Relational Algebra

March 20, 2019
Announcements

• Today
  - Relational algebra, relational calculus

• Reading
  - Chapter 4 (you may skip 4.3.2)

• Acknowledgement
  - Some slide contents courtesy of Ramakrishnan and Gehrke

• Break around 4:15pm
Relational query languages

- Need a language to access relations in a database

- *Query languages*: allow manipulation and retrieval of data from a database

- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic
  - Allows for much optimization

- Query languages ≠ programming languages!
  - QLs are not expected to be “Turing complete”
  - QLs are not intended to be used for complex calculations
  - QLs support easy, efficient access to large datasets
Formal relational query languages

• Two formal query languages associated with the relational model form the basis for “real” languages (e.g., SQL), and for implementation:
  
  - Relational algebra: more operational, very useful for representing execution plans
  
  - Relational calculus: lets users describe what they want, rather than how to compute it (declarative)
Example

SQL:
SELECT S.name
FROM Students S, Enrolled E
WHERE S.gpa > 3.3 AND S.sid = E.sid AND E.cname = "CSE305";

RC:
{ P | ∃ S ∈ Sailors ∃ E ∈ Enrolled
(P.name = S.name ∧ S.gpa > 3.3 ∧ S.sid = E.sid ∧
E.cname = "CSE305") }

RA:
\[ \pi_{\text{name}}(\sigma_{S.gpa > 3.3}(\sigma_{E.cname = "CSE305"}(S \bowtie_{S.sid=E.sid} E))) \]
Relational algebra, relational calculus, SQL

• Relational algebra: procedural – ‘how’
• Relational calculus: declarative – ‘what’
• Relational algebra is equivalent in expressive power to relational calculus
• SQL is based on relational calculus
• Relational calculus is useful to define the semantics of the relational algebra and SQL
• Relational algebra is useful to compute SQL expressions which are translated into relational algebra expressions internally before they are optimized and evaluated
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
    - Determined by definition of query language constructs

- Positional vs. named-field notation:
  - Positional notation easier for formal definitions
  - Named-field notation more readable
  - Both used in SQL
Example schema

**Sailors** *(sid: integer, sname: string, rating: integer, age: real)*

**Boats** *(bid: integer, bname: string, color: string)*

**Reserves** *(sid: integer, bid: integer, day: date)*
Example instances

- Sailors and Reserves relations for our examples
- We’ll use positional or named field notation
- We’ll assume that names of fields in query results are ‘inherited’ from names of fields in query input relations

<table>
<thead>
<tr>
<th>R1 sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>3/10/18</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>4/12/18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S1 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>

<table>
<thead>
<tr>
<th>S2 sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>uppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Relational algebra

• Basic operations:
  - Selection (\(\sigma\)): selects a subset of rows from relation
  - Projection (\(\pi\)): extracts desired columns from relation
  - Cross-product (\(\times\)): allows us to combine two relations
  - Set-difference (\(\setminus\)): tuples in relation 1, but not in relation 2
  - Union (\(\cup\)): tuples in relation 1 or in relation 2 or in both

• Additional operations:
  - Intersection, join, division, renaming: not essential, but useful

• Since each operation returns a relation with input also relations, operations can be composed!
  - algebra is “closed”
Selection

- Selects rows that satisfy selection condition
- No duplicates in result! (Why?)
- Schema of result identical to schema of (only) input relation
- Result relation can be the input for another relational algebra operation! (Operator composition)

\[
\begin{align*}
\sigma & \text{ rating} > 8^{(S2)} \\
\pi & \text{name, rating}^{(\sigma \text{ rating} > 8^{(S2)})}
\end{align*}
\]

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
Projection

- Drops 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 (only) input relation
- Projection operator has to eliminate *duplicates*! (Why?)
  - Note: real systems typically don’t eliminate duplicates unless the user explicitly asks for it (Why not?)

<table>
<thead>
<tr>
<th>sname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>yuppy</td>
<td>9</td>
</tr>
<tr>
<td>lubber</td>
<td>8</td>
</tr>
<tr>
<td>guppy</td>
<td>5</td>
</tr>
<tr>
<td>rusty</td>
<td>10</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>35.0</td>
</tr>
<tr>
<td>55.5</td>
</tr>
</tbody>
</table>

$$\pi_{age}(S2)$$
Union, intersection, set-difference

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields
  - Corresponding fields have the same type
- What is the *schema* of result?

<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>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>28</td>
<td>yuppie</td>
<td>9</td>
<td>35.0</td>
</tr>
</tbody>
</table>

$S_1 \cup S_2$

$S_1 - S_2$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<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>

$S_1 \cap S_2$
Cross-product

- Each row of S1 is paired with each row of R1
- **Result schema** has one field per field of S1 and R1, with field names ‘inherited’ if possible
  - **Naming conflicts**: Both S1 and R1 have a field called *sid*

<table>
<thead>
<tr>
<th>(sid)</th>
<th>sname</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>22</td>
<td>101</td>
<td>3/10/18</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>4/12/18</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>3/10/18</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>4/12/18</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>3/10/18</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>4/12/18</td>
</tr>
</tbody>
</table>

- **Renaming operator**:  \( \rho (C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1) \)
Joins

1. **Condition join**: \( R \bowtie_c S = \sigma_c (R \times S) \)

<table>
<thead>
<tr>
<th>(sid)</th>
<th>sname</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>4/12/18</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>4/12/18</td>
</tr>
</tbody>
</table>

\( S1 \bowtie S1.sid < R1.sid R1 \)

- **Result schema** same as that of cross-product
- Fewer tuples than cross-product, might be able to compute more efficiently
- Also called a **theta-join**
2. **Equi-join:** A special case of condition join where the condition \( c \) contains only **equalities**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</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>3/10/18</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>103</td>
<td>4/12/18</td>
</tr>
</tbody>
</table>

\[ S_1 \bowtie_{\text{sid}} R_1 \]

**Result schema** similar to cross-product, but only one copy of fields for which equality is specified

3. **Natural join:** Equi-join on *all* common fields (omit the condition in this case)

If two relations have no attributes in common, natural join is same as what?
Example schema and instances

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

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>3/10/18</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>4/12/18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Clipper</td>
<td>blue</td>
</tr>
<tr>
<td>103</td>
<td>Sinker</td>
<td>red</td>
</tr>
</tbody>
</table>
Find names of sailors who’ve reserved boat #103.

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

• **Solution 2:** 
  \[ \rho (\text{Temp1}, \sigma_{bid=103} \text{Reserves}) \]
  \[ \rho (\text{Temp2}, \text{Temp1} \bowtie \text{Sailors}) \]
  \[ \pi_{sname}(\text{Temp2}) \]

• **Solution 3:** 
  \[ \pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie \text{Sailors})) \]

What are the differences – is one better than the others?
Find names of sailors who’ve reserved a red boat.

- Information about boat color only available in Boats; so need an extra join:

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

- Another solution:

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

- A query optimizer can find a better one given a bad one!
- Rewrite these using the rename operator \( \rho \).
More examples

• Textbook has more examples (Q1 through Q10)
  – Study them

• Each illustrates some concept(s) that is (are) important to understand
Relational algebra, relational calculus, and SQL

Find all sailors with a rating above 8.

- Relational algebra (over sets)
  \( \sigma_{\text{rating} > 8} (\text{Sailors}) \)

- Tuple relational calculus
  \( \{ S \mid S \in \text{Sailors} \land S.\text{rating} > 8 \} \)

- SQL (over bags, i.e., multisets)
  
  ```sql
  SELECT *
  FROM Sailors S
  WHERE S.rating > 8
  ```
Tuple Relational Calculus (TRC)

- A tuple variable takes on tuples of a relation schema as values

- A TRC query has the form \( \{ T \mid p(T) \} \), where
  - \( T \) is a tuple variable and
  - \( p(T) \) denotes a formula that describes \( T \)

  The result of this query is the set of all tuples \( t \) for which the formula \( p(T) \) evaluates to \textit{true} with \( T=t \).

- For example,
  \[ \{ S \mid S \in \text{Sailors} \land S.\text{rating} > 8 \} \]
  finds all sailors with a rating above 8.
Another example (same as earlier)

SQL:  SELECT S.name
       FROM Students S, Enrolled E
       WHERE S.gpa>3.3 AND S.sid=E.sid AND E.cname="CSE305";

RC:  { P | ∃ S ∈ Sailors ∃ E ∈ Enrolled
      (P. name = S. name ∧ S. gpa > 3.3 ∧ S. sid = E. sid ∧
       E. cname = “CSE305” )}

RA:

$$\pi_{name}(\sigma_{S.gpa>3.3}(\sigma_{E.cname="CSE305"}(S \bowtie S.sid=E.sid E)))$$
SQL queries and relational algebra

SELECT $L$
FROM $R$
WHERE $C$

Here $L$ is a list of expressions, $R$ is a list of relations, and $C$ is a condition. The meaning of any such SQL expression is the same as that of the relational algebra expression:

$$\pi_L(\sigma_C(R))$$

That is, we start with the relation in the FROM clause, apply to each tuple whatever condition is indicated in the WHERE clause, and then project onto the list of attributes and/or expressions in the SELECT clause.
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

• Several ways of expressing a given query; a query optimizer should choose the most (?) efficient version