages

- Two mathematical query languages form the basis for "real" languages (e.g. SQL), and for implementation
  - Relational Algebra
    - Describe a step-by-step procedure for computing the desired answer
    - Operational, useful for representing execution plans
  - Relational Calculus
    - Describe the desired answer, rather than how to compute it
    - Non-operational, declarative

Preliminaries

- A query is applied to relation instances, and the result of a query is also a relation instance
  - Schemas of input relations for a query are fixed
  - The schema for the result of a given query is also fixed
- Positional vs. named-field notation
  - Positional notation is easier for formal definitions; named-field notation is more readable
  - Both are used in SQL

Relational Algebra

- Selection
- Projection
- Set operations
- Renaming
- Joins
- Division

Operators

- Basic operators
  - Selection (σ): select a subset of rows from relation
  - Projection (π): delete unwanted columns from relation
  - Cross-product ($\times$): combine two relations
  - Set-difference ($-$): tuples in relation 1 but not in relation 2
  - Union ($\cup$): tuples in both relation 1 and 2
- Additional operators
  - Intersection($\cap$), join($\bowtie$), division($/$), renaming(ρ)
  - Not essential, but very useful

Operators (Cont.)

- Each operator accepts relation instance(s) as arguments, and returns a relation instance as result
- Algebra expression
  - Composed by operators
  - Describe a procedure by which computing the desired answer
  - Used by relational systems to represent query evaluation plans

Example Instances

Boat (bid: integer, bname: string, color: string)
Sailors (sid: integer, sname: string, rating: integer, age: real)
Reserves (sid: integer, bid: integer, day: date)

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

Instance S1 of Sailors

sid	sname	rating	age
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Instance S2 of Sailors

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Instance R1 of Reserves

Projection π

- To delete attributes that are not in projection list
- Schema of result contains exactly the fields in the projection list, with the same names that they had in the single input relation
- Projection operator has to eliminate duplicates!

sname	rating
Yuppy	9
Lubber	8
guppy	5
Rusty	10

$\pi_{sname,rating}(S2)$

age
55.5
35.0

$\pi_{age}(S2)$

Selection σ

- To select rows that satisfy selection condition
- No duplicates in result
- Schema of result is identical to schema of single input relation
- Result relation can be the input for another relational algebra operation (operator composition)

sid	sname	rating	age
28	Yuppy	9	35.0
58	Rusty	10	35.0

$\sigma_{rating>8}(S2)$

sname	rating
Yuppy	9
Rusty	10

$\pi_{sname,rating}(\sigma_{rating>8}(S2))$

Selection σ (Cont.)

- Selection condition
  - A Boolean combination ($\wedge$,$\vee$) of terms
  - A term has the forms:
    - *attribute* op *constant, or,*
    - *attribute1* op *attribute2*
    - * op is one of: $<, \leq, =, \neq, \geq, >$
  - Example
    - (rating $\geq$ 8) $\vee$ (age $<$ 50)
    - (sid1 = sid2) $\wedge$ (bid1 $\neq$ bid2)

Union, Intersection, Set-Difference

- These 3 operators take 2 input relations, which must be union-compatible:
  - Have the same number of fields
  - Corresponding fields have the same types
- Result schema
  - The first relation

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0
44	guppy	5	35.0
28	Yuppy	9	35.0

$S1 \cup S2$

sid	*sname*	*rating*	*age*
31	Lubber	8	55.5
58	Rusty	10	35.0

$S1 \cap S2$

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0

$S1 - S2$

Cross-Product ×

- R × S = {<*r*, *s*> | *r* ∈ R, *s* ∈ S}
  - Each row of R is paired with each row of S
  - Result schema has one field per field of R and S, with field names inherited if possible
  - The result fields have the same domains as the corresponding fields in R and S
  - Naming conflict: R and S contain field(s) with the same name
    - The corresponding fields in R × S are unnamed and referred to only by position
    - E.g., both S1 and R1 have a field *sid*

Cross-Product × (Cont.)

(sid)	*sname*	*rating*	*age*	*(sid)*	*bid*	*day*
22	Dustin	7	45.0	22	101	10/10/96
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

$S1 \times R1$

Renaming ρ

- ρ *(R(F), E)*
  - *E: a relational algebra expression*
  - *R: a new relation*
  - *F: a list of fields that are renamed*
  - Takes *E* and returns an instance of *R*
  - *R* has the same tuples as the result of *E*
  - *R* has the same schema as *E*, but some fields are renamed

$$\rho\,(C(1 \to sid1, 5 \to sid2), S1 \times R1)$$
$$\rho\,(C, S1 \times R1)$$
$$\rho\,((1 \to sid1, 5 \to sid2), S1 \times R1)$$

Joins

- One of the most useful operations in relational algebra
- The most common way to combine information from two or more relations
- Defined as a cross-product followed by selections and projections
- Has a smaller result than a cross-product
- Condition join, equijoin, natural join, etc.

Joins (Cont.)

- Condition Join

$$R \bowtie_c S = \sigma_c (R \times S)$$

  - *C*: join condition
    may refer to the attributes of both *R* and *S*
  - Result schema is same as that of cross-product
  - Result has fewer tuples than cross-product; might be able to compute more efficiently

(sid)	*sname*	*rating*	*age*	*(sid)*	*bid*	*day*
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

Joins (Cont.)

- Equijoin: a special case of condition join where the condition *C* contains only equalities
  - Equality is of form: *R.name1 = S.name2*
  - Result schema is similar to cross-product, but only one copy of fields for which equality is specified
- Natural Join: equijoin on all common fields

sid	*sname*	*rating*	*age*	*bid*	*day*
22	Dustin	7	45.0	101	10/10/96
58	Rusty	10	35.0	103	11/12/96

$$S1 >< R1, \text{ or, } S1 ><_{S1.sid=R1.sid} R1$$

Division

- Not a primitive operator, but useful for expressing queries like:
  "Find sailors who have reserved all boats"
- Let A have 2 fields, x & y; B have only field y

$$A/B = \{ \langle x \rangle \mid \exists \langle x,y \rangle \in A \ \ \forall \langle Y \rangle \in B \}$$

  - i.e., A/B contains all x tuples (sailors) such that for every y tuple (boat) in B, there is an xy tuple in A (reserves), or,
  - if the set of y values (boats) associated with an x value (sailor) in A contains all y values in B, then x value is in A/B
- In general, x and y can be any lists of fields; y is the list of fields in B, and $x \cup y$ is the list of fields of A

Division (Cont.)

A

S#	P#
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

B1

P#
P2

B2

P#
P2
P4

B3

P#
P1
P2
P4

A/B1

S#
S1
S2
S3
S4

A/B2

S#
S1
S4

A/B3

S#
S1

Division (Cont.)

- Division is not an essential operation; just a useful shorthand
  - (Also true of joins, but joins are so common that systems implement joins specially)
- Expressing division using basic operators
  - Idea: for A/B, compute all x values that are not "disqualified" by some y value in B
  - x value is disqualified if: by attaching y value from B, we obtain an xy tuple that is not in A

Disqualified x values: $\pi_x((\pi_x(A) \times B) - A)$

A/B: $\pi_x(A)$ − all disqualified tuples

Examples

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

Solution 1: $\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$

Solution 2:
$\rho\ (Temp1,\ \sigma_{bid=103} Reserves)$
$\rho\ (Temp2,\ Temp1 \bowtie Sailors)$
$\pi_{sname}(Temp2)$

Solution 3: $\pi_{sname}(\sigma_{bid=103}(Reserves \bowtie Sailors))$

Examples (Cont.)

- Find the names of sailors who have reserved a red boat
  - Information about boat color is only available in Boats; so need an extra join with Boats

Solution 1:

$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie Sailors)$

Solution 2 (more efficient):

$\pi_{sname}(\pi_{sid}((\pi_{bid} \sigma_{color='red'} Boats) \bowtie Res) \bowtie Sailors)$

A query optimizer can find the second solution, given the first one!

Examples (Cont.)

- Find the names of sailors who have reserved a red or a green boat
  - Identify all red or green boats, then find sailors who have reserved one of these boats

$$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'}\ Boats))$$

$$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$$

- *Tempboats* can also be defined using union
- What if "∨" is replaced by "∧" in this query?

Examples (Cont.)

- Find the names of sailors who have reserved a red and a green boat
  - Identify sailors who have reserved red boats, sailors who have reserved green boats, and then, find the intersection
  - Note that *sid* is a key for Sailors

$$\rho\ (Tempred, \pi_{sid}((\sigma_{color='red'}\ Boats) \bowtie Reserves))$$

$$\rho\ (Tempgreen, \pi_{sid}((\sigma_{color='green'}\ Boats) \bowtie Reserves))$$

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

Examples (Cont.)

- Find the names of sailors who have reserved all boats
  - Uses division; schemas of the input relations to the division (/) must be carefully chosen

$$\rho\ (Tempsids, (\pi_{sid,bid} Reserves)\ /\ (\pi_{bid}\ Boats))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

- Find the names of sailors who have reserved all 'Interlake' boats

$$.....\ /\ \pi_{bid}(\sigma_{bname='Interlake'}\ Boats)$$

Summary

- The relational model has rigorously defined query languages that are simple and powerful
- Relational algebra is more operational; useful as internal representation for query evaluation plans
- There might be several ways of expressing a given query; a query optimizer should choose the most efficient version

Relational Calculus

- Domain relational calculus
- Tuple relational calculus

Relational Calculus

- Two variants of relational calculus
  - Tuple relational calculus (TRC): SQL
  - Domain relational calculus (DRC): QBE
- Calculus has variables, constants, comparison operators, logical connectives and quantifiers
  - TRC: variables range over tuples
  - DRC: variables range over domain elements (= field values)
  - Both TRC and DRC are simple subsets of first-order logic
- Calculus expressions are called formulas
- An answer tuple is an assignment of constants to variables that make the formula evaluate to true

Domain Relational Calculus

- DRC query has the form

  $\{ \langle x_1, x_2, \ldots, x_n \rangle \mid p (\langle x_1, x_2, \ldots, x_n \rangle) \}$

  - The answer to the query includes all tuples $\langle x_1, x_2, \ldots, x_n \rangle$ that make the formula $p (\langle x_1, x_2, \ldots, x_n \rangle)$ be true
  - DRC formula is recursively defined, starting with simple atomic formulas, and building bigger and better formulas using the logical connectives
  - Example: find all sailors with a rating above 7

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

Domain Relational Calculus (Cont.)

- DRC atomic formula
  - $\langle x_1, x_2, \ldots, x_n \rangle \in Rname$, or,
  - X *op* Y, or,
  - X *op constant*

  * $Rname$ is relation name; X, Y are domain variables; *op* is one of $<, >, =, \leq, \geq, \neq$
- DRC 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 $p(X)$

Domain Relational Calculus (Cont.)

- Free and bound variables
  - ∃ and ∀ are quantifiers
  - The use of $\exists X$ and $\forall X$ is said to bind X
  - A variable that is not bound is free
- An important restriction on the definition of a DRC query

$$\{ \langle x_1, x_2, \ldots, x_n \rangle \mid p (\langle x_1, x_2, \ldots, x_n \rangle) \}$$

  - The variables $x_1, x_2, \ldots, x_n$ that appear to the left of `|' must be the *only* free variables in the formula $p(\ldots)$

DRC Query Examples

- 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 ⟨I, N, T, A⟩ ∈ Sailors ensures that the domain variables I, N, T and A are bound to fields of the <u>same</u> Sailors tuple
  - "|" should be read as "*such that*"
  - The term ⟨I, N, T, A⟩ to the left of "|" says that every tuple ⟨I, N, T, A⟩ that satisfies T>7 is in the answer

DRC Query Examples (Cont.)

- Find the names of sailors rated > 7 who have reserved boat #103
  - $\exists Ir, Br, D$: a shorthand for $\exists Ir(\exists Br(\exists D()))$
  - ∃: to find a tuple in Reserves that "joins with" the Sailors tuple under consideration

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

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

DRC Query Examples (Cont.)

- Find sailors rated > 7 who have reserved a red boat
  - The parentheses control the scope of each quantifier's binding

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

$$\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in Sailors \wedge T > 7 \wedge \exists \langle I, Br, D \rangle \in Reserves \wedge \exists \langle Br, BN, \text{'red'} \rangle \in Boats \}$$

DRC Query Examples (Cont.)

- Find the names of sailors who have reserved all boats (solution 1)
  - Find all sailors ⟨I, N, T, A⟩ 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 B

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

DRC Query Examples (Cont.)

- Find the names of sailors who have reserved all boats (solution 2)
  - Simpler notation, same query (much clearer!)

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

- To find the names of sailors who jave reserved all red boats

$$\{ \langle N \rangle \mid \exists I, T, A \,(\langle I, N, T, A \rangle \in Sailors \wedge \forall \langle B, BN, C \rangle \in Boats \,(C \neq \text{'red'} \vee \exists \langle Ir, Br, D \rangle \in Reserves \,(Ir = I \wedge Br = B))) \}$$

Tuple Relational Calculus

- TRC query has the form

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

  - T is a tuple variable that takes on tuples of a relation as values
  - p(T) is a formula describing T
  - The answer to the query is the set of all tuples t that make p(T) be true when T = t
  - TRC formula is recursively defined
  - Example: find all sailors with a rating above 7

$$\{ S \mid S \in Sailors \wedge S.rating > 7 \}$$

Tuple Relational Calculus (Cont.)

- TRC atomic formula
  - $R \in Rname$, or,
  - $R.a$ *op* $S.b$, or,
  - $R.a$ *op* *constant*

    * *Rname* is relation name; *R*, S are tuple variables; *a* is an attribute of *R*, *b* is an attribute of *S*; *op* is one of $<, >, =, \leq, \geq, \neq$
- TRC 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 *R* is a tuple variable, or,
  - $\forall R\ (p(R))$, where *R* is a tuple variable

TRC Query Examples

- Find the names and ages of sailors with a rating above 7

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

  - P is a tuple variable with two fields: name and age
  - $P.name = S.sname$ and $P.age = S.age$ gives values to the fields of an answer tuple P
  - * If a variable *R* does not appear in an atomic formula of the form $R \in Rname$, the type of *R* is a tuple whose fields include all (and only) fields of *R* that appear in the formula

TRC Query Examples (Cont.)

- Find the names of sailors who have reserved all boats

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

- Find sailors who have reserved all red boats

$$\{ S \mid S \in Sailors \wedge \forall B \in Boats\ (B.color \neq 'red' \vee (\exists R \in Reserves\ (S.sid = R.sid \wedge R.bid = B.bid))) \}$$

Expressive Power of Algebra and Calculus

- Unsafe query
  - a syntactically correct calculus query that has an infinite number of answers
  - E.g., $\{ S \mid \neg (S \in Sailors) \}$
- 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
- In addition, commercial query languages can express some queries that cannot be expressed in relational algebra

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