Relational Algebra

Introduction to databases
CSCC43 Winter 2011
Ryan Johnson

Thanks to Arnold Rosenbloom and Renee Miller
for material in these slides

Why the relational model?

• Sounds good: matches how we think about data
• Real reason: data independence!
• Earlier models tied to physical data layout
  – Procedural access to data (low-level, explicit access)
  – Relationships stored in data (linked lists, trees, etc.)
  – Change in data layout => application rewrite
• Relational model
  – Declarative access to data (system optimizes for you)
  – Relationships specified by queries (schemas help, too)
  – Develop, maintain apps and data layout separately

Similar battle today with languages

What is the relational model?

• Logical representation of data
  – Two-dimensional tables (relations)
• Formal system for manipulating relations
  – Relational algebra
• Result
  – High-level (logical, declarative) description of data
  – Mechanical rules for rewriting/optimizing low-level access
  – Formal methods to reason about soundness

Relational algebra is the key
### Relations and tuples

<table>
<thead>
<tr>
<th>Relation Name</th>
<th>Attribute (column)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Heading (schema)</td>
</tr>
<tr>
<td></td>
<td>Body</td>
</tr>
</tbody>
</table>

#### Tuple (row)

- **Value (field, component)**: Atomic (no sub-tuples)

#### Set-based: arbitrary row/col ordering

**Logical: physical layout might be *very* different!**

### What is an algebra?

- **Operands (values)**
  - Variables, constants
  - Closed domain
- **Operators**
  - “Addition”
  - “Multiplication”
- **Expressions**:
  - Combine operations with parenthesis (explicit)
  - OR using either precedence (implied)
- **Laws**
  - Identify semantically equivalent expressions
  - Commutativity, associativity, etc.

**Allows formal, sound, mechanical rewriting**

### Example algebra: integer arithmetic

- **Domain**: integers
  - ... -100, ... -1, 0, 1, ... 100, ...
- **Operators**: -, +, *
- **Expressions**
  - \((2 * a) + ((5 * (c + (-d))) + e))\)
- **Laws**
  - \(a * b = b * a\) **Commutative**
  - \(a * (b * c) = (a * b) * c\) **Associative**
  - \(a * (b + c) = a * b + a * c\) **Distributive**

**Allows compilers to reason about, optimize**

### Writing expressions

- **Fully parenthesized**
  - \((2 * a) + ((5 * (c + (-d))) + e))\)
- **Operator precedence**
  - \(2 * a + 5 * (c - d) + e\)
- **Linear form**
  - \(x = 2 * a\)
  - \(y = 5 * (c - d)\)
  - \(z = x + y + e\)

**Tree form**

- \(+\)
  - \(e\)
  - \(+\)
    - \(\cdot\)
      - \(2\)
      - \(\cdot\)
        - \(a\)
        - \((-\)
          - \(5\)
          - \((-\)
            - \(c\)
            - \(-\)
              - \(d\))\)
  - \(+\)
    - \(\cdot\)
      - \(x\)
      - \(\cdot\)
        - \(y\)
        - \(\cdot\)
          - \(e\)
Relational algebra

- Values
  - Finite relations (cardinality and arity both bounded)
  - Attributes may or may not be typed

- Operators
  - Unary: \( \sigma, \pi, \rho \)
  - “Additive” (set): \( \cup, \cap, - \)
  - “Multiplicative”: \( \times, \Join \)
  - [details to come]

- Expressions
  - Same as arithmetic, but called “queries”

- Laws
  - Allow “query rewriting”
  - Basis for query optimization
  - [details to come]

Expressive power equivalent to 1st order logic

Unary operators: select (\( \sigma \))

- \( \sigma_P(R) \) outputs tuples of \( R \) which satisfy \( P \)

- Removes unwanted rows from relation

Unary operators: project (\( \pi \))

- \( \pi_{A,B,C}(R) \) outputs attributes \( A,B,C \) of relation \( R \)

- Removes unwanted columns from relation

Unary operators: rename (\( \rho \))

- \( \rho_{S(A,B,C)}(R) \) renames attributes of \( R \) to \( A,B,C \) and calls the result \( S \)
  - \( \rho_S(R) \) renames relation \( R \) (same attributes)
  - \( \rho_{A=X,C=Y}(R) \) renames attributes \( A \) and \( C \) only

- Modifies schema only (body unchanged)
Additive operators ($U$, $\cap$, $\ominus$)

- Standard set operators
- Operate on tuples within input relations

Input schemas must match

Natural join ($\bowtie$)

- $T = R \bowtie S$ merges tuples from $R$ and $S$ having equal values where their schemas overlap
  - Union schemas (as with $\times$) $\text{schema}(R) \cap \text{schema}(S) \neq \emptyset$
  - $|T| \leq |R| \times |S|$, usually $= \max(|R|, |S|)$ “join cardinality”
- Degenerate cases
  - No overlap: $\times$
  - Full overlap: $\cap$

Equivalent to $\pi(\sigma(R \times \rho(S)))$

Comparison: $\cap$ vs. $\bowtie$ vs. $\times$

- Same general operation
  - Test “overlapping” parts of tuples for equality
  - Combine “matching” pairs (ignore others)
- Differing degrees of schema overlap

“Generalized intersection”

Cartesian product ($\times$)

- $T = R \times S$ contains every pairwise combination of tuples from $R$ and $S$
  - $\text{schema}(T) = \text{schema}(R) \cup \text{schema}(S)$
  - $|T| = |R| \times |S|$

Input schemas must not overlap
Mathematical power vs. efficiency

• Note that $\times$ expresses both $\cap$ and $\Join$
  => Mathematically, intersection and joins unnecessary
• Why bother with them? Two big reasons
• Notation
  – $\pi(\sigma(R \times \rho(S)))$ vs. $R \cap S$
  – Cartesian product seldom useful $\quad$ Why not?
• Performance
  – Efficient algorithms compute result directly
  => $|R|^*|S|$ rows vs. $\min(|R|,|S|)$ $\quad$ Consider $|R|=|S|=10^6$

Equijoin

• Written as $R \bowtie_{A=X,B=Y,\ldots} S$
  – Attribute names in R and S can differ
  – Still compare values for equality
• Like natural join, but using arbitrary attributes
  – Very common due to foreign keys in relations
• Equivalent to $R\bowtie\rho(S)$

Theta join

• Written as $T = R \bowtie P S$
  – Outputs pairwise combinations of tuples which satisfy $P$
  – Join cardinality: $|T| \leq |R|^*|S|$
• Most general join
  – Arbitrary join predicate (not just equality)
• Equivalent to $\sigma_P(R \times \rho(S))$
  – Schemas must not overlap
  – Does not project away any attributes $\quad$ Why not?

Limitations of relational algebra

• Relational algebra is set-based
• Real-life applications need more
  – Expensive (and often unnecessary) to eliminate duplicates
  – Important (and often expensive) to order output
  – Need a way to apply scalar expressions to values
  – Database exists to distill data into useful knowledge
  – What’s *not* there often as important as what is

Answer: non-set extensions
Extension: bag semantics

- In practice, relations are bags (multisets)
  - Sometimes people purposefully insert duplicates
  - Projections produce duplicates
- Example: \( \{1,2,1,1,3\} \) is a bag (still unordered!)
- Most operators still work
  - Select, rename unchanged
  - Project no longer eliminates duplicates
  - Set operations need tweaks
  - Joins tend to multiply the number of duplicates
- Some laws no longer apply

Bag versions of set operations

- Union
  - Concatenation (except unordered)
  - \( \{1, 1, 2, 3\} \cup \{2, 2, 3, 4\} = \{1, 2, 3, 2, 2, 3, 4\} \)
- Intersection
  - Take minimum count of each value
  - \( \{1, 1, 2, 3\} \cap \{2, 2, 3, 4\} = \{2, 3\} \)
- Difference
  - Each occurrence on right can cancel one occurrence on left
  - \( \{1, 1, 2, 3\} \setminus \{1, 2, 3, 4\} = \{1\} \)
- Union, intersection no longer distribute
  - \( \{1\} \cap (\{1\} \cup \{1\}) \) vs. \( (\{1\} \cap \{1\}) \cup (\{1\} \cap \{1\}) \)
  - \( \{1\} \cap \{1, 1\} \) vs. \( \{1\} \cup \{1\} \)
  - \( \{1\} \neq \{1,1\} \)

Projection and duplicates

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toyota</td>
<td>Prius</td>
<td>Gray</td>
</tr>
<tr>
<td>Toyota</td>
<td>Prius</td>
<td>Red</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>Green</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>Red</td>
</tr>
<tr>
<td>Ford</td>
<td>Echo</td>
<td>Red</td>
</tr>
<tr>
<td>Ford</td>
<td>Echo</td>
<td>Gray</td>
</tr>
<tr>
<td>Ford</td>
<td>Echo</td>
<td>White</td>
</tr>
</tbody>
</table>

- Consider a relation \( R \) modeling cars for sale
- What does \( \pi_{\text{Make}}(R) \) return?
- What *should* \( \pi_{\text{Make}}(R) \) return?

Duplicate elimination (\( \delta \))

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toyota</td>
<td>Prius</td>
<td>Gray</td>
</tr>
<tr>
<td>Toyota</td>
<td>Prius</td>
<td>Red</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>Green</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>Red</td>
</tr>
<tr>
<td>Ford</td>
<td>Echo</td>
<td>Red</td>
</tr>
<tr>
<td>Ford</td>
<td>Echo</td>
<td>Gray</td>
</tr>
<tr>
<td>Ford</td>
<td>Echo</td>
<td>White</td>
</tr>
</tbody>
</table>

- Consider a relation \( R \) modeling cars for sale
- What does \( \pi_{\text{Make}}(R) \) return?
- \( \delta(\pi_{\text{Make}}(R)) \) is a set

Duplicates important for summaries (“how many”)
Summarizing groups of tuples (1)

<table>
<thead>
<tr>
<th>Student</th>
<th>Year</th>
<th>Dept</th>
<th>Course</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiao</td>
<td>2009</td>
<td>CS</td>
<td>A08</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2009</td>
<td>CS</td>
<td>A48</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2009</td>
<td>CS</td>
<td>A65</td>
<td>B+</td>
</tr>
<tr>
<td>Xiao</td>
<td>2009</td>
<td>Math</td>
<td>A23</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2009</td>
<td>Math</td>
<td>A30</td>
<td>B+</td>
</tr>
<tr>
<td>Xiao</td>
<td>2009</td>
<td>Math</td>
<td>A37</td>
<td>A</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>CS</td>
<td>B07</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>CS</td>
<td>B09</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>CS</td>
<td>B36</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>CS</td>
<td>B58</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>Math</td>
<td>A23</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>Math</td>
<td>A30</td>
<td>B+</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>Math</td>
<td>A37</td>
<td>A</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>Stats</td>
<td>B24</td>
<td>A</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>Stats</td>
<td>B52</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2011</td>
<td>CS</td>
<td>C24</td>
<td>B+</td>
</tr>
<tr>
<td>Xiao</td>
<td>2011</td>
<td>CS</td>
<td>C43</td>
<td>A</td>
</tr>
<tr>
<td>Xiao</td>
<td>2011</td>
<td>CS</td>
<td>C69</td>
<td>A</td>
</tr>
</tbody>
</table>

All courses Xiao has taken

All courses Xiao took in 2010

All math courses Xiao took in 2010

Summarizing groups of tuples (2)

<table>
<thead>
<tr>
<th>Student</th>
<th>Year</th>
<th>Dept</th>
<th>Course</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xiao</td>
<td>2009</td>
<td>CS</td>
<td>A08</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2009</td>
<td>CS</td>
<td>A48</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2009</td>
<td>CS</td>
<td>A65</td>
<td>B+</td>
</tr>
<tr>
<td>Xiao</td>
<td>2009</td>
<td>Math</td>
<td>A23</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2009</td>
<td>Math</td>
<td>A30</td>
<td>B+</td>
</tr>
<tr>
<td>Xiao</td>
<td>2009</td>
<td>Math</td>
<td>A37</td>
<td>A</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>CS</td>
<td>B07</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>CS</td>
<td>B09</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>CS</td>
<td>B36</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>CS</td>
<td>B58</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>Math</td>
<td>A23</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>Math</td>
<td>A30</td>
<td>B+</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>Math</td>
<td>A37</td>
<td>A</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>Stats</td>
<td>B24</td>
<td>A</td>
</tr>
<tr>
<td>Xiao</td>
<td>2010</td>
<td>Stats</td>
<td>B52</td>
<td>B</td>
</tr>
<tr>
<td>Xiao</td>
<td>2011</td>
<td>CS</td>
<td>C24</td>
<td>B+</td>
</tr>
<tr>
<td>Xiao</td>
<td>2011</td>
<td>CS</td>
<td>C43</td>
<td>A</td>
</tr>
<tr>
<td>Xiao</td>
<td>2011</td>
<td>CS</td>
<td>C69</td>
<td>A</td>
</tr>
</tbody>
</table>

How to summarize this??

Show the best grade? Worst grade? Average?

These columns are easy... equal for every tuple in a group

Summarizing groups of tuples (3)

- Description #1: want to output a single tuple which summarizes a set of related tuples
- Description #2: want to “collapse” a set of tuples into a single, “representative” tuple
- Questions
  - How to identify related tuples (set to collapse)?
  - How to collapse a column into a value (summarize it)?
  => Grouping key: a subset of attributes to test for equality
  => Use an aggregation function (sum, count, avg, min, max, ...)

Grouping (Γ)

- Duplicates useful when computing statistics
  - min, max, sum, count, average, ...
- \( \Gamma_{A,B,C,f(X),g(Y),h(Z)}(R) \) computes aggregate values using some attributes as a grouping key
  - Implicit projection (drops unreferenced attributes)
  - \( A, B, C \) is the grouping key
  - \( X, Y, Z \) are attributes to aggregate
  - \( f, g, h \) are aggregating functions to apply

Aggregating function: commutative, associative
- \( f(x,y) = f(y,x) \), \( f(x, f(y, z)) = f(f(x, y), z) \)
Grouping ($\Gamma$)

- All tuples having the same key go to same group
  - One output tuple for each unique key
  - Output “group total” for each non-key attribute in group
  - e.g. $f(x_1, f(x_2, f(x_3, ...)))$

Duplicates and grouping

- Consider a relation $R$ modeling cars for sale
  - $\Gamma_{\text{Make},\text{count}}(R)$ returns?
    - The number of cars of each make

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toyota</td>
<td>Prius</td>
<td>Gray</td>
</tr>
<tr>
<td>Toyota</td>
<td>Prius</td>
<td>Red</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>Green</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>Red</td>
</tr>
<tr>
<td>Honda</td>
<td>Civic</td>
<td>Red</td>
</tr>
<tr>
<td>Ford</td>
<td>Echo</td>
<td>Red</td>
</tr>
<tr>
<td>Ford</td>
<td>Echo</td>
<td>Gray</td>
</tr>
<tr>
<td>Ford</td>
<td>Echo</td>
<td>White</td>
</tr>
</tbody>
</table>

Duplicates important for summaries (“how many”)

The dangling tuple problem

- Consider the following query
  - $\tau_{\text{Total},\text{Total}}(\Gamma_{\text{Name},\text{sum}(\text{Value})}(\text{Emp} \bowtie \text{Sales}))$
  - “List employees and their total sales in descending order”

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Value</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mary</td>
<td>20</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>Xiao</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>Jaspreet</td>
<td>15</td>
<td>35</td>
</tr>
</tbody>
</table>

- Join hides fact that Xiao has no sales!
  - Challenge: rewrite query to include Xiao’s zero

What’s not there can be very important
Extending the “inner” join

- All joins so far resemble intersection
  => Tuples with no match discarded (“dangling”)
- Sometimes desirable to output dangling tuples
  - List all employees and their sales total (even if zero)
- Problem: how to show missing part(s) of tuple?
  - Introduce special value ⊥ (null)
  - Pad dangling tuples as needed to match schema
  - Note: ⊥ technically outside R.A. value domain

Outer join (⋈)

- \( T = R \bowtie S \) computes the “outer” join of \( R \) and \( S \)
  - Like normal join, but all tuples from \( R \) and \( S \) appear in output
  - Pad (left, right, or all) dangling tuples with ⊥
  - \( |T| \geq \max(|R|, |S|) \)

- Natural, equi-, and theta- variants still apply

Outer join in action

- Consider the following query
  - \( \tau_{\text{Total}}(\pi_{\text{Name}},\text{Total}(\Gamma_{\text{Name}},\text{sum}(\text{Value})(\text{Emp } \bowtie \text{Sales}))) \)
  - “List employees and their total sales in descending order”

<table>
<thead>
<tr>
<th>Emp</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>EID</td>
<td>Name</td>
</tr>
<tr>
<td>1</td>
<td>Mary</td>
</tr>
<tr>
<td>2</td>
<td>Xiao</td>
</tr>
<tr>
<td>3</td>
<td>Jaspreet</td>
</tr>
</tbody>
</table>

Extended projection

- \( \pi_{x=E(R)} \) computes column \( x \) from expression \( E \)
  - Arithmetic \((z=3*x + y)\)
  - String manipulation (substring, capitalization)
  - Some conditional expressions

- Example:

<table>
<thead>
<tr>
<th>Emp</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>EID</td>
<td>Name</td>
</tr>
<tr>
<td>1</td>
<td>Mary</td>
</tr>
<tr>
<td>2</td>
<td>Xiao</td>
</tr>
<tr>
<td>3</td>
<td>Jaspreet</td>
</tr>
</tbody>
</table>