Background

- Introduced by Codd (along with the Tuple Relational Calculus)
- Relational Algebra . . . :
  - Is procedural, like most programming languages
    - we need to supply an ordering of operations
  - Would not be a good replacement for SQL in a DBMS
  - Is a good introduction to the operators provided by SQL
Relational Operators (1 / 2)

Relations are *closed* under Relational Algebra operators

- That is, they accept relations as operands, and produce relations as results.
- Example: Integers are closed under $+$ and $-$. 

The eight basic Relational Algebra operators are:

\[
\times \quad \bigotimes
\]

\[
- \quad \pi
\]

\[
\div \quad \sigma
\]

\[
\cap \quad \bigcup
\]

Relational Operators (2 / 2)

The eight Relational Algebra operators can be grouped in two ways:

1. Set vs. Relational:
   - Set: $\cup$, $\cap$, $-$, $\times$, $\div$
   - Relational: $\sigma$, $\pi$, $\bigotimes$

2. Fundamental vs. Derived:
   - Fundamental: $\sigma$, $\pi$, $\times$, $\cup$, $-$
   - Derived: $\cap$, $\bigotimes$, $\div$
The Fundamental Operators

- Select (σ)
- Project (π)
- Cartesian Product (×)
- Union (∪)
- Difference (−)

Select (σ, sigma) (1 / 2)

- A unary (single argument) operator
- Chooses full tuples from a relation based on a condition
- Form:

**Example(s):**

List all information of the employees in department #5:

Who are the active suppliers in Paris?
Select ($\sigma$, sigma) (2 / 2)

Notes:

- Conditions may be as complex as is necessary
- Select is commutative:
  \[ \sigma_A(\sigma_B(r)) \equiv \sigma_B(\sigma_A(r)) \]
- Cascades of selects \(\equiv\) conjunction in a single select:
  \[ \sigma_A(\sigma_B(\sigma_C(r))) \equiv \sigma_{A\land B\land C}(r) \]

Project ($\pi$, pi) (1 / 2)

Pronunciation: PRO-ject (not pro-JECT, not PRAH-ject)

- Also a unary operator
- Chooses named columns from a relation
  - Resulting group of tuples may include duplicates . . .
  - . . . which we drop to preserve entity integrity
- Form:
Project \((\pi, \pi)\) (2 / 2)

Example(s):

Find the names & salaries of the employees in department 5:

Alternatively:

Cartesian Product \((\times)\) (1 / 2)

- A binary operator (form: \(R \times S\))
- ‘Marries’ all pairings of tuples from the given relations
  - resulting cardinality = \(\text{card}(R) \cdot \text{card}(S)\)
  - resulting degree = \(\text{degree}(R) + \text{degree}(S)\)

Example(s):

\[
\begin{array}{ccc}
A & m & n \\
2 & i & \\
3 & iv & \\
7 & x & \\
\end{array}
\quad
\begin{array}{ccc}
B & o & p \\
3 & \beta & \\
7 & \alpha & \\
\end{array}
\]
Example(s):

What are the names of the active suppliers of nuts?

The complete query:

\[ \pi_{Sname} \left( \sigma_{\text{Status} > 0 \land \text{Pname} = \text{Nut}} \left( \sigma_{S.S# = SP.S#} \left( \sigma_{SP.P# = P.P#} \left( S \times (SP \times P) \right) \right) \right) \right) \]

For a visualization of this query step–by–step with sample data, see the handout:

“Examples of the Relation Algebra Operations \( \sigma \), \( \pi \), and \( \times \)”

Union (\( \cup \)) (1 / 2)

- Another binary operator (form: \( R \cup S \))
- Result contains all tuples of both relations w/o duplicates

Example(s):

“Create a table of the Phoenix and Tucson employee data.”
To perform $R \cup S$, $R$ and $S$ must be union compatible.

**Definition: Union Compatible**

**Example(s):**

---

**Difference (−) (1 / 2)**

- Do you remember this one from basic sets?
  \[ \{a, b, c, d, e, f\} - \{b, d, f, h\} = \{a, c, e\} \]

- Yet another binary operator (form: $R - S$)
- Result is a relation of tuples from $R$ that are not also in $S$
- Note that $R$ and $S$ must be union compatible
Difference (−) (2 / 2)

Example(s):

<table>
<thead>
<tr>
<th>X</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y</th>
<th>u</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

The Derived Operators

- Intersection (∩)
- Join (⊳ ⊲)
- Division (÷)
Intersection \( (\cap) \) (1 / 2)

- YABO — Yet Another Binary Operator! (form: \( R \cap S \))
- Resulting relation has the tuples that appear in both operand relations
- As with difference, \( R \) and \( S \) must be union compatible

Relational Algebra – CSc 460 v1.1 (McCann) – p. 17/30

Intersection \( (\cap) \) (2 / 2)

Example(s):

\[
\begin{array}{cc}
X & s & t \\
\hline
a & 12 \\
m & 4 \\
e & 6 \\
\end{array}
\quad \begin{array}{cc}
Y & u & v \\
\hline
e & 6 \\
a & 16 \\
f & 4 \\
\end{array}
\quad \begin{array}{cc}
X \cap Y & s & t \\
\hline
\end{array}
\]

Relational Algebra – CSc 460 v1.1 (McCann) – p. 18/30
Join (⋈) (1 / 3)

- YABO! (form: R ⋈ \text{condition} S)
- Join is used to exploit PK–FK connections
  (using it with other attributes is unwise!)

Join (⋈) (2 / 3)

Example(s):

What are the names of the parts that can be supplied by individual suppliers in quantity ≥ 200?

Without ⋈: \[ π_{\text{pname}}(σ_{\text{qty} > 200}(σ_{\text{SP}.P# = \text{P}.P#}(\text{SP} \times \text{P}))) \]

With ⋈:
Join (⋈) (3 / 3)

Three join variations:

1. **Theta Join**: $r \bowtie \theta s$

2. **Equijoin**: $r \bowtie \theta s$

3. **Natural Join**: $r \bowtie s \equiv \pi_{R \cup S} (r \bowtie r.a_1 = s.a_1 \land r.a_2 = s.a_2 \land \ldots \land s)$
   where $R$ and $S$ are the attribute sets of $r$ and $s$, respectively

---

Division (÷) (1 / 9)

Let $\alpha$ and $\beta$ be relations, where $A - B$ is the set difference of the attributes of $\alpha$ & $\beta$

Definition of Relational Division:
Division (÷) (2 / 9)

Let's review multiplication and division of integers:

Ex: \(4 \times 6 = 24\), so \(24/6 = 4\) and \(24/4 = 6\).

Now, consider Cartesian Product and Division with relations:

\[
M = \begin{bmatrix}
4 \\
8
\end{bmatrix}
\quad N = \begin{bmatrix}
3 \\
1 \\
7
\end{bmatrix}
\]

\[
M \times N = O = \begin{bmatrix}
4 & 3 \\
4 & 1 \\
4 & 7 \\
8 & 3 \\
8 & 1 \\
8 & 7
\end{bmatrix}
\]

Division (÷) (3 / 9)

What purpose does Division serve?

Example(s):

Recall:

\[
S = \begin{bmatrix}
S# & Sname & Status & City
\end{bmatrix}
\]

\[
P = \begin{bmatrix}
P# & Pname & Color & Weight & City
\end{bmatrix}
\]

\[
SP = \begin{bmatrix}
S# & P# & Qty
\end{bmatrix}
\]
Division (÷) (4 / 9)

\[ \alpha \div \beta = \pi_{A,B}(\alpha) - \pi_{A,B}(\pi_{A,B}(\alpha \times \beta) - \alpha) \]

**Example(s):** (continued)

(Find the S#s of the suppliers that supply all parts of weight = 17)

“Find the X . . .” ← X is S# (the matches we want)

“. . . matched w/ all Y” ← Y is the set of weight 17 P#s

Next, create the dividend (\(\alpha\)) and divisor (\(\beta\)) relations:

The content of \(\alpha\) and \(\beta\):

<table>
<thead>
<tr>
<th>(\alpha) S#</th>
<th>P#</th>
<th>(\beta) P#</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>P1</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>P3</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>P5</td>
<td>P2</td>
</tr>
<tr>
<td>S3</td>
<td>P3</td>
<td>P3</td>
</tr>
<tr>
<td>S3</td>
<td>P4</td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>P6</td>
<td>P3</td>
</tr>
<tr>
<td>S5</td>
<td>P1</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>P2</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>P3</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>P4</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>P5</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>P6</td>
<td></td>
</tr>
</tbody>
</table>

Do any suppliers supply ALL of \(\beta\)'s parts?
Let’s examine the definition in detail:

\[
\alpha \div \beta = \pi_{S\#}(\alpha) - \pi_{S\#}(\pi_{S\#}(\alpha) \times \beta) - \alpha
\]
And so, finally, we have our answer:

\[
\begin{array}{c|c|c}
\pi_{S#}(\alpha) & \# & \pi_{S#}((\pi_{S#}(\alpha) \times \beta) - \alpha) \\
S1 & & S1 \\
S2 & & S2 \\
S3 & & S3 \\
S4 & & S4 \\
S5 & & S5 \\
\end{array}
\]

\[
\alpha \div \beta = \pi_{S#}(\alpha) - \pi_{S#}((\pi_{S#}(\alpha) \times \beta) - \alpha) = S5
\]

Supplier S5 supplies all parts of weight 17 (P2 and P3).

Is there a short-cut to avoid that mess? **NO!**

Consider this very similar query:

Find the S#s of the suppliers that supply
all parts of weight = 19.

The only weight = 19 part is P6, which this simple
expression produces: \( \pi_{S#}(\alpha \bowtie_{P#} \beta) (\rightarrow S4 \text{ and } S5) \)

But, when weight = 17, that query gives S2, S3, and S5!