Relational Algebra
Relational Query Languages

Recall: Query = “Retrieval Program”

Language Examples:

Theoretical:

1. Relational Algebra
2. Relational Calculus
   a. Tuple Relational Calculus (TRC)
   b. Domain Relational Calculus (DRC)

Practical:

1. SQL (originally: SEQUEL from System R)
2. Quel (used in Ingres)
3. Datalog (Prolog-like – used in research lab systems)

Theoretical QLs give semantics to Practical QLs
Relational Algebra

• Basic Operators
  1. select (σ)
  2. project (π)
  3. union (∪)
  4. set difference (−)
  5. cartesian product (∗)
  6. rename (ρ)

• Closure Property
Select ( σ )

Notation: $\sigma_{predicate}(Relation)$

Relation: Can be name of table or result of another query

Predicate:

1. Simple
   - attribute$_1$ = attribute$_2$
   - attribute = constant value (also: $\neq$, $<$, $>$, $\leq$, $\geq$)

2. Complex
   - predicate AND predicate
   - predicate OR predicate
   - NOT predicate

Idea:

Select rows from a relation based on a predicate
# Bank Database

## Account

<table>
<thead>
<tr>
<th>bname</th>
<th>acct_no</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Mianus</td>
<td>A-215</td>
<td>700</td>
</tr>
<tr>
<td>Perry</td>
<td>A-102</td>
<td>400</td>
</tr>
<tr>
<td>R.H.</td>
<td>A-305</td>
<td>350</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Redwood</td>
<td>A-222</td>
<td>700</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>750</td>
</tr>
</tbody>
</table>

## Branch

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Brooklyn</td>
<td>9M</td>
</tr>
<tr>
<td>Redwood</td>
<td>Palo Alto</td>
<td>2.1M</td>
</tr>
<tr>
<td>Perry</td>
<td>Horseneck</td>
<td>1.7M</td>
</tr>
<tr>
<td>Mianus</td>
<td>Horseneck</td>
<td>0.4M</td>
</tr>
<tr>
<td>R.H.</td>
<td>Horseneck</td>
<td>8M</td>
</tr>
<tr>
<td>Pownel</td>
<td>Bennington</td>
<td>0.3M</td>
</tr>
<tr>
<td>N. Town</td>
<td>Rye</td>
<td>3.7M</td>
</tr>
<tr>
<td>Brighton</td>
<td>Brooklyn</td>
<td>7.1M</td>
</tr>
</tbody>
</table>

## Depositor

<table>
<thead>
<tr>
<th>cname</th>
<th>acct_no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnson</td>
<td>A-101</td>
</tr>
<tr>
<td>Smith</td>
<td>A-215</td>
</tr>
<tr>
<td>Hayes</td>
<td>A-102</td>
</tr>
<tr>
<td>Turner</td>
<td>A-305</td>
</tr>
<tr>
<td>Johnson</td>
<td>A-201</td>
</tr>
<tr>
<td>Jones</td>
<td>A-217</td>
</tr>
<tr>
<td>Lindsay</td>
<td>A-222</td>
</tr>
</tbody>
</table>

## Borrower

<table>
<thead>
<tr>
<th>cname</th>
<th>lno</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>L-17</td>
</tr>
<tr>
<td>Smith</td>
<td>L-23</td>
</tr>
<tr>
<td>Hayes</td>
<td>L-15</td>
</tr>
<tr>
<td>Jackson</td>
<td>L-14</td>
</tr>
<tr>
<td>Curry</td>
<td>L-93</td>
</tr>
<tr>
<td>Smith</td>
<td>L-11</td>
</tr>
<tr>
<td>Williams</td>
<td>L-17</td>
</tr>
<tr>
<td>Adams</td>
<td>L-16</td>
</tr>
</tbody>
</table>

## Customer

<table>
<thead>
<tr>
<th>cname</th>
<th>cstreet</th>
<th>ccity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>Main</td>
<td>Harrison</td>
</tr>
<tr>
<td>Smith</td>
<td>North</td>
<td>Rye</td>
</tr>
<tr>
<td>Hayes</td>
<td>Main</td>
<td>Harrison</td>
</tr>
<tr>
<td>Curry</td>
<td>North</td>
<td>Rye</td>
</tr>
<tr>
<td>Lindsay</td>
<td>Park</td>
<td>Pittsfield</td>
</tr>
<tr>
<td>Turner</td>
<td>Putnam</td>
<td>Stanford</td>
</tr>
<tr>
<td>Williams</td>
<td>Nassau</td>
<td>Princeton</td>
</tr>
<tr>
<td>Adams</td>
<td>Spring</td>
<td>Pittsfield</td>
</tr>
<tr>
<td>Johnson</td>
<td>Alma</td>
<td>Palo Alto</td>
</tr>
<tr>
<td>Glenn</td>
<td>Sand Hill</td>
<td>Woodside</td>
</tr>
<tr>
<td>Brooks</td>
<td>Senator</td>
<td>Brooklyn</td>
</tr>
<tr>
<td>Green</td>
<td>Walnut</td>
<td>Stanford</td>
</tr>
</tbody>
</table>

## Loan

<table>
<thead>
<tr>
<th>bname</th>
<th>lno</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-23</td>
<td>2000</td>
</tr>
<tr>
<td>Perry</td>
<td>L-15</td>
<td>1500</td>
</tr>
<tr>
<td>Downtown</td>
<td>L-14</td>
<td>1500</td>
</tr>
<tr>
<td>Mianus</td>
<td>L-93</td>
<td>500</td>
</tr>
<tr>
<td>R.H.</td>
<td>L-11</td>
<td>900</td>
</tr>
<tr>
<td>Perry</td>
<td>L-16</td>
<td>1300</td>
</tr>
</tbody>
</table>
Select ( $\sigma$ )

Notation: $\sigma_{\text{predicate}}(\text{Relation})$

$\sigma_{\text{bcity} = \text{"Brooklyn"}} (\text{branch}) =$

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Brooklyn</td>
<td>9M</td>
</tr>
<tr>
<td>Brighton</td>
<td>Brooklyn</td>
<td>7.1M</td>
</tr>
</tbody>
</table>

$\sigma_{\text{assets} > \$8M} (\sigma_{\text{bcity} = \text{"Brooklyn"}} (\text{branch})) =$

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Brooklyn</td>
<td>9M</td>
</tr>
</tbody>
</table>
Project ( $\pi$ )

Notation: $\pi_{A_1, \ldots, A_n}$ (Relation)

- Relation: name of a table or result of another query
- Each $A_i$ is an attribute
- Idea: $\pi$ selects columns (vs. $\sigma$ which selects rows)

$\pi_{\text{cstreet, ccity}}$ (customer) =

<table>
<thead>
<tr>
<th>cstreet</th>
<th>ccity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>Harrison</td>
</tr>
<tr>
<td>North</td>
<td>Rye</td>
</tr>
<tr>
<td>Park</td>
<td>Pittsfield</td>
</tr>
<tr>
<td>Putnam</td>
<td>Stanford</td>
</tr>
<tr>
<td>Nassau</td>
<td>Princeton</td>
</tr>
<tr>
<td>Spring</td>
<td>Pittsfield</td>
</tr>
<tr>
<td>Alma</td>
<td>Palo Alto</td>
</tr>
<tr>
<td>Sand Hill</td>
<td>Woodside</td>
</tr>
<tr>
<td>Senator</td>
<td>Brooklyn</td>
</tr>
<tr>
<td>Walnut</td>
<td>Stanford</td>
</tr>
</tbody>
</table>
Project ( $\pi$ )

$$\pi_{\text{bcity}}(\sigma_{\text{assets} > 5M}(\text{branch})) = \begin{array}{|c|}
\hline
\text{bcity} \\
\hline
\text{Brooklyn} \\
\hline
\text{Horseneck} \\
\hline
\end{array}$$

**Question:** Does the result of Project always have the same number of tuples as its input?
Union $(\cup)$

**Notation:** $\text{Relation}_1 \cup \text{Relation}_2$

$R \cup S$ valid only if:

1. $R, S$ have same number of columns (*arity*)
2. $R, S$ corresponding columns have same name and domain (*compatibility*)

**Example:**

$$(\pi\text{ _\text{cname} (depositor)})) \cup (\pi\text{ _\text{cname} (borrower)}) =$$

**Schema:**

<table>
<thead>
<tr>
<th>Depositor</th>
<th>Borrower</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cname</strong></td>
<td><strong>lname</strong></td>
</tr>
<tr>
<td>acct_no</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnson</td>
<td>Smith</td>
<td>Hayes</td>
</tr>
<tr>
<td>Turner</td>
<td>Jones</td>
<td>Lindsay</td>
</tr>
<tr>
<td>Jackson</td>
<td>Curry</td>
<td>Williams</td>
</tr>
<tr>
<td>Adams</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Set Difference (−)

Notation: $Relation_1 - Relation_2$

$R - S$ valid only if:

1. $R, S$ have same number of columns (arity)
2. $R, S$ corresponding columns have same domain (compatibility)

Example:

$$(\pi_{bname} (\sigma_{\text{amount} \geq 1000 (\text{loan})})) - (\pi_{bname} (\sigma_{\text{balance} < 800 (\text{account})})) =$$
What About Intersection?

Remember:

$$R \cap S = R - (R - S)$$
Cartesian Product \((\times)\)

**Notation:** \(\text{Relation}_1 \times \text{Relation}_2\)

\(R \times S\) like cross product for mathematical relations:

- *every tuple of \(R\) appended to every tuple of \(S\)*
- *flattened!!!*

**Example:**

\[
\text{depositor} \times \text{borrower} =
\]

<table>
<thead>
<tr>
<th>depositor. cname</th>
<th>acct_no</th>
<th>borrower. cname</th>
<th>lno</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnson</td>
<td>A-101</td>
<td>Jones</td>
<td>L-17</td>
</tr>
<tr>
<td>Johnson</td>
<td>A-101</td>
<td>Smith</td>
<td>L-23</td>
</tr>
<tr>
<td>Johnson</td>
<td>A-101</td>
<td>Hayes</td>
<td>L-15</td>
</tr>
<tr>
<td>Johnson</td>
<td>A-101</td>
<td>Jackson</td>
<td>L-14</td>
</tr>
<tr>
<td>Johnson</td>
<td>A-101</td>
<td>Curry</td>
<td>L-93</td>
</tr>
<tr>
<td>Johnson</td>
<td>A-101</td>
<td>Smith</td>
<td>L-11</td>
</tr>
<tr>
<td>Johnson</td>
<td>A-101</td>
<td>Williams</td>
<td>L-17</td>
</tr>
<tr>
<td>Johnson</td>
<td>A-101</td>
<td>Adams</td>
<td>L-16</td>
</tr>
<tr>
<td>Smith</td>
<td>A-215</td>
<td>Jones</td>
<td>L-17</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

*How many tuples in the result?*

**A:** 56
rename ( ρ )

Notation:  \( \rho \) \_\text{identifier} (\text{Relation})

renames a relation, or

Notation:  \( \rho \) \_\text{identifier}_0 (\text{identifier}_1, \ldots, \text{identifier}_n) (\text{Relation})

renames relation and columns of \( n \)-column relation

Use:

massage relations to make \( \cup, \cap, - \) valid, or \( \times \) more readable
Rename (ρ)

Notation: $\rho \text{identifier}_0 (\text{identifier}_1, \ldots, \text{identifier}_n) (\text{Relation})$

Example:

$\rho \text{result (dcname, acctno, bname, lno)} (\text{depositor} \times \text{borrower}) = \begin{array}{|c|c|c|c|} \hline \text{dcname} & \text{acctno} & \text{bname} & \text{lno} \\ \hline \text{Johnson} & \text{A-101} & \text{Jones} & \text{L-17} \\ \text{Johnson} & \text{A-101} & \text{Smith} & \text{L-23} \\ \text{Johnson} & \text{A-101} & \text{Hayes} & \text{L-15} \\ \text{Johnson} & \text{A-101} & \text{Jackson} & \text{L-14} \\ \text{Johnson} & \text{A-101} & \text{Curry} & \text{L-93} \\ \text{Johnson} & \text{A-101} & \text{Smith} & \text{L-11} \\ \text{Johnson} & \text{A-101} & \text{Williams} & \text{L-17} \\ \text{Johnson} & \text{A-101} & \text{Adams} & \text{L-16} \\ \text{Smith} & \text{A-215} & \text{Jones} & \text{L-17} \\ \ldots & \ldots & \ldots & \ldots \\ \hline \end{array}$
Example Query in RA

• *Determine ino for loans that are for an amount that is larger than the amount of some other loan.* (i.e. ino for all non-minimal loans)

Can do in steps:

\[
\text{Temp}_1 \leftarrow \ldots \\
\text{Temp}_2 \leftarrow \ldots \text{ Temp}_1 \ldots \\
\ldots
\]
Example Query in RA

1. Find the base data we need

\[ \text{Temp}_1 \leftarrow \pi_{\text{ino}, \text{amt}} (\text{loan}) \]

2. Make a copy of (1)

\[ \text{Temp}_2 \leftarrow \rho_{\text{Temp}_2 (\text{ino}_2, \text{amt}_2)} (\text{Temp}_1) \]
Example Query in RA

3. Take the cartesian product of 1 and 2

\[ \text{Temp}_3 \leftarrow \text{Temp}_1 \times \text{Temp}_2 \]

<table>
<thead>
<tr>
<th>lno</th>
<th>amt</th>
<th>lno2</th>
<th>amt2</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-17</td>
<td>1000</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>L-17</td>
<td>1000</td>
<td>L-23</td>
<td>2000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>L-17</td>
<td>1000</td>
<td>L-16</td>
<td>1300</td>
</tr>
<tr>
<td>L-23</td>
<td>2000</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>L-23</td>
<td>2000</td>
<td>L-23</td>
<td>2000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>L-23</td>
<td>2000</td>
<td>L-16</td>
<td>1300</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example Query in RA

4. Select non-minimal loans

\[ \text{Temp}_4 \leftarrow \sigma_{\text{amt} > \text{amt}_2} (\text{Temp}_3) \]

5. Project on \text{lno}

\[ \text{Result} \leftarrow \pi_{\text{lno}} (\text{Temp}_4) \]

... or, if you prefer...

\[ \pi_{\text{lno}} (\sigma_{\text{amt} > \text{amt}_2} (\pi_{\text{lno,amt}} (\text{loan}) \times (\rho_{\text{Temp}_2 (\text{lno}_2, \text{amt}_2)} (\pi_{\text{lno,amt}} (\text{loan})))))) \]
Relational Algebra

1. SELECT (σ)
2. PROJECT (π)
3. UNION (∪)
4. SET DIFFERENCE (–)
5. CARTESIAN PRODUCT (×)
6. RENAME (ρ)

• Relational algebra gives semantics to practical query languages
• Above set: minimal relational algebra
  ➔ will now look at some redundant (but useful!) operators
Review

Express the following query in the RA:

Find the names of customers who have both accounts and loans

\[ T_1 \leftarrow \rho T_1 \left( \text{cname2, lno} \right) (\text{borrower}) \]
\[ T_2 \leftarrow \text{depositor} \times T_1 \]
\[ T_3 \leftarrow \sigma_{\text{cname} = \text{cname2}} (T_2) \]
\[ \text{Result} \leftarrow \pi_{\text{cname}} (T_3) \]

Above sequence of operators (\( \rho \), \( \times \), \( \sigma \)) very common.

Motivates additional (redundant) RA operators.
Relational Algebra

Additional Operators

1. Natural Join (⋈)
2. Division ( ÷ )
3. Generalized Projection (π)
4. Aggregation
5. Outer Joins (outer join)
6. Update (←) (we’ve already been using this)

- **1 & 2 Redundant:** Can be expressed in terms of minimal RA
  - e.g. depositor ⋈ borrower =
    - \( \pi \ldots (\sigma \ldots (\text{depositor} \times \rho \ldots (\text{borrower}))) \)

- **3 – 6 Added for extra power**
Natural Join

Notation: \(R_1 \bowtie \bowtie R_2\)

Idea: combines \(\rho, \times, \sigma\)

\[
\begin{array}{|c|c|c|c|}
\hline
A & B & C & D \\
\hline
1 & \alpha & + & 10 \\
2 & \alpha & - & 10 \\
2 & \alpha & - & 20 \\
3 & \beta & + & 10 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
E & B & D \\
\hline
\text{‘a’} & \alpha & 10 \\
\text{‘a’} & \alpha & 20 \\
\text{‘b’} & \beta & 10 \\
\text{‘c’} & \beta & 10 \\
\hline
\end{array}
\]

\[
R \bowtie S = \\
\begin{array}{|c|c|c|c|}
\hline
A & B & C & D & E \\
\hline
1 & \alpha & + & 10 & \text{‘a’} \\
2 & \alpha & - & 10 & \text{‘a’} \\
2 & \alpha & - & 20 & \text{‘a’} \\
3 & B & + & 10 & \text{‘b’} \\
3 & \beta & + & 10 & \text{‘c’} \\
\hline
\end{array}
\]

depositor \bowtie \bowtie \bowtie \text{borrower}

\[
\equiv \\
\pi_{\text{cname, acct_no, lno}} (\sigma_{\text{cname} = \text{cname2}} (\text{depositor} \times \rho_{t(\text{cname2, lno})} (\text{borrower})))
\]
Division

Notation: $\text{Relation}_1 \div \text{Relation}_2$

Idea: expresses “for all” queries

Query: Find values for $A$ in $r$ which have corresponding $B$ values for all $B$ values in $s$
Division

Another way to look at it: \( \div \) and \( \times \)

\[
17 \div 3 = 5
\]

The largest value of \( i \) such that: \( i \times 3 \leq 17 \)

Relational Division

\[
\begin{array}{|c|c|c|}
\hline
r & A & B \\
\hline
\alpha & 1 & \\
\alpha & 2 & \\
\alpha & 3 & \\
\beta & 1 & \\
\gamma & 1 & \\
\gamma & 3 & \\
\gamma & 4 & \\
\gamma & 6 & \\
\delta & 1 & \\
\delta & 2 & \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
s & B \\
\hline
1 & \\
2 & \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
t & A \\
\hline
\alpha & \\
\delta & \\
\hline
\end{array}
\]

The largest value of \( t \) such that:
\[
(t \times s \subseteq r)
\]
A More Complex Example

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
& A & B & C & D & E \\
\hline
\alpha & a & a & \alpha & a & 1 \\
\alpha & a & \gamma & a & \gamma & 1 \\
\alpha & a & \gamma & b & 1 \\
\beta & a & \gamma & a & \beta & 3 \\
\beta & a & \gamma & b & 1 \\
\gamma & a & \gamma & a & 1 \\
\gamma & a & \gamma & b & 1 \\
\gamma & a & \beta & b & 1 \\
\hline
\end{array}
\]

\[ \div \]

\[
\begin{array}{|c|c|}
\hline
D & E \\
\hline
a & 1 \\
b & 1 \\
\hline
\end{array}
\]

\[=\]

\[
\begin{array}{|c|c|c|}
\hline
A & B & C \\
\hline
\alpha & a & \gamma \\
\gamma & a & \gamma \\
\hline
\end{array}
\]

?
Division Adds No Power

Definition in terms of the basic algebra operation
Let \( r(R) \) and \( s(S) \) be relations, and let \( S \subseteq R \)

\[
r \div s = \prod_{R-S} (r) - \prod_{R-S} ( (\prod_{R-S} (r) \times s) - \prod_{R-S,S} (r) )
\]

To see why
- \( \prod_{R-S,S} (r) \) simply reorders attributes of \( r \)
- \( \prod_{R-S} (\prod_{R-S} (r) \times s) - \prod_{R-S,S} (r) \) gives those tuples \( t \) in
  \( \prod_{R-S} (r) \) such that for some tuple \( u \in s, tu \notin r \).
Generalized Projection

Notation: $\pi_{e_1,\ldots,e_n} (Relation)$

$e_1,\ldots,e_n$ can include arithmetic expressions – not just attributes

Example

<table>
<thead>
<tr>
<th></th>
<th>limit</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>5000</td>
<td>2000</td>
</tr>
<tr>
<td>Turner</td>
<td>3000</td>
<td>2500</td>
</tr>
</tbody>
</table>

credit =

Then...

$\pi_{\text{name}, \text{limit - balance}} (\text{credit}) =$

<table>
<thead>
<tr>
<th></th>
<th>limit-balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>3000</td>
</tr>
<tr>
<td>Turner</td>
<td>500</td>
</tr>
</tbody>
</table>
Aggregate Functions and Operations

★ An **aggregate 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

```
G1, G2, ..., Gn  g  F1( A1), F2( A2),..., Fn( An) (E)
```

- E is any relational-algebra expression
- G₁, G₂ ..., Gₙ is a list of attributes on which to group
  (can be empty)
- Each Fᵢ is an aggregate function
- Each Aᵢ is an attribute name
Aggregate Operation – Example

- Relation $r$:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$\alpha$</td>
<td>7</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>7</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>3</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>10</td>
</tr>
</tbody>
</table>

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

$\text{sum-C}$

27

No grouping
Aggregate Operation – Example

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

<table>
<thead>
<tr>
<th><code>branch-name</code></th>
<th><code>account-number</code></th>
<th><code>balance</code></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><code>branch-name</code></th>
<th><code>sum(balance)</code></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>
Aggregate Functions (Cont.)

Result of aggregation does not have a name

– Can use rename operation to give it a name
– For convenience, we permit renaming as part of aggregate operation

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

Motivation:

\[
\text{loan} = \begin{array}{ccc}
\text{bname} & \text{lno} & \text{amt} \\
\text{Downtown} & L-170 & 3000 \\
\text{Redwood} & L-230 & 4000 \\
\text{Perry} & L-260 & 1700
\end{array}
\]

\[
\text{borrower} = \begin{array}{c}
\text{cname} \\
\text{Jones} \\
\text{Smith} \\
\text{Hayes}
\end{array}
\]

\[
\begin{array}{ccc}
\text{lno} \\
L-170 \\
L-230 \\
L-155
\end{array}
\]

\[
\text{loan} \bowtie \text{borrower} = \begin{array}{cccc}
\text{bname} & \text{lno} & \text{amt} & \text{cname} \\
\text{Downtown} & L-170 & 3000 & \text{Jones} \\
\text{Redwood} & L-230 & 4000 & \text{Smith}
\end{array}
\]

Join result loses…

→ any record of Perry
→ any record of Hayes
Outer Joins

1. Left Outer Join ($\bowtie$)
   
   $\text{preserves all tuples in left relation}$

   $\text{loan} \bowtie \text{borrower} =$

<table>
<thead>
<tr>
<th>bname</th>
<th>lno</th>
<th>amt</th>
<th>cname</th>
<th>lno</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-170</td>
<td>3000</td>
<td>Jones</td>
<td>L-170</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
<td>Smith</td>
<td>L-230</td>
</tr>
<tr>
<td>Perry</td>
<td>L-260</td>
<td>1700</td>
<td>Hayes</td>
<td>L-155</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\bot</td>
</tr>
</tbody>
</table>

\(\bot = \text{NULL}\)
Outer Joins

2. Right Outer Join (\(\bowtie\))
   - preserves all tuples in right relation

\[
\begin{align*}
\text{loan} & = \\
\text{borrower} & = \\
\hline
\text{bname} & | \text{lno} | \text{amt} & | \text{cname} & | \text{lno} \\
\text{Downtown} & | L-170 | 3000 & | Jones & | L-170 \\
\text{Redwood} & | L-230 | 4000 & | Smith & | L-230 \\
\text{Perry} & | L-260 | 1700 & | Hayes & | L-155 \\
\hline
\end{align*}
\]

\[
\begin{align*}
\text{loan} \bowtie \text{borrower} & = \\
\hline
\text{bname} & | \text{lno} | \text{amt} & | \text{cname} & | \text{lno} \downarrow = \text{NULL} \\
\text{Downtown} & | L-170 | 3000 & | Jones & | L-170 \\
\text{Redwood} & | L-230 | 4000 & | Smith & | L-230 \\
\downarrow & | L-155 | \downarrow & | Hayes & \\
\hline
\end{align*}
\]
### Outer Joins

<table>
<thead>
<tr>
<th>bname</th>
<th>lno</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-170</td>
<td>3000</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
</tr>
<tr>
<td>Perry</td>
<td>L-260</td>
<td>1700</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cname</th>
<th>lno</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>L-170</td>
</tr>
<tr>
<td>Smith</td>
<td>L-230</td>
</tr>
<tr>
<td>Hayes</td>
<td>L-155</td>
</tr>
</tbody>
</table>

3. **Full Outer Join (⪯⪯)**

- *preserves all tuples in both relations*

\[
\text{loan} \⪯⪯\text{borrower} = \]

<table>
<thead>
<tr>
<th>bname</th>
<th>lno</th>
<th>amt</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>L-170</td>
<td>3000</td>
<td>Jones</td>
</tr>
<tr>
<td>Redwood</td>
<td>L-230</td>
<td>4000</td>
<td>Smith</td>
</tr>
<tr>
<td>Perry</td>
<td>L-260</td>
<td>1700</td>
<td>Hayes</td>
</tr>
<tr>
<td>↓</td>
<td>L-155</td>
<td>↓</td>
<td>↓</td>
</tr>
</tbody>
</table>

\[\bot = \text{NULL}\]
Update

Notation:  $\textbf{Identifier} \leftarrow \textbf{Query}$

Common Uses:

1. **Deletion**: $r \leftarrow r - s$
   
   e.g., account $\leftarrow$ account $- \sigma_{\text{bname}=\text{Perry}}$(account)
   
   *(deletes all Perry accounts)*

2. **Insertion**: $r \leftarrow r \cup s$
   
   e.g., branch $\leftarrow$ branch $\cup \{(\text{Waltham, Boston, 7M})\}$
   
   *(inserts new branch with bname = Waltham, bcity = Boston, assets = 7M)*

   e.g., depositor $\leftarrow$ depositor $\cup (\rho_{\text{temp}}(\text{cname,acct_no})(\text{borrower}))$
   
   *(adds all borrowers to depositors, treating lno’s as acct_no’s)*

3. **Update**: $r \leftarrow \pi_{e_1,\ldots,e_n} (r)$
   
   e.g., account $\leftarrow \pi_{\text{bname,acct_no,bal*1.05}}$(account)
   
   *(adds 5% interest to account balances)*
Views

• Limited access to DB.
• Tailored schema

• Consider a person who needs to know a customer’s loan number but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by

$$\Pi_{\text{customer-name, loan-number}} (\text{borrower} \bowtie \text{loan})$$

• Any relation that is not of the conceptual model but is made visible to a user as a “virtual relation” is called a view.
View Definition

- A view is defined using the `create view` statement which has the form:

  
  ```
  create view v as <query expression>
  
  where <query expression> is any legal relational algebra query expression. The view name is represented by v.
  ```

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.

- View definition is not the same as creating a new relation by evaluating the query expression. Rather, a view definition causes the saving of an expression to be substituted into queries using the view.
View Examples

• Consider the view (named *all-customer*) consisting of branches and their customers.

  create view *all-customer* as

  \[\Pi_{\text{branch-name}, \text{customer-name}} (\text{depositor} \Join \text{account}) \cup \Pi_{\text{branch-name}, \text{customer-name}} (\text{borrower} \Join \text{loan})\]

• We can find all customers of the Perryridge branch by writing:

  \[\Pi_{\text{customer-name}} (\sigma_{\text{branch-name} = \text{“Perryridge”}} (\text{all-customer}))\]
Updates Through View

• Database modifications expressed as views must be translated to modifications of the actual relations in the database.

• Consider the person who needs to see all loan data in the loan relation except amount. The view given to the person, branch-loan, is defined as:

  create view branch-loan as
  \[ \Pi_{\text{branch-name, loan-number}} \text{(loan)} \]

• Since we allow a view name to appear wherever a relation name is allowed, the person may write:

  \[ \text{branch-loan} \leftarrow \text{branch-loan} \cup \{(\text{"Perryridge"}, \text{L-37})\} \]
Updates Through Views (Cont.)

- The previous insertion must be represented by an insertion into the actual relation *loan* from which the view *branch-loan* is constructed.

- An insertion into *loan* requires a value for *amount*. The insertion can be dealt with by either:
  - rejecting the insertion and returning an error message to the user.
  - inserting a tuple ("L-37", “Perryridge”, null) into the *loan* relation
Updates Through Views (Cont.)

• Some updates through views are impossible to translate into database relation updates
  – create view \( v \) as \( \sigma_{branch-name = "Perryridge"} (account) \)
    \( v \leftarrow v \cup (L-99, \text{Downtown}, 23) \)

• Others cannot be translated uniquely
  – all-customer \( \leftarrow \) all-customer \( \cup (\text{Perryridge}, \text{John}) \)
    • Have to choose loan or account, and create a new loan/account number!
Views Defined Using Other Views

• One view may be used in the expression defining another view

• A view relation $v_1$ is said to depend directly on a view relation $v_2$ if $v_2$ is used in the expression defining $v_1$

• A view relation $v_1$ is said to depend on view relation $v_2$ if either $v_1$ depends directly on $v_2$ or there is a path of dependencies from $v_1$ to $v_2$
Let view $v_1$ be defined by an expression $e_1$ that may itself contain uses of view relations.

View expansion of an expression repeats the following replacement step:

```
repeat
  Find any view relation $v_i$ in $e_1$
  Replace the view relation $v_i$ by the expression defining $v_i$
until no more view relations are present in $e_1$
```

As long as the view definitions are not recursive, this loop will terminate.