Relational Algebra
Relational Query Languages

• Query languages enable the manipulation and retrieval of data from a database
• The relational model supports simple, powerful query languages:
  • With a strong formal foundation based on logic, for clear semantics
  • Amenable to query optimization, for efficiency
• Query languages are not programming languages
  • Not expected to be “Turing complete”
  • Not intended for complex calculations
  • Designed to support easy, efficient access to large data sets

Formal Relational Query Languages

Two mathematical query languages form the basis for “real-world” query languages (notably, SQL) and their implementation:

• **Relational Algebra**: Somewhat “operational,” useful for representing execution plans, as we will see towards the end of the semester

• **Relational Calculus**: “Declarative”; lets users describe what they want, rather than how to compute it (not covered in our class)

Understanding relational algebra is key to understanding SQL and, later, query processing
Relational Algebra: Preliminaries

• A relational algebra query is applied to relation instances; the result of a query is also a relation instance
  • Schemas of input relations for a query are clearly specified
  • Schema for the result of each query is also clearly defined, unambiguously determined by the definition of each relational algebra construct

• We can refer to attributes in a query using “positional” notation or the attribute names
  • Positional notation is often easier in formal definitions, as we will see; named-attribute notation is more readable
  • Both options are used also in SQL

Relational Algebra: Preliminaries

• A relation instance in relational algebra is **always a set of tuples, with no duplicate rows**
• This is true also for query results (which are also relations): think of conceptually “eliminating duplicates” automatically whenever they arise
Relations for Examples

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

Reserves keeps track of which sailors reserved which boats and when

Example Instances in Textbook

- S1 and S2 are two instances of Sailors
- R1 is an instance of Reserves
- Boats also has an instance, not shown here

<table>
<thead>
<tr>
<th>S1</th>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>22</td>
<td>dustin</td>
<td></td>
<td>45.0</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>lubber</td>
<td></td>
<td>55.5</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>rusty</td>
<td></td>
<td>35.0</td>
</tr>
<tr>
<td>S2</td>
<td>sid</td>
<td>sname</td>
<td>rating</td>
<td>age</td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>uppy</td>
<td></td>
<td>35.0</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>lubber</td>
<td></td>
<td>55.5</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>guppy</td>
<td></td>
<td>35.0</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>rusty</td>
<td></td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R1</th>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Relational Algebra: Operations

- Operations that remove parts of a relation
  - $\Pi$ (projection): unary operator that keeps only the wanted columns of the input relation instance
  - $\sigma$ (selection): unary operator that selects a subset of the rows of the input relation instance

- Set operations, applied to two relations
  - $U$ (union): binary operator that returns all the tuples that are in at least one of the two input relation instances
  - $\cap$ (intersection): binary operator that returns the tuples that are in both input relation instances
  - $-$ (set difference): binary operator that returns all the tuples in the first relation instance that do not appear in the second relation instance

Relational Algebra: Operations (cont.)

- Operations that combine the tuples of two relations
  - $X$ (cross-product): binary operator that returns the cross product of the two input relation instances
  - $\bowtie$ (join): binary operator that combines the tuples of both input relations in powerful ways

- Operation to change the schema of a relation
  - $\rho$ (renaming): operator to rename a relation and/or its attributes

Additionally, we will discuss $/$ (division), a binary operator with complex (and useful) semantics.

Since each operation returns a relation, operations can be composed to form complex expressions.
**Projection** Removes Attributes Not In Projection List

- Schema of result contains exactly the attributes in the projection list, in that order and with the same names that they had in the schema of the input relation.
- The output of the projection operator, as always in relational algebra, has no duplicate tuples.

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

\[ \pi_{\text{name},\text{rating}}(S2) \]

**Selection** Removes Rows Not Satisfying Selection Condition

- Schema of result is identical to schema of input relation.
- Operator composition: Result relation can be input to another relational algebra operation.

<table>
<thead>
<tr>
<th>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>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{rating} > 8}(S2) \]

\[ \pi_{\text{name},\text{rating}}(\sigma_{\text{rating} > 8}(S2)) \]
Union, Intersection, Set Difference

The two input relations must be union-compatible:
- Same number of attributes
- "Corresponding" attributes have same type (not necessarily same name, though)

Schema of result is the schema of the left relation

<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>uppy</td>
<td>9</td>
<td>35.0</td>
</tr>
</tbody>
</table>

$S_1 \cup S_2$

<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>

$S_1 \setminus S_2$

Cross Product
- Each row of first input relation instance is paired with each row of second input relation instance
  - If first relation has $n$ tuples and second relation has $m$ tuples, their cross product has $n*m$ tuples
- Schema of result has all attributes of first relation followed by all attributes of second relation, in that order
  - If first relation has $i$ attributes and the second relation has $j$ attributes, the cross product has $i+j$ attributes
  - If the input relations have attributes with the same name, problem: we cannot have repeated attribute names in the output relation
  - Such attributes with conflicting names are left unnamed in the output relation and hence can only be referred to using positional notation
Cross Product Example: S1xR1

- Each row of S1 is paired with each row of R1
- Result schema has one attribute per attribute of S1 and R1, with attribute names “inherited” if possible
- Both S1 and R1 have an attribute called sid, so those attributes are left unnamed in the result:

<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>10/10/96</td>
</tr>
<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>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</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>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

Renaming Operator

Specifies that the output relation should have a different name for the relation and/or some of its attributes. For example:

\( \rho(N(1 \rightarrow a, 3 \rightarrow b, 4 \rightarrow c), T) \)

specifies that the output should be a relation \( N \) such that:

- \( N \) contains the same exact tuples as input relation \( T \)
- \( N \) shares the schema with \( T \), with the following exceptions:
  - The name of the output relation is \( N \), not \( T \)
  - The names of the first, third, and fourth attributes of \( N \) are \( a \), \( b \), and \( c \), respectively, not the names these attributes had in \( T \)

If we just want to change the name of the output relation, we can omit the attribute renaming portion: \( \rho(N, T) \); analogously, if we just want to rename the attributes, we can omit the new name for the output relation: \( \rho((1 \rightarrow a, 3 \rightarrow b, 4 \rightarrow c), T) \)
Cross Product Example $S1 \times R1$ Revisited

<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>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<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>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</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>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

We can now assign names to the first and fifth attributes (which come from sid from S1 and R1, respectively, and are unnamed in the cross product):

$$\rho \left( C(1 \rightarrow \text{sid}1, 5 \rightarrow \text{sid}2), S1 \times R1 \right)$$

The output of this expression is a relation with the 6 tuples above and schema:

$$C(\text{sid}1, \text{name}, \text{rating}, \text{age}, \text{sid}2, \text{bid}, \text{day})$$

Condition (or “Theta”) Join $R \bowtie_{\text{cond}} S$

$R \bowtie_{\text{cond}} S$ is, by definition, shorthand for $\sigma_{\text{cond}}(R \times S)$:

- The schema for $R \bowtie_{\text{cond}} S$ is the schema for $\sigma_{\text{cond}}(R \times S)$, which in turn is the schema for $R \times S$
- The tuples in $R \bowtie_{\text{cond}} S$ are the tuples in $R \times S$ that satisfy condition $\text{cond}$

$$S1 \bowtie_{\text{cond}} S1.\text{sid} < R1.\text{sid} \quad R1$$

<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>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
An equijoin $R \bowtie_{eq\_cond} S$ is a special case of condition join where the condition $eq\_cond$ is a conjunction of equality conditions, each involving one attribute of $R$ and one attribute of $S$

- The schema for $R \bowtie_{eq\_cond} S$ is equal to that of the corresponding condition join, but where we only keep the left attribute—with its name and in its place—in each equality condition
- For example, the schema of $S1 \bowtie_{S1.sid=R1.sid} R1$ is $(sid, sname, rating, age, bid, day)$

Whenever all equality conditions in the equijoin involve attributes that are named the same in both input relations, we can simply list the attributes as a subscript of the join symbol and omit the equality signs
- So we can write $S1 \bowtie_{S1.sid=R1.sid} R1$ more simply as $S1 \bowtie_{sid} R1$
Equijoin \( R \bowtie_{eq\_cond} S \)

<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>
</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>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

\( S1 \bowtie_{sid} R1 \)

<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>

Natural Join \( R \bowtie S \)

- A special (and most frequent) case of equijoin in which an equality condition is requested for each pair of attributes named the same in \( R \) and \( S \)
- Natural joins don’t have any subscript: all equalities as above, based on the actual names of the attributes across \( R \) and \( S \), are assumed to be in the (implicit) equijoin condition
- In our example, \( S1 \bowtie R1 \) is equivalent to equijoin \( S1 \bowtie_{sid} R1 \) (in turn equivalent to \( S1 \bowtie_{S1.sid=R1.sid} R1 \))
Division A(x, y)/B(y)

- Useful shorthand for expressing queries such as:
  “Find sailors who have reserved all boats”
- We assume for now that A has attributes x and y, and B has
  one attribute, y, that matches the domain of A.y
  - A/B is defined as \{\langle x \rangle | \exists \langle x, y \rangle \in A \land \langle y \rangle \in B}\}
  - In other words, A/B contains all the values of x in A such
    that, for every y tuple in B, there is a tuple <x, y> in A

- In general, A can have more than two attributes and B can
  have more than one attribute, as long as A=(x_1, x_2, ..., x_m, y_1,
  y_2, ..., y_m) and B=(y_1, y_2, ..., y_m) and the y_i attributes in A and
  B have respectively matching domains; then x_1, x_2, ..., x_m play
  the role of x, and y_1, y_2, ..., y_m play the role of y in the
  definition above

Examples of Division A/B

<table>
<thead>
<tr>
<th>sno</th>
<th>pno</th>
<th>pno</th>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p1</td>
<td></td>
<td>p2</td>
</tr>
<tr>
<td>s2</td>
<td>p1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>p2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td>p2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td>p4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A/B1

B1

B2

B3

A

A/B1

A/B2

A/B3
Expressing A/B Using Basic Operators

- Like joins, division is just a useful shorthand and can be expressed in terms of basic relational algebra operators
- To express A/B in terms of basic operators, let’s first compute all x values that are “disqualified” from the division output by some y value in B:

An x value is disqualified if by “attaching” a y value from B we obtain an <x, y> tuple that is not in A
Disqualified x values: $\pi_X((\pi_X(A) \times B) - A)$

- Then, A/B = $\pi_X(A) - \text{disqualified x values}$

Find names of sailors who have 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: $\pi_{sname}(\sigma_{bid=103} (\text{Reserves} \bowtie \text{Sailors}))$
Find names of sailors who have reserved a red boat

- Sailors(sid, sname, rating, age)
- Boats(bid, bname, color)
- Reserves(sid, bid, day)

Boat color only available in Boats, so need an extra join:

$$\pi_{sname}(\sigma_{\text{color}=\text{\textquoteleft red\textquoteright	extquoteleft}} \text{Boats} ) \bowtie \text{Reserves} \bowtie \text{Sailors} )$$

An alternative solution:

$$\pi_{\text{sname}}(\pi_{\text{sid}}(\pi_{\text{bid}}(\sigma_{\text{color}=\text{\textquoteleft red\textquoteright	extquoteleft}} \text{Boats} ) ) \bowtie \text{Reserves} ) \bowtie \text{Sailors} )$$

Find names of sailors who have reserved a red or a green boat

Can identify all red or green boats, then find sailors who have reserved one of these boats:

$$\rho\left(\text{Tempboats},(\sigma_{\text{color}=\text{\textquoteleft red\textquoteright	extquoteleft} \lor \text{color}=\text{\textquoteleft green\textquoteright	extquoteleft}} \text{Boats})\right) \pi_{\text{sname}}(\text{Tempboats} \bowtie \text{Reserves} \bowtie \text{Sailors})$$

Can also define Tempboats using union (how?)

What happens if we replace “or” with “and” in this query?
Find names of sailors who have reserved both a red and a green boat

Previous approach won’t work: must identify sailors who have reserved red boats, sailors who have reserved green boats, and then find the intersection:

\[
\rho (\text{Tempred}, \pi_{\text{sid}} ((\sigma_{\text{color} = \text{red}} \ Boats) \bowtie \text{Reserves}))
\]

\[
\rho (\text{Tempgreen}, \pi_{\text{sid}} ((\sigma_{\text{color} = \text{green}} \ Boats) \bowtie \text{Reserves}))
\]

\[
\pi_{\text{sname}}((\text{Tempred} \cap \text{Tempgreen}) \bowtie \text{Sailors})
\]

Find the names of sailors who have reserved all boats

Use division, choosing the schemas of the input relations carefully to be able to apply division properly

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

\[
\pi_{\text{sname}}(\text{Tempsids} \bowtie \text{Sailors})
\]

To find sailors who have reserved all “Interlake” boats:

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