CS 405G: Introduction to Database Systems

Relational Algebra
Review

- Database
  - Relation schemas, relation instances and relational constraints.

- What’s next?
  - Relational query language.

- Reading
  - Chapter 6.1~6.5
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 $\neq$ programming languages!
  - QLs not intended to be used for complex calculations and inference (e.g. logical reasoning)
  - 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-procedural, declarative.)

✉ Understanding Algebra & Calculus is key to understanding SQL, query processing!
Relational algebra

A language for querying relational databases based on operators:

- Core set of operators:
  - Selection, projection, cross product, union, difference, and renaming
- Additional, derived operators:
  - Join, natural join, intersection, etc.
- Compose operators to make complex queries
Input: a table $R$
Notation: $\sigma_p R$
- $p$ is called a selection condition/predicate
Purpose: filter rows according to some criteria
Output: same columns as $R$, but only rows of $R$ that satisfy $p$
Selection example

- Students with GPA higher than 3.0

\[ \sigma_{GPA > 3.0} \text{ Student} \]
More on selection

- Selection predicate in general can include any column of $R$, constants, comparisons ($=, \cdot, \text{etc.}$), and Boolean connectives ($\land$: and, $\lor$: or, and $\lnot$: not)
  - Example: straight A students under 18 or over 21
    $$\sigma_{GPA = 4.0 \land (\text{age} < 18 \lor \text{age} > 21)} \text{Student}$$

- But you must be able to evaluate the predicate over a single row of the input table
  - Example: student with the highest GPA
    $$\sigma_{GPA \geq \text{all GPA in Student table}} \text{Student}$$
Input: a table $R$
Notation: $\pi_L \ R$
$L$ is a list of columns in $R$
Purpose: select columns to output
Output: same rows, but only the columns in $L$
Order of the rows is preserved
Number of rows may be less (depends on where we have duplicates or not)
Projection example

- ID’s and names of all students

\[ \pi_{SID, \text{name}} \text{Student} \]

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>22</td>
<td>3.8</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>2.6</td>
</tr>
<tr>
<td>1204</td>
<td>Susan Wong</td>
<td>22</td>
<td>3.4</td>
</tr>
<tr>
<td>1306</td>
<td>Kevin Kim</td>
<td>21</td>
<td>2.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
</tr>
<tr>
<td>1204</td>
<td>Susan Wong</td>
</tr>
<tr>
<td>1306</td>
<td>Kevin Kim</td>
</tr>
</tbody>
</table>
More on projection

- Duplicate output rows are removed (by definition)

- Example: student ages

\[
\pi_{\text{age}} \text{Student}
\]

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>22</td>
<td>3.8</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>2.6</td>
</tr>
<tr>
<td>1204</td>
<td>Susan Wong</td>
<td>22</td>
<td>3.4</td>
</tr>
<tr>
<td>1306</td>
<td>Kevin Kim</td>
<td>21</td>
<td>2.9</td>
</tr>
</tbody>
</table>

\[
\pi_{\text{age}}
\]

\[
\begin{array}{c}
\text{age} \\
21 \\
22
\end{array}
\]
Cross product

- Input: two tables $R$ and $S$
- Notation: $R \times S$
- Purpose: pairs rows from two tables
- Output: for each row $r$ in $R$ and each row $s$ in $S$, output a row $rs$ (concatenation of $r$ and $s$)
Cross product example

- **Student X Enroll**

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>22</td>
<td>3.8</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>2.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
<td>1234</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>22</td>
<td>3.8</td>
<td>1234</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>2.6</td>
<td>1234</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>22</td>
<td>3.8</td>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>2.6</td>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
</tbody>
</table>
A note on column ordering

- The ordering of columns in a table is considered unimportant (as is the ordering of rows)

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>22</td>
<td>3.8</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>2.6</td>
</tr>
</tbody>
</table>

That means cross product is commutative, i.e., \( R \times S = S \times R \) for any \( R \) and \( S \)
Derived operator: join

- Input: two tables $R$ and $S$
- Notation: $R \Join_p S$
  - $p$ is called a join condition/predicate
- Purpose: relate rows from two tables according to some criteria
- Output: for each row $r$ in $R$ and each row $s$ in $S$, output a row $rs$ if $r$ and $s$ satisfy $p$

- Shorthand for $\sigma_p ( R \times S )$
Join example

- Info about students, plus CID’s of their courses

\[
\text{Student} \Join_{\text{Student.SID} = \text{Enroll.SID}} \text{Enroll}
\]

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>22</td>
<td>3.8</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>2.6</td>
</tr>
</tbody>
</table>

Use \textit{table\_name.column\_name} syntax to disambiguate identically named columns from different input tables.
Derived operator: natural join

- Input: two tables $R$ and $S$
- Notation: $R* S$
- Purpose: relate rows from two tables, and
  - Enforce equality on all common attributes
  - Eliminate one copy of common attributes
- Shorthand for $\pi_L ( R \bowtie_p S )$, where
  - $p$ equates all attributes common to $R$ and $S$
  - $L$ is the union of all attributes from $R$ and $S$, with duplicate attributes removed
Natural join example

- $\text{Student} \ast \text{Enroll} = \pi_L (\text{Student} \bowtie_p \text{Enroll})$

  $= \pi_{SID, \text{name}, \text{age}, \text{GPA}, \text{CID}} (\text{Student} \mid_{\text{Student.SID} = \text{Enroll.SID}} \text{Enroll})$

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>22</td>
<td>3.8</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>22</td>
<td>2.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>22</td>
<td>3.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>108</td>
<td>A</td>
</tr>
</tbody>
</table>
Union

- **Input:** two tables $R$ and $S$

- **Notation:** $R \cup S$
  - $R$ and $S$ must have identical schema

- **Output:**
  - Has the same schema as $R$ and $S$
  - Contains all rows in $R$ and all rows in $S$, with duplicate rows eliminated
Difference

- Input: two tables $R$ and $S$
- Notation: $R - S$
  - $R$ and $S$ must have identical schema
- Output:
  - Has the same schema as $R$ and $S$
  - Contains all rows in $R$ that are not found in $S$
Derived operator: intersection

- **Input:** two tables $R$ and $S$
- **Notation:** $R \setminus S$
  - $R$ and $S$ must have identical schema
- **Output:**
  - Has the same schema as $R$ and $S$
  - Contains all rows that are in both $R$ and $S$
- **Shorthand for** $R - (R - S)$
- **Also equivalent to** $S - (S - R)$
- **And to** $R \ast S$
Renaming

- Input: a table $R$
- Notation: $\rho_S R$, $\rho_{(A_1, A_2, \ldots)} R$ or $\rho_{S(A_1, A_2, \ldots)} R$
- Purpose: rename a table and/or its columns
- Output: a renamed table with the same rows as $R$
- Used to
  - Avoid confusion caused by identical column names
  - Create identical columns names for natural joins
Renaming Example

\[ \rho_{\text{Enroll}}(\text{SID1, CID1, Grade1}) \] Enroll

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid1</th>
<th>cid1</th>
<th>grade1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
</tbody>
</table>
Review: Summary of core operators

- Selection: $\sigma_p R$
- Projection: $\pi_L R$
- Cross product: $R \times S$
- Union: $R \cup S$
- Difference: $R - S$
- Renaming: $\rho_{S(A_1, A_2, \ldots)} R$

Does not really add "processing" power
Review Summary of derived operators

- Join: \( R \bowtie_p S \)
- Natural join: \( R \ast S \)
- Intersection: \( R \cap S \)

- Many more
  - Outer join, Division,
  - Semijoin, anti-semijoin, …
Using Join

- Which classes is Lisa taking?
  - Student(sid: string, name: string, gpa: float)
  - Course(cid: string, department: string)
  - Enrolled(sid: string, cid: string, grade: character)

- An Answer:
  - Student_Lisa ← σ\_name = "Lisa" Student
  - Lisa_Enrolled ← Student_Lisa * Enrolled
  - Lisa’s classes ← π\_CID Lisa_Enrolled

- Or:
  - Student_Enrolled ← Student * Enrolled
  - Lisa_Enrolled ← σ\_name = "Lisa" Student_Enrolled
  - Lisa’s classes ← π\_CID Lisa_Enrolled
Join Example

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John</td>
<td>21</td>
<td>3.5</td>
</tr>
<tr>
<td>1123</td>
<td>Mary</td>
<td>22</td>
<td>3.8</td>
</tr>
<tr>
<td>1012</td>
<td>Lisa</td>
<td>22</td>
<td>2.6</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{name} = "Lisa"}(\text{students}) \times \pi_{\text{cid}}(\text{grades}) \]

\[ \begin{array}{c|c|c|c|c|}
\hline
\text{sid} & \text{name} & \text{age} & \text{gpa} & \text{cid} & \text{grade} \\
\hline
1012 & Lisa & 22 & 2.6 & 647 & A \\
1012 & Lisa & 22 & 2.6 & 108 & B \\
\hline
\end{array} \]
Lisa’s Class

\[ \pi_{CID} \left( (\sigma_{name = \text{“Lisa”}} \text{Student}) \ast \text{Enrolled} \right) \]

Who’s Lisa?

\[ \sigma_{name = \text{“Lisa”}} \]

Enroll

Student

Lisa’s classes \( \pi_{CID} \)
Students in Lisa’s Classes

- SID of Students in Lisa’s classes
  - \( \text{Student}_{Lisa} \leftarrow \sigma_{name = "Lisa"} \text{Student} \)
  - \( \text{Lisa}_{Enrolled} \leftarrow \text{Student}_{Lisa} \ast \text{Enrolled} \)
  - \( \text{Lisa’s classes} \leftarrow \pi_{CID} \text{Lisa}_{Enrolled} \)
  - Enrollment in Lisa’s classes \(\leftarrow \text{Lisa’s classes} \ast \text{Enrolled}\)
  - Students in Lisa’s class \(\leftarrow \pi_{SID} \text{Enrollment in Lisa’s classes} \)

Who’s Lisa?

\( \sigma_{name = "Lisa"} \text{Student} \)

Enroll
Tips in Relational Algebra

- Use temporary variables
- Use foreign keys to join tables
An exercise

- Names of students in Lisa’s classes

Their names \( \pi_{name} \)

Students in Lisa’s classes \( \pi_{SID} \)

Lisa’s classes \( \pi_{CID} \)

Who’s Lisa? \( \sigma_{name = “Lisa”} \)

Enroll

\( \ast \)
Set Minus Operation

- CID’s of the courses that Lisa is NOT taking

\[ \pi_{CID} \left( \text{Course} \right) \]

\[ - \]

\[ \pi_{CID} \left( \text{CID’s of the courses that Lisa IS taking} \right) \]

\[ \sigma_{name = "Lisa"} \]

\[ \ast \]

\[ \pi_{CID} \left( \text{Enroll} \right) \]

\[ \text{Student} \]
Renaming Operation

\( \rho_{Enrolled1}(SID1, CID1, Grade1) \) Enrolled

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
</tbody>
</table>

\( \rho_{Enroll1}(SID1, CID1, Grade1) \)

<table>
<thead>
<tr>
<th>sid1</th>
<th>cid1</th>
<th>grade1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
</tbody>
</table>
Example

- We have the following relational schemas
  - Student(sid: string, name: string, gpa: float)
  - Course(cid: string, department: string)
  - Enrolled(sid: string, cid: string, grade: character)
- SID’s of students who take at least two courses

\[ \pi_{\text{SID}} (\text{Enrolled} \bowtie \text{Enrolled}) \]

\[ \text{Enrolled}.\text{SID} = \text{Enrolled}.\text{SID} \land \text{Enrolled}.\text{CID} \neq \text{Enrolled}.\text{CID} \]
Example (cont.)

\[ \rho_{Enroll_1}(SID_1, CID_1, \text{Grade}1) \text{ Enrolled} \]
\[ \rho_{Enroll_2}(SID_2, CID_2, \text{Grade}2) \text{ Enrolled} \]
\[ \pi_{SID} (Enroll_1 \Join_{SID_1 = SID_2 \& CID_1 \neq CID_2} Enroll_2) \]

Expression tree syntax:

\[ \pi_{SID_1} \Join_{SID_1 = SID_2 \& CID_1 \neq CID_2} \]

\[ \rho_{Enroll_1}(SID_1, CID_1, \text{Grade}1) \]
\[ \rho_{Enroll_2}(SID_2, CID_2, \text{Grade}2) \]
How does it work?

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
<tr>
<td>1012</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1012</td>
<td>108</td>
<td>B</td>
</tr>
</tbody>
</table>

$\text{Enroll1} \times_{\text{SID}1 = \text{SID}2} \text{Enroll2}$

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
<th>sid2</th>
<th>cid2</th>
<th>grade2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1123</td>
<td>108</td>
<td>A</td>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
<tr>
<td>1012</td>
<td>647</td>
<td>A</td>
<td>1012</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1012</td>
<td>647</td>
<td>A</td>
<td>1012</td>
<td>108</td>
<td>B</td>
</tr>
<tr>
<td>1012</td>
<td>108</td>
<td>B</td>
<td>1012</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1012</td>
<td>108</td>
<td>B</td>
<td>1012</td>
<td>108</td>
<td>B</td>
</tr>
<tr>
<td>sid</td>
<td>cid</td>
<td>grade</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1123</td>
<td>108</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1012</td>
<td>647</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1012</td>
<td>108</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$Enroll1 \neq Enroll2$

$SID1 = SID2 \& CID1 \neq CID2$

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
<tr>
<td>1012</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1012</td>
<td>108</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid2</th>
<th>cid2</th>
<th>grade2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1123</td>
<td>108</td>
<td>A</td>
</tr>
<tr>
<td>1012</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1012</td>
<td>108</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1012</td>
<td>647</td>
<td>A</td>
</tr>
<tr>
<td>1012</td>
<td>108</td>
<td>B</td>
</tr>
<tr>
<td>1012</td>
<td>108</td>
<td>B</td>
</tr>
</tbody>
</table>
A trickier exercise

- Who has the highest GPA?
  - Who has a GPA?
  - Who does NOT have the highest GPA?
    - Whose GPA is lower than somebody else’s?

A deeper question:
When (and why) is “-” needed?
Tips in Relational Algebra

- A comparison is to identify a relationship
Review: Summary of core operators

- Selection: \( \sigma_p R \)
- Projection: \( \pi_L R \)
- Cross product: \( R \times S \)
- Union: \( R \cup S \)
- Difference: \( R - S \)
- Renaming: \( \rho_{S(A_1, A_2, \ldots)} R \)

- Does not really add “processing” power
Review: Summary of derived operators

- Join: \( R \bowtie_p S \)
- Natural join: \( R \ast S \)
- Intersection: \( R \cap S \)
Review

- Relational algebra
  - Use temporary variable
  - Use foreign key to join relations
  - A comparison is to identify a relationship
## Exercises of R. A.

### Reserves

<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>

### Boats

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>Blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>Red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>Green</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>Red</td>
</tr>
</tbody>
</table>

### Sailors

<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>
Problem 1 Find names of sailors who’ve reserved boat #103

Solution:

\[ \pi_{sname}((\sigma_{bid=103} \text{Reserves})^*\text{Sailors}) \]

Who reserved boat #103?

Boat #103

\[ \sigma_{bid = "103"} \text{Reserves} \]
Problem 2: Find names of sailors who’ve reserved a red boat

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

\[
\begin{align*}
\text{Names of sailors who reserved red boat} & \quad \pi_{sname} \\
\text{Who reserved red boats?} & \quad \pi_{SID} \\
\text{Red boats} & \quad \sigma_{color = \text{“red”}} \\
\text{Boat} & \quad \ast \\
\text{Reserve} & \quad \ast \\
\text{Sailors} & 
\end{align*}
\]
Problem 3: Find names of sailors who’ve reserved a red boat or a green boat

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

  Names of sailors who reserved red boat \( \pi_{sname} \)

  Who reserved red boats?

  Red boats

  \( \sigma_{\text{color} = \text{"red"}} \lor \sigma_{\text{color} = \text{"green"}} \)
Problem 4: Find names of sailors who’ve reserved only one boat.
Monotone operators

- If some old output rows may need to be removed
  - Then the operator is non-monotone
- Otherwise the operator is monotone
  - That is, old output rows always remain “correct” when more rows are added to the input
- Formally, for a monotone operator $op$: $R \cup R'$ implies $op(R) \cup op(R')$

Add more rows to the input...

What happens to the output?
Why is “-” needed for highest GPA?

- Composition of monotone operators produces a monotone query
  - Old output rows remain “correct” when more rows are added to the input

- Highest-GPA query is non-monotone
  - Current highest GPA is 4.1
  - Add another GPA 4.2
  - Old answer is invalidated

☞ So it must use difference!
Classification of relational operators

- Selection: $\sigma_p R$  Monotone
- Projection: $\pi_L R$  Monotone
- Cross product: $R \times S$  Monotone
- Join: $R \bowtie_p S$  Monotone
- Natural join: $R \ast S$  Monotone
- Union: $R \cup S$  Monotone
- Difference: $R - S$  Monotone w.r.t. $R$; non-monotone w.r.t $S$
- Intersection: $R \cap S$  Monotone
Why do we need core operator \( X \)?

- Cross product
  - The only operator that adds columns
- Difference
  - The only non-monotone operator
- Union
  - The only operator that allows you to add rows?
- Selection? Projection?
Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.
  
  - `avg`: average value
  - `min`: minimum value
  - `max`: maximum value
  - `sum`: sum of values
  - `count`: number of values

- **Aggregate operation** in relational algebra
  
  \[ G_1, G_2, \ldots, G_n \{ F_1(A_1), F_2(A_2), \ldots, F_n(A_n) \} (E) \]

  - `E` is any relational-algebra expression
  - \( G_1, G_2, \ldots, G_n \) is a list of attributes on which to group (can be empty)
  - Each \( F_i \) is an aggregate function
  - Each \( A_i \) is an attribute name
Aggregate Operation – Example

- Relation $r$:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$\alpha$</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

$g_{\text{sum}(c)}(r)$

$$\text{sum-C} = 27$$
Aggregate Operation – Example

- Relation *account* grouped by *branch-name*:

<table>
<thead>
<tr>
<th>branch-name</th>
<th>account-number</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perryridge</td>
<td>A-102</td>
<td>400</td>
</tr>
<tr>
<td>Perryridge</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>750</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-215</td>
<td>750</td>
</tr>
<tr>
<td>Redwood</td>
<td>A-222</td>
<td>700</td>
</tr>
</tbody>
</table>

\[
\text{branch-name} \ g \ \text{sum(balance)} \ (\text{account})
\]

<table>
<thead>
<tr>
<th>branch-name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perryridge</td>
<td>1300</td>
</tr>
<tr>
<td>Brighton</td>
<td>1500</td>
</tr>
<tr>
<td>Redwood</td>
<td>700</td>
</tr>
</tbody>
</table>
Null Values

- It is possible for tuples to have a null value, denoted by `null`, for some of their attributes.
- `null` signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving `null` is `null`.
- Aggregate functions simply ignore null values.
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same.
Null Values

- Comparisons with null values return the special truth value *unknown*
  - If *false* was used instead of *unknown*, then \( \text{not} (A < 5) \) would not be equivalent to \( A \geq 5 \)

- Three-valued logic using the truth value *unknown*:
  - OR: \((\text{unknown or true}) = \text{true}\),
    \((\text{unknown or false}) = \text{unknown}\)
    \((\text{unknown or unknown}) = \text{unknown}\)
  - AND: \((\text{true and unknown}) = \text{unknown}\),
    \((\text{false and unknown}) = \text{false}\),
    \((\text{unknown and unknown}) = \text{unknown}\)
  - NOT: \((\text{not unknown}) = \text{unknown}\)

- Result of select predicate is treated as *false* if it evaluates to *unknown*
Additional Operators

- Outer join
- Division
(Left) Outer Join

- **Input:** two tables $R$ and $S$
- **Notation:** $R \underset{p}{\bowtie} S$
- **Purpose:** pairs rows from two tables
- **Output:** for each row $r$ in $R$ and each row $s$ in $S$,
  - if $p$ satisfies, output a row $rs$ (concatenation of $r$ and $s$)
  - Otherwise, output a row $r$ with NULLs
- **Right outer join and full outer join** are defined similarly
Left Outer Join Example

- **Employee** \( \Join \) **Department**

<table>
<thead>
<tr>
<th>Eid</th>
<th>Name</th>
<th>Did</th>
<th>Mid</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John Smith</td>
<td>4</td>
<td>1234</td>
<td>Research</td>
</tr>
<tr>
<td>1123</td>
<td>Mary Carter</td>
<td>5</td>
<td>1123</td>
<td>Finance</td>
</tr>
<tr>
<td>1011</td>
<td>Bob Lee</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Diagram:
- Eid = Mid
- Employee \( \Join \) Department
- Eid = Mid
Division Operator

- **Input:** two tables \( R \) and \( S \)
- **Notation:** \( R \div S \)
- **Purpose:** Find the subset of items in one set \( R \) that are related to all items in another set
Division Operator

- Find professors who have taught courses in all departments
- Why does this involve division?

Contains row <p,d> if professor p has taught a course in department d

ProfId | DeptId
------|------

DeptId

All department Ids

9/22/2014
Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations are expressed using the assignment operator.
Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes
- A deletion is expressed in relational algebra by:

\[ r \leftarrow r - E \]

where \( r \) is a relation and \( E \) is a relational algebra query.
Deletion Examples

- Delete all account records in the Perryridge branch.
  \[ \text{account} \leftarrow \text{account} - \sigma \text{branch\_name} = \text{“Perryridge”} (\text{account}) \]

- Delete all loan records with amount in the range of 0 to 50
  \[ \text{loan} \leftarrow \text{loan} - \sigma \text{amount} \geq 0 \text{ and } \text{amount} \leq 50 (\text{loan}) \]

- Delete all accounts at branches located in Needham.
  \[ r_1 \leftarrow \sigma \text{branch\_city} = \text{“Needham”} (\text{account} \bowtie \text{branch}) \]
  \[ r_2 \leftarrow \Pi \text{branch\_name, account\_number, balance} (r_1) \]
  \[ r_3 \leftarrow \Pi \text{customer\_name, account\_number} (r_2 \bowtie \text{depositor}) \]
  \[ \text{account} \leftarrow \text{account} - r_2 \]
  \[ \text{depositor} \leftarrow \text{depositor} - r_3 \]
Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- in relational algebra, an insertion is expressed by:
  \[ r \leftarrow r \cup E \]
  where \( r \) is a relation and \( E \) is a relational algebra expression.
- The insertion of a single tuple is expressed by letting \( E \) be a constant relation containing one tuple.
Insertion Examples

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

  \[
  account \leftarrow account \cup \{("Perryridge", A-973, 1200)\}
  \]
  \[
  depositor \leftarrow depositor \cup \{("Smith", A-973)\}
  \]

- Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

  \[
  r_1 \leftarrow (\sigma_{branch\_name="Perryridge"}(borrower \bowtie loan))
  \]
  \[
  account \leftarrow account \cup \prod_{branch\_name,\ loan\_number,\ 200}(r_1)
  \]
  \[
  depositor \leftarrow depositor \cup \prod_{customer\_name,\ loan\_number}(r_1)
  \]
A mechanism to change a value in a tuple without changing all values in the tuple

Use the generalized projection operator to do this task

\[ r \leftarrow \prod_{F_1,F_2,\ldots,F_i}(r) \]

Each \( F_i \) is either

- the \( i \)th attribute of \( r \), if the \( i \)th attribute is not updated, or,
- if the attribute is to be updated \( F_i \) is an expression, involving only constants and the attributes of \( r \), which gives the new value for the attribute
Update Examples

- Make interest payments by increasing all balances by 5 percent.

\[ \text{account} \leftarrow \Pi_{\text{account\_number, branch\_name, balance}} \ast 1.05 (\text{account}) \]

- Pay all accounts with balances over $10,000 6 percent interest and pay all others 5 percent

\[ \text{account} \leftarrow \Pi_{\text{account\_number, branch\_name, balance}} \ast 1.06 (\sigma_{\text{BAL} > 10000} (\text{account})) \cup \Pi_{\text{account\_number, branch\_name, balance}} \ast 1.05 (\sigma_{\text{BAL} \leq 10000} (\text{account})) \]
What is “algebra”

- Mathematical model consisting of:
  - **Operands** --- Variables or values;
  - **Operators** --- Symbols denoting procedures that construct new values from a given values

- **Relational Algebra** is algebra whose operands are relations and operators are designed to do the most commons things that we need to do with relations
Why is r.a. a good query language?

- **Simple**
  - A small set of core operators whose semantics are easy to grasp

- **Declarative?**
  - Yes, compared with older languages like CODASYL
  - Though operators do look somewhat "procedural"

- **Complete?**
  - With respect to what?
Review

- Expression tree
- Tips in writing R.A.
  - Use temporary variables
  - Use foreign keys to join tables
  - A comparison is to identify a relationship
  - Use set minus in non-monotonic results