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 ($\sigma$): select a subset of rows from relation
  - Projection ($\pi$): 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($\div$), renaming($\rho$)
  - 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

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

Instance S1 of Sailors

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

Instance S2 of Sailors

Projection \( \pi \)

- 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!

\[ \pi_{\text{name, rating}}(S) \]

Selection \( \sigma \)

- 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)

\[ \sigma_{\text{rating} \geq 8}(S_2) \]

Selection \( \sigma \) (Cont.)

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

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>28</td>
<td>Yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
</tbody>
</table>

Cross-Product \( \times \)

\( R \times S = \{ <r, s> | r \in R, s \in S \} \)
- 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 \( R \times S \) are unnamed and referred to only by position
  - E.g., both \( S1 \) and \( R1 \) have a field \( sid \)

Cross-Product \( \times \) (Cont.)

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

Renaming \( \rho \)

\( \rho (R(F), E) \)
- \( E \): 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

- \( \rho ( C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1 ) \)
- \( \rho ( C, S1 \times R1) \)
- \( \rho ( (1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1) \)
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

<table>
<thead>
<tr>
<th>(sid)</th>
<th>name</th>
<th>rating</th>
<th>age</th>
<th>(sid)</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
<td>103</td>
<td>11/12/96</td>
<td></td>
</tr>
</tbody>
</table>

\[ S_I \bowtie S_I.sid < R_I.sid R_I \]

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

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>rating</th>
<th>age</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

\[ S_I \bowtie R_I, \text{ or } S_I \bowtie_{S_I.sid=R_I.sid} R_I \]

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
  \[ A/B = \{ \langle x \rangle \mid \exists \langle x,y \rangle \in A \land \langle y \rangle \in B \} \]
  - 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.)

**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 $A$

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B1$</th>
<th>$B2$</th>
<th>$B3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S#</td>
<td>P#</td>
<td>P#</td>
<td>P#</td>
</tr>
<tr>
<td>S1</td>
<td>P1</td>
<td>P2</td>
<td>P1</td>
</tr>
<tr>
<td>S1</td>
<td>P2</td>
<td>P2</td>
<td>P2</td>
</tr>
<tr>
<td>S1</td>
<td>P3</td>
<td></td>
<td>P4</td>
</tr>
<tr>
<td>S1</td>
<td>P4</td>
<td></td>
<td>P4</td>
</tr>
</tbody>
</table>

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:** $\pi_{\text{name}} (\sigma_{\text{bid}=103} (\text{Reserves} \bowtie \text{Sailors}))$
  - **Solution 2:** $\rho (\text{Temp1}, \sigma_{\text{bid}=103} (\text{Reserves}))$
  - $\rho (\text{Temp2}, \text{Temp1} \bowtie \text{Sailors})$
  - $\pi_{\text{name}} (\text{Temp2})$
  - **Solution 3:** $\pi_{\text{name}} (\sigma_{\text{bid}=103} (\text{Reserves} \bowtie \text{Sailors}))$

### 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:** $\pi_{\text{name}} (\sigma_{\text{color}=\text{red}} (\text{Boats} \bowtie \text{Reserves} \bowtie \text{Sailors}))$
  - **Solution 2 (more efficient):** $\pi_{\text{name}} (\sigma_{\text{bid}} (\sigma_{\text{color}=\text{red}} (\text{Boats} \bowtie \text{Reserves} \bowtie \text{Sailors})))$
  - 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

\[ \rho (\text{Tempboats}, (\sigma_{\text{color} = \text{'red'}} \vee \text{color} = \text{'green'} \text{Boats})) \]
\[ \pi_{\text{sname}}(\text{Tempboats} \bowtie \text{Reserves} \bowtie \text{Sailors}) \]

- **Tempboats** can also be defined using union
- What if “\(\vee\)” is replaced by “\(\wedge\)” 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*

\[ \rho (\text{Tempred}, \pi_{\text{sid}}((\sigma_{\text{color} = \text{'red'}} \text{Boats}) \bowtie \text{Reserves})) \]
\[ \rho (\text{Tempgreen}, \pi_{\text{sid}}((\sigma_{\text{color} = \text{'green'}} \text{Boats}) \bowtie \text{Reserves})) \]
\[ \pi_{\text{sname}}((\text{Tempred} \cap \text{Tempgreen}) \bowtie \text{Sailors}) \]

---

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

\[ \rho (\text{Tempsids}, (\pi_{\text{sid, bid}} \text{Reserves}) / (\pi_{\text{bid}} \text{Boats})) \]
\[ \pi_{\text{sname}}(\text{Tempsids} \bowtie \text{Sailors}) \]

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

\[ \cdots / \pi_{\text{bid}}(\sigma_{\text{bname} = \text{'Interlake'}} \text{Boats}) \]

---

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

Domain Relational Calculus

- DRC query has the form
  \( \{ \langle x_1, x_2, \ldots, x_n \rangle \mid p \left( \langle x_1, x_2, \ldots, x_n \rangle \right) \} \)
  - The answer to the query includes all tuples \( \langle x_1, x_2, \ldots, x_n \rangle \) that make the formula \( p \left( \langle x_1, x_2, \ldots, x_n \rangle \right) \) 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
    \( \{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{Sailors} \land T > 7 \} \)

Domain Relational Calculus (Cont.)

- DRC atomic formula
  - \( \langle x_1, x_2, \ldots, x_n \rangle \in R_{name}, \ or, \)
  - \( X \text{ op } Y, \ or, \)
  - \( X \text{ op constant} \)
    - \( R_{name} \) is relation name; \( X, Y \) are domain variables;
    - \( \text{op} \) is one of \( <, >, =, \leq, \geq, \neq \)
- DRC formula
  - an atomic formula, or,
  - \( \neg p, p \land q, p \lor q, \) 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 ∃X and ∀X is said to bind X
  - A variable that is not bound is free
- An important restriction on the definition of a DRC query
  \[ \{ \langle x_1, x_2, ..., x_n \rangle \mid p(\langle x_1, x_2, ..., x_n \rangle) \} \]
  - The variables \( x_1, x_2, ..., x_n \) 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
  \[ \{ \langle I, N, T, A \rangle \mid (I, N, T, A) \in Sailors \land T > 7 \} \]
  - The condition \( (I, N, T, A) \in Sailors \) ensures that the domain variables I, N, T and A are bound to fields of the same 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

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

DRC Query Examples (Cont.)

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

DRC Query Examples (Cont.)

- Find sailors rated > 7 who have reserved a red boat
  - The parentheses control the scope of each quantifier’s binding
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 \(B\)

\[
\{ (N) | \exists I, T, A ( (I, N, T, A) \in Sailors \land \forall B, BN, C ( (B, BN, C) \in Boats ) \lor ( \exists (Ir, Br, D) \in Reserves ( Ir = I \land Br = B )) ) \} \]

DRC Query Examples (Cont.)

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

\[
\{ (N) | \exists I, T, A ( (I, N, T, A) \in Sailors \land \forall (B, BN, C) \in Boats ( C \neq \text{\textquoteleft red\textquoteright} ) \lor ( \exists (Ir, Br, D) \in Reserves ( Ir = I \land Br = B )) ) \} \]

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

\[
\{ (N) | \exists I, T, A ( (I, N, T, A) \in Sailors \land \forall (B, BN, C) \in Boats ( C = \text{\textquoteleft red\textquoteright} ) \lor ( \exists (Ir, Br, D) \in Reserves ( Ir = I \land Br = B )) ) \} \]

Tuple Relational Calculus

- TRC query has the form

\[
\{ T | p(T) \}
\]
  - \(T\) 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

\[
\{ S | S \in Sailors \land S\.rating > 7 \}
\]

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,
  - \(\neg p, p \land q, p \lor q\), 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

\[ \{ P | \exists S \in \text{Sailors} \ (S.rating > 7 \land P.name = S.sname \land P.age = S.age) \} \]

- \( P \) 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 \( R \in \text{Rname} \), 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

\[ \{ P | \exists S \in \text{Sailors} \ \forall B \in \text{Boats} \ (\exists R \in \text{Reserves} \ (S.sid = R.sid \land R.bid = B.bid \land P.sname = S.sname)) \} \]

- Find sailors who have reserved all red boats

\[ \{ S | S \in \text{Sailors} \land \forall B \in \text{Boats} \ (B.color \neq \text{‘red’} \lor \exists R \in \text{Reserves} \ (S.sid = R.sid \land R.bid = B.bid)) \} \]

Expressive Power of Algebra and Calculus

- Unsafe query
  - a syntactically correct calculus query that has an infinite number of answers
  - E.g., \( \{ S | \neg (S \in \text{Sailors}) \} \)
- 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