“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

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

Selection

\[ R_1 = \sigma_C(R_2) \]
where \( C \) is a condition involving the attributes of relation \( R_2 \).

Example

Relation \text{Sells}:

\begin{tabular}{l|l|l}
bar & beer & price \\
\hline
Joe's & Bud & 2.50 \\
Joe's & Miller & 2.75 \\
Sue's & Bud & 2.50 \\
Sue's & Coors & 3.00 \\
\end{tabular}

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

\begin{tabular}{l|l|l}
bar & beer & price \\
\hline
Joe's & Bud & 2.50 \\
Joe's & Miller & 2.75 \\
\end{tabular}
Extra Example Relations

DEPOSIT(branchName, acctNo, custName, balance)
CUSTOMER(custName, street, custCity)
BORROW(branchName, loan-no, custName, amount)
BRANCH(branchName, assets, branchCity)
CLIENT(custName, emplName)

<table>
<thead>
<tr>
<th>Borrow</th>
<th>BN</th>
<th>L#</th>
<th>CN</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

SELECT (σ)

arity(σ(R)) = arity(R)
0 ≤ card(σ(R)) ≤ card(R)

σ_c (R) = σ_c (R) ⊆ (R)
c is selection condition: terms of form: attr op value  attr op attr
op is one of < ≤ > ≥ ≠
example of term: branch-name = 'Midtown'
terms are connected by  ∧ ∨ ¬

σ branchName = 'Midtown' ∧ amount > 1000 (Borrow)
σ custName = emplName (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>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>

• 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 \( j = 1,\ldots,m \)

\[ \pi \quad \text{branchName, custName} \quad \text{(Borrow)} \]

<table>
<thead>
<tr>
<th>Midtown</th>
<th>Fred</th>
</tr>
</thead>
<tbody>
<tr>
<td>Midtown</td>
<td>Sally</td>
</tr>
<tr>
<td>Downtown</td>
<td>Tom</td>
</tr>
</tbody>
</table>
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 \( S \)
Theta-Join

\[ R = R_1 \bowtie_C R_2 \]

is equivalent to \( R = \sigma_c(R_1 \times R_2) \).

Example

<table>
<thead>
<tr>
<th>Sells</th>
<th>Bars</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{bar} )</td>
<td>( \text{beer} )</td>
</tr>
<tr>
<td>Joe's</td>
<td>Bud</td>
</tr>
<tr>
<td>Joe's</td>
<td>Miller</td>
</tr>
<tr>
<td>Sue's</td>
<td>Bud</td>
</tr>
<tr>
<td>Sue's</td>
<td>Coors</td>
</tr>
</tbody>
</table>

BarInfo = Sells \( \bowtie \) Bars

<table>
<thead>
<tr>
<th>BarInfo</th>
<th>( \text{Sells.Bars.Name} )</th>
<th>Bars</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{bar} )</td>
<td>( \text{beer} )</td>
<td>( \text{price} )</td>
</tr>
<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>
Theta-Join: \( R \bowtie_{i \theta j} S \)

- \( \text{arity}(R) = r \)
- \( \text{arity}(S) = s \)
- \( \text{arity}(R \bowtie_{i \theta j} S) = r + s \)

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

\( \theta \) can be \(<\), \(\geq\), \(\leq\), \(\neq\), \(\,=\)

If equal (=), then it is an \textit{EQUIJOIN}.

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

\[
\begin{array}{ccc}
R(ABC) & S(CDE) & T(ABCC'DE) \\
135 & 211 & 135122 \\
246 & 122 & 135334 \\
357 & 334 & 135443 \\
468 & 443 & 246334 \\
\end{array}
\]

Result has schema \(T(A B C C' D E)\)

Natural Join

\( R = R_1 \bowtie_{c} R_2 \)

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 \( \text{Bars} \) was changed to \( \text{bar} \), to match the bar name in \( \text{Sells} \).

\( \text{BarInfo} = \text{Sells} \bowtie_{c} \text{Bars} \)

<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

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>

\[ \rho_{R(\text{bar}, \text{addr})}(\text{Bars}) = \]

<table>
<thead>
<tr>
<th>bar</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>

- The name of the second relation is \( R \).

Set Operations: Union

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

\[ \pi_{\text{Cust-Name}} (\sigma_{\text{Branch-Name} = "Perryridge"} (\text{BORROW} \cup \text{DEPOSIT})) \]
Set Operations: Intersection

**SET INTERSECTION**

- arity(R) = arity(S) = arity \( R \cap S \)
- \( 0 \leq \text{card}(R \cap S) \leq \min(\text{card}(R), \text{card}(S)) \)
- \( \emptyset \subseteq R \cap S \subseteq R \)
- \( \emptyset \subseteq R \cap S \subseteq S \)
- tuples both in R and in S

\( R - (R - S) = R \cap S \)

Set Operations: Difference

**Difference(R \setminus S)**

- arity(R) = arity(S) = arity \( R \setminus S \)
- \( 0 \leq \text{card}(R \setminus S) \leq \text{card}(R) \)
- \( \emptyset \subseteq R \setminus S \subseteq R \)
- is the tuples in R not in S

Depositors of Perryridge who aren’t borrowers of Perryridge

\( \pi_{\text{custName}} \left( \sigma_{\text{branchName} = \text{‘Perryridge’}} \left( \text{DEPOSIT} - \text{BORROW} \right) \right) \)

Deposit: < Perryridge, 36, Pat, 500 >
Borrow: < Perryridge, 72, Pat, 10000 >

\( \pi_{\text{custName}} \left( \sigma_{\text{branchName} = \text{‘Perryridge’}} \left( \text{DEPOSIT} \right) \right) - \pi_{\text{custName}} \left( \sigma_{\text{branchName} = \text{‘Perryridge’}} \left( \text{BORROW} \right) \right) \)

Does \( \sigma \left( \pi \left( \text{D} \right) - \pi \left( \text{B} \right) \right) \) 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.
Operator Precedence

The normal way to group operators is:
1. Unary operators $\sigma$, $\pi$, and $\rho$ have highest precedence.
2. Next highest are the “multiplicative” operators, $\bowtie$, $\bowtie_c$, and $\times$.
3. Lowest are the “additive” operators, $\cup$, $\cap$, and $-$.
   • 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 $R \cup \sigma S \bowtie T$ as $R \cup (\sigma(S) \bowtie T)$.

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.

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

\[ \pi_{\text{name}} \rightarrow \sigma_{\text{addr}=\text{MapleSt.}} \]

\[ \pi_{\text{bar}} \rightarrow \rho_{\text{R}(\text{name})} \]

\[ \rightarrow \sigma_{\text{price}<83 \ AND \ \text{beer=}\text{Bud}} \]

Example

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

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

\[ \pi_{\text{bar}} \rightarrow \sigma_{\text{beer} \neq \text{beer1}} \]

\[ \ni \]

\[ \rho_{S}(\text{bar,beer1,price}) \]

\[ \rightarrow \text{Sells} \leftrightarrow \text{Sells} \]
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=Maple St.}(Bars))
R2(name) := \pi_{bar}(\sigma_{beer=Bud AND price<$3}(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

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

- FDs:
  - name \rightarrow addr
  - name \rightarrow favoriteBeer
  - beersLiked \rightarrow manf
- Decompose:

Project onto \text{Drinkers1}(\text{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 \text{Drinkers3}(\text{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 \text{Drinkers4}(\text{name, beersLiked}):

<table>
<thead>
<tr>
<th>name</th>
<th>beersLiked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Janeway</td>
<td>Bud</td>
</tr>
<tr>
<td>Janeway</td>
<td>WickedAle</td>
</tr>
<tr>
<td>Spock</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: Lossless Join

• Decomposition of \( XYZ \) into \( XY \) and \( XZ \):
  – Let \( XY = \Pi XYZ \); \( XZ = \Pi XYZ \)
  – \( XY \bowtie XZ \) guaranteed to reconstruct \( XYZ \) if and only if \( X \rightarrow Y \)
    • Remember that \( X \rightarrow Z \Rightarrow X \rightarrow Y \)
  – 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\} \ union \ {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\} \ intersect \ {1,2,3,3} = \{1,2\}.

Bag Difference

Proper-subtract the number of occurrences in the two bags.
- Example: \{1,2,1\} \ minus \ {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 \( \Join \).
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

- \[ \tau_L(R) = \text{list of tuples of } R, \text{ ordered according to attributes on list } L. \]
- 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**

\[ R = \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

\[ \tau_B(R) = [(5,2), (1,3), (3,4)]. \]
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.
- Use with grouping (see next slide) or shorthand as "special" projection:

\[ R: A \quad B \\
1 \quad 2 \\
3 \quad 4 \\
\]

\[ \pi_{\text{Max}(A), \text{Min}(B)}(R) = \begin{array}{cc}
\text{Max}(A) & \text{Min}(B) \\
3 & 2 \\
\end{array} \]

- Remember: Aggregations return a single row – can’t combine with non-aggregates in projection
Grouping Operator

\[ \gamma_L(R), \text{ where } L \text{ is a list of elements that are either} \]
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 \( \gamma_{\text{beer}, \text{AVG(price)}}(R) \).
1. Group by the grouping attribute(s), \( \text{beer} \) 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 $\bot$ 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

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

<table>
<thead>
<tr>
<th>$S$ =</th>
<th>$B$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

$$R \bowtie S = \begin{array}{ccc} A & B & C \\ 3 & 4 & 5 \; \text{part of natural join} \\ 1 & 2 & \bot \; \text{part of right-outerjoin} \\ \bot & 6 & 7 \; \text{part of left-outerjoin} \end{array}$$
Division Operator

- Let \( R=XY, S=Y \). Then \( R \div S \) produces a relation \( X \) where
  - \( x \in R \)
  - \( \forall y \in S, xy \in R \)
- Example: Bars that serve everyone’s favorite beers
  - \( \Pi_{\text{bars,beers}}(Sells) \div \Pi_{\text{favoriteBeer}}(Drinkers) \)
- Division isn’t a fundamental operator
  - \( R \div S = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)) \)

“Breaking” the Model

- Some SQL constructs break the traditional relational model
  select bar
  from sells
  where beer in
    (select favorite_beer from drinkers);
- What is the equivalent relational algebra?
  - Why does it break the model?
Relational Algebra

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

Finiteness

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

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 \( \bowtie \).
Relational Calculus

- Two flavors
  - Domain Relational Calculus
  - Tuple Relational Calculus

Tuple Relational Calculus

- Query: \{ t \mid P(t) \}
  - All tuples such that P is true for t
  - t[A] denotes value of attribute t for a
  - t \in r denotes t is in relation r
  - P similar to predicate calculus

- Quantifiers
  - \exists t \in r(Q(t))
    - There is a tuple in r such that Q(t) holds
  - \forall t \in r(Q(t))
    - Q(t) holds for all tuples in r
Domain Relational Calculus

- Query: \( \{ <x_1, \ldots, x_n> | P(x_1, \ldots, x_n) \} \)
  - \( x_i \) are domain variables
  - \( P \) is a predicate

Safety of Expressions

- Calculus expressions meeting certain conditions are “safe”
  - Processing clearly defined
  - Essentially requires that all values in an expression appear in predicate or relation