CompSci 516
Data Intensive Computing Systems

Lecture 4
Relational Algebra
and
Relational Calculus

Instructor: Sudeepa Roy
Announcements

• HW1 reminder:
  – Due on 09/21 (Thurs), 11:55 pm, no late days

• Project proposal deadline extended:
  – Preliminary idea and team members due by 09/18 (Mon) by email to the instructor
  – Proposal due on sakai by 09/25 (Mon), 11:55 pm
  – One report per group
  – Form your group soon
  – Look at recent research papers/demonstrations in top db conferences for ideas (see the project idea file on sakai)

• Your piazza, sakai, gradiance accounts should be active
  – Occasional Pop up quizzes will start from next week
  – Bring a laptop in class
Today’s topics

• Finish NULLs and Views in SQL from Lecture 3
• Relational Algebra (RA) and Relational Calculus (RC)
• Reading material
  – [RG] Chapter 4 (RA, RC)
  – [GUW] Chapters 2.4, 5.1, 5.2

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.
Nulls and Views in SQL
Null Values

• Field values in a tuple are sometimes
  – unknown, e.g., a rating has not been assigned, or
  – inapplicable, e.g., no spouse’s name
  – SQL provides a special value null for such situations.
Standard Boolean 2-valued logic

- True = 1, False = 0
- Suppose X = 5
  - (X < 100) AND (X >= 1) is \( T \land T = T \)
  - (X > 100) OR (X >= 1) is \( F \lor T = T \)
  - (X > 100) AND (X >= 1) is \( F \land T = F \)
  - NOT(X = 5) is \( \neg T = F \)

- Intuitively,
  - \( T = 1, F = 0 \)
  - For \( V_1, V_2 \in \{1, 0\} \)
    - \( V_1 \land V_2 = \text{MIN}(V_1, V_2) \)
    - \( V_1 \lor V_2 = \text{MAX}(V_1, V_2) \)
    - \( \neg(V_1) = 1 - V_1 \)
2-valued logic does not work for nulls

• Suppose rating = null, X = 5
• Is rating > 8 true or false?
• What about AND, OR and NOT connectives?
  – (rating > 8) AND (X = 5)?
• What if we have such a condition in the WHERE clause?
3-Valued Logic For Null

- TRUE (= 1), FALSE (= 0), UNKNOWN (= 0.5)
  - unknown is treated as 0.5

- Now you can apply rules from 2-valued logic!
  - For $V_1, V_2 \in \{1, 0, 0.5\}$
  - $V_1 \land V_2 = \min(V_1, V_2)$
  - $V_1 \lor V_2 = \max(V_1, V_2)$
  - $\neg(V_1) = 1 - V_1$

- Therefore,
  - NOT UNKNOWN = UNKNOWN
  - UNKNOWN OR TRUE = TRUE
  - UNKNOWN AND TRUE = UNKNOWN
  - UNKNOWN AND FALSE = FALSE
  - UNKNOWN OR FALSE = UNKNOWN
New issues for Null

• The presence of null complicates many issues. E.g.:
  – Special operators needed to check if value IS/IS NOT NULL
  – Be careful!
  – “WHERE X = NULL” does not work!
  – Need to write “WHERE X IS NULL”

• Meaning of constructs must be defined carefully
  – e.g., WHERE clause eliminates rows that don’t evaluate to true
  – So not only FALSE, but UNKNOWNs are eliminated too
  – very important to remember!

• But NULL allows new operators (e.g. outer joins)

• Arithmetic with NULL
  – all of +, -, *, / return null if any argument is null

• Can force ”no nulls” while creating a table
  – sname char(20) NOT NULL
  – primary key is always not null
Aggregates with NULL

<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</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?
Aggregates with NULL

- What do you get for
  - SELECT count(*) from R1?
  - SELECT count(rating) from R1?
- Ans: 3 for both

<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</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

R1
Aggregates with NULL

R1

<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</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

R2

<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</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>null</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

• What do you get for
  • SELECT count(*) from R1?
  • SELECT count(rating) from R1?
  • Ans: 3 for both

• What do you get for
  • SELECT count(*) from R2?
  • SELECT count(rating) from R2?
Aggregates with NULL

### R1

<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</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

- What do you get for
- SELECT count(*) from R1?
- SELECT count(rating) from R1?
- Ans: 3 for both

### R2

<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</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>null</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

- What do you get for
- SELECT count(*) from R2?
- SELECT count(rating) from R2?
- Ans: First 3, then 2
**Aggregates with NULL**

- **COUNT, SUM, AVG, MIN, MAX (with or without DISTINCT)**
  - Discards null values first
  - Then applies the aggregate
  - Except count(*)
- If only applied to null values, the result is null

<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</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>null</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

**R2**
- SELECT sum(rating) from R2?
- Ans: 17

<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>null</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>null</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>null</td>
<td>35</td>
</tr>
</tbody>
</table>

**R3**
- SELECT sum(rating) from R3?
- Ans: null
Overview: General Constraints

- Useful when more general ICs than keys are involved

- There are also ASSERTIONS to specify constraints that span across multiple tables

- There are TRIGGERS too: procedure that starts automatically if specified changes occur to the DBMS
  - see additional slides at the end

CREATE TABLE Sailors
  ( sid INTEGER,
    sname CHAR(10),
    rating INTEGER,
    age REAL,
    PRIMARY KEY (sid),
    CHECK (rating >= 1 AND rating <= 10)
  )

CREATE TABLE Reserves
  ( sname CHAR(10),
    bid INTEGER,
    day DATE,
    PRIMARY KEY (bid, day),
    CONSTRAINT noInterlakeRes
      CHECK ("Interlake" <>
          (SELECT B.bname
            FROM Boats B
            WHERE B.bid=bid)))
Views

• A view is just a relation, but we store a definition, rather than a set of tuples

```
CREATE VIEW YoungActiveStudents (name, grade)
AS SELECT S.name, E.grade
FROM Students S, Enrolled E
WHERE S.sid = E.sid and S.age<21
```

• Views can be dropped using the DROP VIEW command

• Views and Security: Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s)
  • the above view hides courses “cid” from E
Can create a new table from a query on other tables too

```
SELECT... INTO.... FROM.... WHERE
```

```
SELECT S.name, E.grade
INTO YoungActiveStudents
FROM  Students S, Enrolled E
WHERE  S.sid = E.sid and S.age < 21
```
“WITH” clause – very useful!

• You will find “WITH” clause very useful!

  WITH Temp1 AS
    (SELECT .... ..),
  Temp2 AS
    (SELECT .... ..)
SELECT X, Y
FROM TEMP1, TEMP2
WHERE....

• Can simplify complex nested queries
Summary

• SQL has a huge number of constructs and possibilities
  – You need to learn and practice it on your own
  – Given a problem, you should be able to write a SQL query and verify whether a given one is correct

• Pay attention to NULLs

• Can limit answers using “LIMIT” or “TOP” clauses
  – e.g. to output TOP 20 results according to an aggregate
  – also can sort using ASC or DESC keywords
Relational Query Languages
Relational Query Languages

• **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 not intended to be used for complex calculations
  – QLs 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: More operational, very useful for representing execution plans
  – Relational Calculus: Lets users describe what they want, rather than how to compute it (Non-operational, declarative, or procedural)

• Note: Declarative (RC, SQL) vs. Operational (RA)
• 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
    • query will run regardless of instance
  – 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
### Example Schema and Instances

Sailors($sid$, $sname$, rating, age)

Boats($bid$, $bname$, color)

Reserves($sid$, $bid$, day)

<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>

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

- \( \exists \) There exists
- \( \forall \) For all
- \( \wedge \) Logical AND
- \( \vee \) Logical OR
- \( \neg \) NOT
Relational Algebra (RA)
Relational Algebra

• Takes one or more relations as input, and produces a relation as output
  – operator
  – operand
  – semantic
  – so an algebra!

• Since each operation returns a relation, operations can be composed
  – Algebra is “closed”
Relational Algebra

• Basic operations:
  – Selection (σ) Selects a subset of rows from relation
  – Projection (π) Deletes unwanted columns from relation.
  – Cross-product (x) Allows us to combine two relations.
  – Set-difference (-) Tuples in reln. 1, but not in reln. 2.
  – Union (∪) Tuples in reln. 1 or in reln. 2.

• Additional operations:
  – Intersection (∩)
  – join ⋈
  – division(/)
  – renaming (ρ)
  – Not essential, but (very) useful.
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 (performance)
Selection

- Selects rows that satisfy selection condition
- No duplicates in result. Why?
- Schema of result identical to schema of (only) input relation

<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>

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

<table>
<thead>
<tr>
<th>sname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>yuppy</td>
<td>9</td>
</tr>
<tr>
<td>rusty</td>
<td>10</td>
</tr>
</tbody>
</table>

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

- Result relation can be the input for another relational algebra operation
  - Operator composition

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

\[
\pi_{\text{snname, rating}}(\sigma_{\text{rating} > 8}(S2))
\]
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
  - Same schema as the inputs
### Union, Intersection, Set-Difference

<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>

<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>

- **Note:** no duplicate
  - “Set semantic”
  - SQL: **UNION**
  - SQL allows “bag semantic” as well: **UNION ALL**

<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>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
</tbody>
</table>
## Union, Intersection, Set-Difference

### $S_1$

<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>

### $S_2$

<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>

### $S_1 - S_2$

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

### $S_1 \cap S_2$

<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>
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.
  - **Conflict:** 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>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 $\rho$

$$(\rho_{\text{sid} \rightarrow \text{sid}_1} \text{ } S1) \times (\rho_{\text{sid} \rightarrow \text{sid}_1} \text{ } R1)$$

or

$$\rho(C(1 \rightarrow \text{sid}_1, 5 \rightarrow \text{sid}_2), \text{ } S1 \times R1)$$

<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>

- In general, can use $\rho(<\text{Temp}>, <\text{RA-expression}>)$

C is the new relation name
Joins

\[ R \bowtie_c S = \sigma_c (R \times S) \]

<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>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>

\[ S1 \bowtie S1.sid < R1.sid \quad R1 \]

- Result schema same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
Find names of sailors who’ve reserved boat #103

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)
Find names of sailors who’ve reserved boat #103

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

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

• Solution 2: \( \pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie \text{Sailors})) \)
Expressing an RA expression as a Tree

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

Also called a logical query plan

\[ \pi_{sname} ( (\sigma_{bid=103} \text{Reserves}) \bowtie \text{Sailors} ) \]
Find sailors who’ve reserved a red or a green boat

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

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

\[ \rho (Tempboats, (\sigma_{\text{color} = 'red' \lor \text{color} = 'green'} Boats)) \]

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

Can also define Tempboats using union
Try the “AND” version yourself
What about aggregates?

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

- Extended relational algebra
- $\forall_{age, \text{avg}\text{rating}} \rightarrow \text{avgr} \text{Sailors}$
- Also extended to “bag semantic”: allow duplicates
  - Take into account cardinality
  - $R$ and $S$ have tuple $t$ resp. $m$ and $n$ times
  - $R \cup S$ has $t$ $m+n$ times
  - $R \cap S$ has $t$ $\min(m, n)$ times
  - $R - S$ has $t$ $\max(0, m-n)$ times
  - sorting($\tau$), duplicate removal ($\delta$) operators
Relational Calculus (RC)
Relational Calculus

- RA is procedural
  - $\pi_A(\sigma_{A=a} \ R)$ and $\sigma_{A=a} (\pi_A \ R)$ are equivalent but different expressions

- RC
  - non-procedural and declarative
  - describes a set of answers without being explicit about how they should be computed

- TRC (tuple relational calculus)
  - variables take tuples as values
  - we will primarily do TRC

- DRC (domain relational calculus)
  - variables range over field values
TRC: example

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

• Find the name and age of all sailors with a rating above 7

\{P \mid \exists S \in \text{Sailors} \ (S.\text{rating} > 7 \land P.\text{sname} = S.\text{sname} \land P.\text{age} = S.\text{age})\}

• P is a tuple variable
  – with exactly two fields sname and age (schema of the output relation)
  – P.\text{sname} = S.\text{sname} \land P.\text{age} = S.\text{age} gives values to the fields of an answer tuple

• Use parentheses, \( \forall \ \exists \ \lor \ \land \ \geq \ \leq \ = \ \neq \ \neg \) etc as necessary

• A \implies B is very useful too
  – next slide
A \implies B

- A “implies” B
- Equivalently, if A is true, B must be true
- Equivalently, \neg A \lor B, i.e.
  - either A is false (then B can be anything)
  - otherwise (i.e. A is true) B must be true
Useful Logical Equivalences

• \( \forall x \ P(x) = \neg \exists x \ [\neg P(x)] \)

• \( \neg (P \lor Q) = \neg P \land \neg Q \)
• \( \neg (P \land Q) = \neg P \lor \neg Q \)

— Similarly, \( \neg (\neg P \lor Q) = P \land \neg Q \) etc.

• \( A \implies B = \neg A \lor B \)
TRC: example

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

• Find the names of sailors who have reserved at least two boats
TRC: example

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

• Find the names of sailors who have reserved at least two boats

\{P \mid \exists S \in \text{Sailors} (\exists R1 \in \text{Reserves} \exists R2 \in \text{Reserves} (S.\text{sid} = R1.\text{sid} \land S.\text{sid} = R2.\text{sid} \land R1.\text{bid} \neq R2.\text{bid}) \land P.\text{fname} = S.\text{sname})\}
TRC: example

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

• Find the names of sailors who have reserved all boats
• Called the “Division” operation
TRC: example

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

• Find the names of sailors who have reserved all boats
• Division operation in RA!

\{P \mid \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\}
TRC: example

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

• Find the names of sailors who have reserved all red boats

How will you change the previous TRC expression?
TRC: example

Sailors\((\text{sid}, \text{sname}, \text{rating}, \text{age})\)
Boats\((\text{bid}, \text{bname}, \text{color})\)
Reserves\((\text{sid}, \text{bid}, \text{day})\)

• Find the names of sailors who have reserved all red boats
{P \mid \exists S \in \text{Sailors} (\forall B \in \text{Boats} (B.\text{color} = \text{‘red’} \Rightarrow (\exists R \in \text{Reserves} (S.\text{sid} = R.\text{sid} \land R.\text{bid} = B.\text{bid})))) \land P.\text{pname} = S.\text{sname})}

Recall that \(A \Rightarrow B\) is logically equivalent to \(\neg A \lor B\)
so \(\Rightarrow\) can be avoided, but it is cleaner and more intuitive