<table>
<thead>
<tr>
<th>Date</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan. 22 (T)</td>
<td>SQL Queries.</td>
</tr>
<tr>
<td>Jan. 24 (TH)</td>
<td>Subqueries, Grouping and Aggregation.</td>
</tr>
<tr>
<td>Jan. 29 (T)</td>
<td>Modifications, Schemas, Views.</td>
</tr>
<tr>
<td></td>
<td>Read Sections 6.1-6.2. Assignment 2 due.</td>
</tr>
<tr>
<td></td>
<td>Read Sections 6.3-6.4. Project Part 2 due.</td>
</tr>
<tr>
<td></td>
<td>Read Sections 6.5-6.7. Assignment 3 due.</td>
</tr>
</tbody>
</table>
“Core” Relational Algebra

A small set of operators that allow us to manipulate relations in limited but useful ways. The operators are:
1. Union, intersection, and difference: the usual set operators.
   - But the relation schemas must be the same.
2. Selection: Picking certain rows from a relation.
4. Products and joins: Composing relations in useful ways.
5. Renaming of relations and their attributes.
Relational Algebra

- limited expressive power (subset of possible queries)
- good optimizer possible
- rich enough language to express enough useful things

Finiteness

\[ \sigma \text{ SELECT} \]
\[ \pi \text{ PROJECT} \]
\[ \times \text{ CARTESIAN PRODUCT} \]
\[ \cup \text{ UNION} \]
\[ - \text{ SET-DIFFERENCE} \]
\[ \cap \text{ SET-INTERSECTION} \]
\[ \Join_\theta \text{ THETA-JOIN} \]
\[ \bowtie \text{ NATURAL JOIN} \]
\[ \div \text{ DIVISION or QUOTIENT} \]

\begin{align*}
\sigma & \text{ SELECT} & \text{UNARY} & \text{FUNDAMENTAL} \\
\pi & \text{ PROJECT} & \text{BINARY} & \text{FUNDAMENTAL} \\
\times & \text{ CARTESIAN PRODUCT} & & \\
\cup & \text{ UNION} & & \\
- & \text{ SET-DIFFERENCE} & & \\
\cap & \text{ SET-INTERSECTION} & & \\
\Join_\theta & \text{ THETA-JOIN} & \text{CAN BE DEFINED} & \text{IN TERMS OF} \\
\bowtie & \text{ NATURAL JOIN} & & \text{FUNDAMENTAL OPS} \\
\div & \text{ DIVISION or QUOTIENT} & & \\
\end{align*}
Extra Example Relations

DEPOSIT(branch-name, acct-no, cust-name, balance)
CUSTOMER(cust-name, street, cust-city)
BORROW(branch-name, loan-no, cust-name, amount)
BRANCH(branch-name, assets, branch-city)
CLIENT(cust-name, empl-name)

<table>
<thead>
<tr>
<th>Borrow</th>
<th>B-N</th>
<th>L-#</th>
<th>C-N</th>
<th>AMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Midtown</td>
<td>123</td>
<td>Fred</td>
<td>600</td>
</tr>
<tr>
<td>T2</td>
<td>Midtown</td>
<td>234</td>
<td>Sally</td>
<td>1200</td>
</tr>
<tr>
<td>T3</td>
<td>Midtown</td>
<td>235</td>
<td>Sally</td>
<td>1500</td>
</tr>
<tr>
<td>T4</td>
<td>Downtown</td>
<td>612</td>
<td>Tom</td>
<td>2000</td>
</tr>
</tbody>
</table>
### Selection

\[ R_1 = \sigma_C(R_2) \]

where \( C \) is a condition involving the attributes of relation \( R_2 \).

#### Example

**Relation **Sells:**

<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>2.75</td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Sue's</td>
<td>Coors</td>
<td>3.00</td>
</tr>
</tbody>
</table>

\[ \text{JoeMenu} = \sigma_{\text{bar}=\text{Joe's}}(\text{Sells}) \]

<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>2.75</td>
</tr>
</tbody>
</table>
SELECT $\sigma$  

$\text{arity}(\sigma(R)) = \text{arity}(R)$

$0 \leq \text{card}(\sigma(R)) \leq \text{card}(R)$

$\sigma_c(R)$  

$\sigma_c(R) \subseteq (R)$

$c$ is selection condition: terms of form: attr op value  attr op attr

op is one of $< = > \leq \neq \geq$

example of term: $\text{branch-name} = "\text{Midtown}"$

terms are connected by $\land \lor \neg$

$\sigma \text{ branch-name} = "\text{Midtown}" \land \text{amount} > 1000 \ (\text{Borrow})$

$\sigma \text{ cust-name} = \text{emp-name} \ (\text{client})$
Projection

\[ R_1 = \pi_L(R_2) \]

where \( L \) is a list of attributes from the schema of \( R_2 \).

Example

\[ \pi_{\text{beer, price}}(\text{Sells}) \]

<table>
<thead>
<tr>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Miller</td>
<td>2.75</td>
</tr>
<tr>
<td>Coors</td>
<td>3.00</td>
</tr>
</tbody>
</table>

• Notice elimination of duplicate tuples.
Projection \( (\pi) \)

\[ 0 \leq \text{card} (\pi_A (R)) \leq \text{card} (R) \]

\[ \text{arity} (\pi_A (R)) = m \leq \text{arity}(R) = k \]

\[ \pi_{i_1, \ldots, i_m} (R) \quad 1 \leq i_j \leq k \text{ distinct} \]

produces set of \( m \)-tuples \( \langle a_1, \ldots, a_m \rangle \)

such that \( \exists k \)-tuple \( \langle b_1, \ldots, b_k \rangle \) in \( R \) where \( a_j = b_{i_j} \) for \( j = 1, \ldots, m \)

\[ \pi \quad \text{branch-name, cust-name} \quad (\text{Borrow}) \]

Midtown Fred
Midtown Sally
Downtown Tom
Product

\[ R = R_1 \times R_2 \]

pairs each tuple \( t_1 \) of \( R_1 \) with each tuple \( t_2 \) of \( R_2 \) and puts in \( R \) a tuple \( t_1 t_2 \).
Cartesian Product ($\times$)

$\text{arity}(R) = k_1$  $\text{arity}(R \times S) = k_1 + k_2$

$\text{arity}(S) = k_2$  $\text{card}(R \times S) = \text{card}(R) \times \text{card}(S)$

$R \times S$ is the set all possible ($k_1 + k_2$)-tuples

whose first $k_1$ attributes are a tuple in $R$

last $k_2$ attributes are a tuple in $S$

\[
\begin{array}{cccc}
A & B & C & D \\
D & E & F \\
\end{array}
\quad
\begin{array}{cccc}
A & B & C & D' & E & F \\
\end{array}
\quad
\begin{array}{cccc}
A & B & C & D & D' & E & F \\
\end{array}
\]
Theta-Join

\[ R = R_1 \bowtie_C R_2 \]

is equivalent to \( R = \sigma_C(R_1 \times R_2) \).
## Example

### Sells

<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>2.75</td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Sue's</td>
<td>Coors</td>
<td>3.00</td>
</tr>
</tbody>
</table>

### Bars

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Maple St.</td>
</tr>
<tr>
<td>Sue's</td>
<td>River Rd.</td>
</tr>
</tbody>
</table>

### BarInfo = Sells $\bowtie$ Sells.Bar=Bars.Name Bars

<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
<th>name</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>2.50</td>
<td>Joe's</td>
<td>Maple St.</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>2.75</td>
<td>Joe's</td>
<td>Maple St.</td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
<td>2.50</td>
<td>Sue's</td>
<td>River Rd.</td>
</tr>
<tr>
<td>Sue's</td>
<td>Coors</td>
<td>3.00</td>
<td>Sue's</td>
<td>River Rd.</td>
</tr>
</tbody>
</table>
Theta-Join

\[ R \bowtie_{\theta} S \]

\[ \sigma_{i \theta j}^{(r+j)} (R \times S) \]

\[
\begin{array}{c|c}
R & S \\
\hline
1 \ldots r & 1 \ldots s \\
\hline
i & j
\end{array}
\]

\[
\text{arity}(R) = r \quad \text{arity}(S) = s
\]

\[
\text{arity } (R \bowtie_{\theta} S) = r + s
\]

\[
0 \leq \text{card}(R \bowtie_{\theta} S) \leq \text{card}(R) \times \text{card}(S)
\]

\[ \theta \text{ can be } < > = \neq \leq \geq \]

If equal (=), then it is an EQUIJoin

\[ R \bowtie_{C} S = \sigma_{C} (R \times S) \]

<table>
<thead>
<tr>
<th>R(ABC)</th>
<th>S(CDE)</th>
<th>T(ABCC'DE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 3 5</td>
<td>2 1 1</td>
<td>1 3 5 1 2 2</td>
</tr>
<tr>
<td>2 4 6</td>
<td>1 2 2</td>
<td>1 3 5 3 3 4</td>
</tr>
<tr>
<td>3 5 7</td>
<td>3 3 4</td>
<td>1 3 5 4 4 3</td>
</tr>
<tr>
<td>4 6 8</td>
<td>4 4 3</td>
<td>2 4 6 3 3 4</td>
</tr>
</tbody>
</table>

result has schema \[ T(A \ B \ C \ C' \ D \ E) \]

Winter 2002

Arthur Keller – CS 180

5–13
Natural Join

\[ R = R_1 \bowtie R_2 \]

calls for the theta-join of \( R_1 \) and \( R_2 \) with the condition that all attributes of the same name be equated. Then, one column for each pair of equated attributes is projected out.

Example

Suppose the attribute name in relation Bars was changed to bar, to match the bar name in Sells.

\[
\text{BarInfo} = \text{Sells} \bowtie \text{Bars}
\]

<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>2.50</td>
<td>Maple St.</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>2.75</td>
<td>Maple St.</td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
<td>2.50</td>
<td>River Rd.</td>
</tr>
<tr>
<td>Sue's</td>
<td>Coors</td>
<td>3.00</td>
<td>River Rd.</td>
</tr>
</tbody>
</table>
Renaming

\( \rho_{S(A_1, \ldots, A_n)}(R) \) produces a relation identical to \( R \) but named \( S \) and with attributes, in order, named \( A_1, \ldots, A_n \).

Example

\begin{align*}
\text{Bars} = & \quad \begin{array}{|c|c|}
\hline
\text{name} & \text{addr} \\
\hline
\text{Joe's} & \text{Maple St.} \\
\text{Sue's} & \text{River Rd.} \\
\hline
\end{array} \\
\rho_{R(\text{bar, addr})}(\text{Bars}) = & \quad \begin{array}{|c|c|}
\hline
\text{bar} & \text{addr} \\
\hline
\text{Joe's} & \text{Maple St.} \\
\text{Sue's} & \text{River Rd.} \\
\hline
\end{array}
\end{align*}

• The name of the second relation is \( R \).
Union \((R \cup S)\)  \(\text{arity}(R) = \text{arity}(S) = \text{arity}(R \cup S)\)

\[
\max(\text{card}(R), \text{card}(S)) \leq \text{card}(R \cup S) \leq \text{card}(R) + \text{card}(S)
\]

set of tuples in \(R\) or \(S\) or both  \(R \subseteq R \cup S\)

\(S \subseteq R \cup S\)

Find customers of Perryridge Branch

\(\pi_{\text{Cust-Name}} (\sigma \text{ Branch-Name} = "Perryridge" (\text{BORROW} \cup \text{DEPOSIT}) )\)
Difference($R - S$)

\[ \text{arity}(R) = \text{arity}(S) = \text{arity}(R - S) \]

\[ 0 \leq \text{card}(R - S) \leq \text{card}(R) \quad \emptyset \subseteq R - S \subseteq R \]

is the tuples in $R$ not in $S$

Depositors of Perryridge who aren't borrowers of Perryridge

\[ \pi_{\text{Cust-Name}} (\sigma_{\text{Branch-Name} = "Perryridge"} (\text{DEPOSIT} - \text{BORROW})) \]

Deposit  $< \text{Perryridge, 36, Pat, 500} >$

Borrow  $< \text{Perryridge, 72, Pat, 10000} >$

\[ \pi_{\text{Cust-Name}} (\sigma_{\text{Branch-Name} = "Perryridge"} (\text{DEPOSIT})) - \]

\[ \pi_{\text{Cust-Name}} (\sigma_{\text{Branch-Name} = "Perryridge"} (\text{BORROW})) \]

Does  $\sigma (\pi (D) - \pi (B))$ work?
Combining Operations

Algebra =
1. Basis arguments +
2. Ways of constructing expressions.

For relational algebra:
1. Arguments = variables standing for relations + finite, constant relations.
2. Expressions constructed by applying one of the operators + parentheses.

Query = expression of relational algebra.
\[ \pi \text{Cust-Name,Cust-City} \]
\[ (\sigma \text{CLIENT.Banker-Name} = "Johnson" \]
\[ (\text{CLIENT} \times \text{CUSTOMER}) ) = \]
\[ \pi \text{Cust-Name,Cust-City} \ (\text{CUSTOMER}) \]

• Is this always true?

\[ \pi \text{CLIENT.Cust-Name, CUSTOMER.Cust-City} \]
\[ (\sigma \text{CLIENT.Banker-Name} = "Johnson" \]
\[ \ \wedge \ \text{CLIENT.Cust-Name} = \text{CUSTOMER.Cust-Name} \]
\[ (\text{CLIENT} \times \text{CUSTOMER}) ) \]

\[ \pi \text{CLIENT.Cust-Name, CUSTOMER.Cust-City} \]
\[ (\sigma \text{CLIENT.Cust-Name} = \text{CUSTOMER.Cust-Name} \]
\[ (\text{CUSTOMER} \times \pi_{\text{Cust-Name}} \]
\[ (\sigma \text{CLIENT.Banker-Name} = "Johnson" \ (\text{CLIENT}) ) ) \]
SET INTERSECTION

\[ (R \cap S) \]

\[ \text{arity}(R) = \text{arity}(S) = \text{arity}(R \cap S) \]

\[ 0 \leq \text{card}(R \cap S) \leq \text{min}(\text{card}(R), \text{card}(S)) \]

tuples both in R and in S

\[ \emptyset \subseteq R \cap S \subseteq R \]

\[ \emptyset \subseteq R \cap S \subseteq S \]

\[ R - (R - S) = R \cap S \]
Operator Precedence

The normal way to group operators is:

1. Unary operators $\sigma$, $\pi$, and $\rho$ have highest precedence.
2. Next highest are the “multiplicativc” operators, $\Join$, $\Join_C$, and $\times$.
3. Lowest are the “additive” operators, $\cup$, $\cap$, and $\neg$.

- But there is no universal agreement, so we always put parentheses around the argument of a unary operator, and it is a good idea to group all binary operators with parentheses enclosing their arguments.

Example

Group $R \cup \sigma S \Join T$ as $R \cup (\sigma(S) \Join T)$. 
Each Expression Needs a Schema

- If $\cup$, $\cap$, — applied, schemas are the same, so use this schema.
- Projection: use the attributes listed in the projection.
- Selection: no change in schema.
- Product $R \times S$: use attributes of $R$ and $S$.
  - But if they share an attribute $A$, prefix it with the relation name, as $R.A$, $S.A$.
- Theta-join: same as product.
- Natural join: use attributes from each relation; common attributes are merged anyway.
- Renaming: whatever it says.
Example

- Find the bars that are either on Maple Street or sell Bud for less than $3.

\[ \text{Sells}(\text{bar, beer, price}) \cup \text{Bars}(\text{name, addr}) \]
Example

Find the bars that sell two different beers at the same price.

\[ \text{Sells} (\text{bar}, \text{beer}, \text{price}) \]
Linear Notation for Expressions

• Invent new names for intermediate relations, and assign them values that are algebraic expressions.
• Renaming of attributes implicit in schema of new relation.

Example

Find the bars that are either on Maple Street or sell Bud for less than $3.

Sells(bar, beer, price)
Bars(name, addr)

R1(name) := $\pi_{name}(\sigma_{addr = \text{Maple St.}}(\text{Bars}))$
R2(name) := $\pi_{bar}(\sigma_{beer=\text{Bud \ AND \ price}<$3}(\text{Sells}))$
R3(name) := R1 $\cup$ R2
Why Decomposition “Works”?

What does it mean to “work”? Why can’t we just tear sets of attributes apart as we like?

• Answer: the decomposed relations need to represent the same information as the original.
  ◆ We must be able to reconstruct the original from the decomposed relations.

Projection and Join Connect the Original and Decomposed Relations

• Suppose $R$ is decomposed into $S$ and $T$. We project $R$ onto $S$ and onto $T$. 
Example

$$R = \begin{array}{cccc}
\text{name} & \text{addr} & \text{beersLiked} & \text{manf} & \text{favoriteBeer} \\
\text{Janeway} & \text{Voyager} & \text{Bud} & \text{A.B.} & \text{WickedAle} \\
\text{Janeway} & \text{Voyager} & \text{WickedAle} & \text{Pete's} & \text{WickedAle} \\
\text{Spock} & \text{Enterprise} & \text{Bud} & \text{A.B.} & \text{Bud} \\
\end{array}$$

- Recall we decomposed this relation as:

$$R$$

```
Drinkers1

Drinkers2

Drinkers3

Drinkers4
```
Project onto `Drinkers1(name, addr, favoriteBeer)`:

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>favoriteBeer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Bud</td>
</tr>
</tbody>
</table>

Project onto `Drinkers3(beersLiked, manf)`:

<table>
<thead>
<tr>
<th>beersLiked</th>
<th>manf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>A.B.</td>
</tr>
<tr>
<td>WickedAle</td>
<td>Pete's</td>
</tr>
<tr>
<td>Bud</td>
<td>A.B.</td>
</tr>
</tbody>
</table>

Project onto `Drinkers4(name, beersLiked)`:

<table>
<thead>
<tr>
<th>name</th>
<th>addr</th>
<th>beersLiked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>Bud</td>
</tr>
<tr>
<td>Janeway</td>
<td>Voyager</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Spock</td>
<td>Enterprise</td>
<td>Bud</td>
</tr>
</tbody>
</table>
Reconstruction of Original

Can we figure out the original relation from the decomposed relations?

- Sometimes, if we natural join the relations.

Example

\[
\text{Drinkers}_3 \bowtie \text{Drinkers}_4 =
\]

<table>
<thead>
<tr>
<th>name</th>
<th>beersLiked</th>
<th>manf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Bud</td>
<td>A.B.</td>
</tr>
<tr>
<td>Janeway</td>
<td>WickedAle</td>
<td>Pete's</td>
</tr>
<tr>
<td>Spock</td>
<td>Bud</td>
<td>A.B.</td>
</tr>
</tbody>
</table>

- Join of above with \text{Drinkers}_1 = \text{original } R.
Theorem

Suppose we decompose a relation with schema \( XYZ \) into \( XY \) and \( XZ \) and project the relation for \( XYZ \) onto \( XY \) and \( XZ \). Then \( XY \nrightarrow XZ \) is \textit{guaranteed} to reconstruct \( XYZ \) if and only if \( X \rightarrow Y \) (or equivalently, \( X \rightarrow Z \)).

- Usually, the MVD is really a FD, \( X \rightarrow Y \) or \( X \rightarrow Z \).

- BCNF: When we decompose \( XYZ \) into \( XY \) and \( XZ \), it is because there is a FD \( X \rightarrow Y \) or \( X \rightarrow Z \) that violates BCNF.
  - Thus, we can always reconstruct \( XYZ \) from its projections onto \( XY \) and \( XZ \).

- 4NF: when we decompose \( XYZ \) into \( XY \) and \( XZ \), it is because there is an MVD \( X \rightarrow Y \) or \( X \rightarrow Z \) that violates 4NF.
  - Again, we can reconstruct \( XYZ \) from its projections onto \( XY \) and \( XZ \).
Bag Semantics

A relation (in SQL, at least) is really a bag or multiset.

- It may contain the same tuple more than once, although there is no specified order (unlike a list).
- Example: \{1,2,1,3\} is a bag and not a set.
- Select, project, and join work for bags as well as sets.
  - Just work on a tuple-by-tuple basis, and don't eliminate duplicates.
Bag Union

Sum the times an element appears in the two bags.

- Example: \( \{1,2,1\} \cup \{1,2,3,3\} = \{1,1,1,2,2,3,3\} \).

Bag Intersection

Take the minimum of the number of occurrences in each bag.

- Example: \( \{1,2,1\} \cap \{1,2,3,3\} = \{1,2\} \).

Bag Difference

Proper-subtract the number of occurrences in the two bags.

- Example: \( \{1,2,1\} - \{1,2,3,3\} = \{1\} \).
Laws for Bags Differ From Laws for Sets

• Some familiar laws continue to hold for bags.
  ◆ Examples: union and intersection are still commutative and associative.
• But other laws that hold for sets do not hold for bags.

Example

\[ R \cap (S \cup T) \equiv (R \cap S) \cup (R \cap T) \] holds for sets.

• Let \( R, S, \) and \( T \) each be the bag \( \{1\} \).
• Left side: \( S \cup T = \{1,1\} \); \( R \cap (S \cup T) = \{1\} \).
• Right side: \( R \cap S = R \cap T = \{1\}; (R \cap S) \cup (R \cap T) = \{1\} \cup \{1\} = \{1,1\} \neq \{1\}. \)
Extended (“Nonclassical”) Relational Algebra

Adds features needed for SQL, bags.
1. Duplicate-elimination operator $\delta$.
2. Extended projection.
3. Sorting operator $\tau$.
4. Grouping-and-aggregation operator $\gamma$.
5. Outerjoin operator $\bowtie$.
Duplicate Elimination

\[ \delta(R) = \text{relation with one copy of each tuple that appears one or more times in } R. \]

Example

\[ R = \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

\[ \delta(R) = \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Sorting

• \( \tau_L(R) = \) list of tuples of \( R \), ordered according to attributes on list \( L \).

• Note that result type is outside the normal types (set or bag) for relational algebra.
  ◆ Consequence: \( \tau \) cannot be followed by other relational operators.

Example

\[
R = \begin{array}{c|c}
A & B \\
\hline
1 & 3 \\
3 & 4 \\
5 & 2 \\
\end{array}
\]

\( \tau_B(R) = [(5,2), (1,3), (3,4)] \).
Extended Projection

Allow the columns in the projection to be functions of one or more columns in the argument relation.

Example

\[ R = \begin{array}{cc}
A & B \\
1 & 2 \\
3 & 4 \\
\end{array} \]

\[ \pi_{A+B, A, A}(R) = \]

\[ \begin{array}{ccc}
A+B & A1 & A2 \\
3 & 1 & 1 \\
7 & 3 & 3 \\
\end{array} \]
Aggregation Operators

• These are not relational operators; rather they summarize a column in some way.
• Five standard operators: Sum, Average, Count, Min, and Max.
Grouping Operator

$\gamma_L(R)$, where $L$ is a list of elements that are either

a) Individual (*grouping*) attributes or

b) Of the form $\theta(A)$, where $\theta$ is an aggregation operator and $A$ the attribute to which it is applied,

is computed by:

1. Group $R$ according to all the grouping attributes on list $L$.
2. Within each group, compute $\theta(A)$, for each element $\theta(A)$ on list $L$.
3. Result is the relation whose columns consist of one tuple for each group. The components of that tuple are the values associated with each element of $L$ for that group.
Example

Let $R =$

<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>2.00</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>2.75</td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Sue's</td>
<td>Coors</td>
<td>3.00</td>
</tr>
<tr>
<td>Mel's</td>
<td>Miller</td>
<td>3.25</td>
</tr>
</tbody>
</table>

Compute $\gamma_{beer, \text{AVG(price)}}(R)$.

1. Group by the grouping attribute(s), beer in this case:

<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe's</td>
<td>Bud</td>
<td>2.00</td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
<td>2.50</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
<td>2.75</td>
</tr>
<tr>
<td>Mel's</td>
<td>Miller</td>
<td>3.25</td>
</tr>
<tr>
<td>Sue's</td>
<td>Coors</td>
<td>3.00</td>
</tr>
</tbody>
</table>
2. Compute average of price within groups:

<table>
<thead>
<tr>
<th>beer</th>
<th>AVG(price)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bud</td>
<td>2.25</td>
</tr>
<tr>
<td>Miller</td>
<td>3.00</td>
</tr>
<tr>
<td>Coors</td>
<td>3.00</td>
</tr>
</tbody>
</table>
Outerjoin

The normal join can “lose” information, because a tuple that doesn’t join with any from the other relation (dangles) has no vestage in the join result.

• The null value $\perp$ can be used to “pad” dangling tuples so they appear in the join.
• Gives us the outerjoin operator $\bowtie$.
• Variations: theta-outerjoin, left- and right-outerjoin (pad only dangling tuples from the left (respectively, right).
Example

\[ R = \]
\[
\begin{array}{|c|c|}
\hline
A & B \\
\hline
1 & 2 \\
3 & 4 \\
\hline
\end{array}
\]

\[ S = \]
\[
\begin{array}{|c|c|}
\hline
B & C \\
\hline
4 & 5 \\
6 & 7 \\
\hline
\end{array}
\]

\[ R \bowtie S = \]
\[
\begin{array}{|c|c|c|}
\hline
A & B & C \\
\hline
3 & 4 & 5 \\
1 & 2 & \perp \\
\perp & 6 & 7 \\
\hline
\end{array}
\]

- part of natural join
- part of right-outerjoin
- part of left-outerjoin