ICS 321 Data Storage & Retrieval

The Relational Model of Data (i)

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa
Data Models

A data model is a collection of concepts for describing data

• Structure of the data.
  – More of a conceptual model rather than a physical data model. Eg. Arrays, objects in C/C++

• Operations on the data
  – Queries and modifications only

• Constraints on the data
  – Limitations on the data. Eg. Data type etc.

Examples: the relational model and the semi-structured model (XML)
The Relational Model

• **Relational database**: a set of *relations*

• **A relation** is made up of 2 parts:
  - *Instance*: a *table*, with rows and columns.  
    
    #Rows = *cardinality*, #fields = *degree / arity*.
  
  - *Schema*: specifies name of relation, plus name and *domain/type* of each column or attribute.
    
    • E.G. Students(sid: string, name: string, login: string, age: integer, gpa: real).

• Can think of a relation as a *set* of rows or *tuples* (i.e., all rows are distinct).
Example Instance of Students Relation

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.4</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@eeecs</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.8</td>
</tr>
</tbody>
</table>

- Cardinality = 3, degree=5, all rows distinct
- Do all columns in a relation instance have to be distinct?
Relational Query Languages

• The relational model supports simple, powerful *querying* of data.
• Queries are written declaratively in SQL, and the DBMS finds an efficient execution plan.
• Query Languages != programming languages!
• Two mathematical query languages
  – *Relational Algebra*: More operational, useful for representing query execution plans.
  – *Relational Calculus*: More declarative
Preliminaries

• A query takes *relation instances* as input and outputs a relation instance.

• Positional vs. named-field notation:
  – Named-field notation more readable.
  – Both used in SQL
  – Field names in query results are ‘inherited’ from input relations

• “Sailors” and “Reserves” relations for our examples.
Relational Algebra

• Basic operations:
  – **Selection** \((\sigma)\) Selects a subset of rows from relation.
  – **Projection** \((\pi)\) Deletes unwanted columns from relation.
  – **Cross-product** \((\times)\) Allows us to combine two relations.
  – **Set-difference** \((-)\) Tuples in reln. 1, but not in reln. 2.
  – **Union** \((U)\) Tuples in reln. 1 and in reln. 2.

• Additional operations:
  – Intersection, **join**, division, renaming: Not essential, but (very!) useful.

• Since each operation returns a relation, operations can be **composed**! (Algebra is “closed”.)
Projection

• Deletes 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 do duplicate elimination unless the user explicitly asks for it. (Why not?)
Selection

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

\[
\begin{array}{|c|c|c|c|}
\hline
\text{sid} & \text{sname} & \text{rating} & \text{age} \\
\hline
28 & Yuppy & 9 & 35.0 \\
31 & Lubber & 8 & 55.5 \\
44 & Guppy & 5 & 35.0 \\
58 & Rusty & 10 & 35.0 \\
\hline
\end{array}
\]

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

\[
\begin{array}{|c|c|c|}
\hline
\text{sid} & \text{sname} & \text{rating} \\
\hline
28 & Yuppy & 9 \\
31 & Lubber & 8 \\
44 & Guppy & 5 \\
58 & Rusty & 10 \\
\hline
\end{array}
\]

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

\[
\begin{array}{|c|c|c|}
\hline
\text{sid} & \text{sname} & \text{rating} \\
\hline
28 & Yuppy & 9 \\
31 & Lubber & 8 \\
44 & Guppy & 5 \\
58 & Rusty & 10 \\
\hline
\end{array}
\]
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>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>7</td>
<td>45.0</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S2</th>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>28</td>
<td>Yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>Guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td></td>
<td>58</td>
<td>Rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

S1 U S2

<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>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>
### Intersection & Set-Difference

#### S1 ∩ S2

<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>

#### S1 - S2

<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>
</tbody>
</table>

#### S1

<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>

#### S2

<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>
Cross-Product

- Consider the cross product of S1 with R1
- 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.
  - **Conflict:** Both S1 and R1 have a field called *sid*.
  - Rename to *sid1* and *sid2*

<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>

<table>
<thead>
<tr>
<th>S1</th>
<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></td>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td></td>
<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></td>
<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></td>
<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></td>
<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

- The expression:
  \[ \rho \left( C \left( 1 \rightarrow \text{sid1}, 5 \rightarrow \text{sid2} \right), S1 \times R1 \right) \]
- Renames the result of the cross product of S1 and R1 to “C”
- Renames column 1 to sid1 and column 5 to sid2

\[ \rho \left( C \left( 1 \rightarrow \text{sid1}, 5 \rightarrow \text{sid2} \right), S1 \times R1 \right) \]

<table>
<thead>
<tr>
<th>sid1</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>sid2</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45</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</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>
Joins

- **Condition Join**: \( R \bowtie_c S = \sigma_c (R \times S) \)
- **Result schema** same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a **theta-join**.

\[ S_1 \bowtie_{S_1.sid < R_1.sid} R_1 \]

<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</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>
Equi-Joins & Natural Joins

• **Equi-join**: A special case of condition join where the condition \( c \) contains only *equalities*.  
  – **Result schema** similar to cross-product, but only one copy of fields for which equality is specified.

• **Natural Join**: Equi-join on *all* common fields.

\[ S_1 \bowtie_{sid} R_1 \]

<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</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>
Find names of sailors who’ve reserved boat #103

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

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

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

- What **tables/relations** are needed to answer the query?
  - What columns are needed?
  - Which tables do they belong to?
- How should the tables be **linked together**?
  - Joins, cross-product etc
- What columns are needed in the **final output**?
  - Projection operator
- What **filtering conditions** are needed?
  - Selection operator
Find names of sailors who’ve reserved a red boat

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

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

• A more efficient solution:

\[ \pi_{\text{name}} (\pi_{\text{sid}} ((\pi_{\text{bid}} \sigma_{\text{color} = 'red'} \text{Boats}) \bowtie \text{Res} \bowtie \text{Sailors}) \]
Find sailors who’ve reserved a red or a green boat

• First find all red or green boats, then find sailors who’ve reserved one of these boats:

\[
\rho \left( Tempboats, (\sigma \text{color} = \text{'red'} \lor \text{color} = \text{'green'} \ Boats) \right) \\
\pi \text{sname}(Tempboats \bowtie Reserves \bowtie Sailors)
\]

• Can also define Tempboats using union! (How?)
• What happens if \( \lor \) is replaced by \( \land \) in this query?
Find sailors who’ve reserved a red and a green boat

• Previous approach won’t work! Must identify sailors who’ve reserved red boats, sailors who’ve reserved green boats, then find the intersection (note that sid is a key for Sailors):

\[
\rho \left( \text{Tempred}, \pi_{\text{sid}}((\sigma_{\text{color}='\text{red}' \text{ Boats}}) \Join \text{Reserves}) \right)
\]

\[
\rho \left( \text{Tempgreen}, \pi_{\text{sid}}((\sigma_{\text{color}='\text{green} \text{ Boats}}) \Join \text{Reserves}) \right)
\]

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

• The Relational Data Model
• Two theoretical relational query languages: relational algebra & relational calculus
• Relational Algebra (RA) operators: selection, projection, cross-product, set difference, union, intersection, join, division, renaming
• Operators are closed and can be composed
• RA is more operational and could be used as internal representation for query evaluation plans.
• For the same query, the RA expression is not unique.
• Query optimizer can choose the most efficient version.