Database Systems I

Relational Algebra

Instructor: Ouldooz Baghban Karimi

CMPT 354 - Summer 2019
Relational Algebra (2)

✓ Review Discussed Operators

• More on Derived Operators

• Combining Operators

• Equivalent Expressions

• Operations on Multisets
Relationship Among Operators

- Select: $\sigma$
- Project: $\pi$
- Union: $\bigcup$
- Intersection $\cap$
- Set difference: $-$
- Cartesian product: $\times$
- Join: $\Join$
- Rename: $\rho$
- Division: $/$
Schemas For Results

• **Union, intersection, and difference**: the schemas of the two operands must be the same, so use that schema for the result

• **Selection**: schema of the result is the same as the schema of the operand

• **Projection**: list of attributes tells us the schema.
Schemas for Results

• **Product**: schema is the attributes of both relations. Use R.A, etc., to distinguish two attributes named A.

• **Theta-join**: same as product.

• **Natural join**: union of the attributes of the two relations.

• **Renaming**: the operator tells the schema
Derived Operators

• Intersection $\cap$

• Join: $\Join$
  • Natural Join
  • Theta Join
  • Equijoin
  • Inner Join
  • Outer Join

• Rename: $\rho$

• Divide: $/$
Intersection

• $R \cap S = R - (R - S)$

  • First subtracting $S$ from $R$ to form a relation $T$ consisting of all those tuples in $R$ but not $S$.

  • Subtract $T$ from $R$, leaving only those tuples of $R$ that are also in $S$. 
Relational Algebra (2)

• Review Discussed Operators

✓ More on Derived Operators

• Combining Operators

• Equivalent Expressions

• Operations on Multisets
Division

• R/S
  • Find tuples in R that match all tuples in another relation
    \[ \frac{R}{S} = \pi_{(R-S)}(R) - \pi_{(R-S)}[(\pi_{(R-S)}(R) \times S) - R] \]

• R relation with attributes \( A_1, \ldots, A_n, B_1, \ldots, B_m \)
• S relation with attributes \( B_1, \ldots, B_m \) (a subset of R’s attributes)
• \( \frac{R}{S} \) with attributes \( A_1, \ldots, A_n \) is a relation with attributes that consist of all tuples that
  • For every tuple with attributes \( A_1, \ldots, A_n \) concatenated with a tuple with attributes \( B_1, \ldots, B_m \) with values in S the concatenated tuple with attributes \( A_1, \ldots, A_n, B_1, \ldots, B_m \) is in R
Division Example

The relation $R$:

<table>
<thead>
<tr>
<th>Student</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Database1</td>
</tr>
<tr>
<td>Fred</td>
<td>Database2</td>
</tr>
<tr>
<td>Fred</td>
<td>Compiler1</td>
</tr>
<tr>
<td>Eugene</td>
<td>Database1</td>
</tr>
<tr>
<td>Eugene</td>
<td>Compiler1</td>
</tr>
<tr>
<td>Sarah</td>
<td>Database1</td>
</tr>
<tr>
<td>Sarah</td>
<td>Database2</td>
</tr>
</tbody>
</table>

The relation $S$:

<table>
<thead>
<tr>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database1</td>
</tr>
<tr>
<td>Database2</td>
</tr>
</tbody>
</table>

The relation $R/S$:

<table>
<thead>
<tr>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
</tr>
<tr>
<td>Sarah</td>
</tr>
</tbody>
</table>
Division Derivation In Detail

\[ \frac{R}{S} = \pi_{(R-S)}(R) - \pi_{(R-S)} \left[ (\pi_{(R-S)}(R) \times S) - R \right] \]

- Start with computing a table with attributes \(A_1, \ldots, A_n\) from \(R\): it will be \(\pi_{(R-S)}(R)\)

- Then find the product of that table with table \(S\): it will be \((\pi_{(R-S)}(R) \times S)\)

- This \((\pi_{(R-S)}(R) \times S)\) table will have attributes \(A_1, \ldots, A_n, B_1, \ldots, B_m\) as a result of the product. It’s attributes \(B_1, \ldots, B_m\) will have values from \(S\)
Division Derivation In Detail

• Then we remove from that table, any tuple that is in initial table $R$: it will be $[((\pi_{(R-S)}(R) \times S) – R]$

• Finally, we remove all tuples $A_I,..., A_n$ that are in that product, but not in the $A_I,..., A_n$ of $R$, from the $R$’s projection of $A_I,..., A_n$ with $R/S = \pi_{(R-S)}(R) – \pi_{(R-S)} [((\pi_{(R-S)}(R) \times S) – R]$

• The last two steps are similar to derivation of intersect
Derived Division Example

The relation $R$:

<table>
<thead>
<tr>
<th>Student</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Database1</td>
</tr>
<tr>
<td>Fred</td>
<td>Database2</td>
</tr>
<tr>
<td>Fred</td>
<td>Compiler1</td>
</tr>
<tr>
<td>Eugene</td>
<td>Database1</td>
</tr>
<tr>
<td>Eugene</td>
<td>Compiler1</td>
</tr>
<tr>
<td>Sarah</td>
<td>Database1</td>
</tr>
<tr>
<td>Sarah</td>
<td>Database2</td>
</tr>
</tbody>
</table>

The relation $S$:

<table>
<thead>
<tr>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database1</td>
</tr>
<tr>
<td>Database2</td>
</tr>
</tbody>
</table>

1. The relation $\pi_{(R-S)}(R)$

<table>
<thead>
<tr>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
</tr>
<tr>
<td>Eugene</td>
</tr>
<tr>
<td>Sarah</td>
</tr>
</tbody>
</table>

2. The relation $(\pi_{(R-S)}(R) \times S)$

<table>
<thead>
<tr>
<th>Student</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Database1</td>
</tr>
<tr>
<td>Fred</td>
<td>Database2</td>
</tr>
<tr>
<td>Eugene</td>
<td>Database1</td>
</tr>
<tr>
<td>Eugene</td>
<td>Database2</td>
</tr>
<tr>
<td>Sarah</td>
<td>Database1</td>
</tr>
<tr>
<td>Sarah</td>
<td>Database2</td>
</tr>
</tbody>
</table>
Derived Division Example

3. \[ (\pi_{(R-S)}(R) \times S) - R \]

<table>
<thead>
<tr>
<th>Student</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eugene</td>
<td>Database2</td>
</tr>
</tbody>
</table>

4. \( \pi_{(R-S)} [(\pi_{(R-S)}(R) \times S) - R] \)

<table>
<thead>
<tr>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eugene</td>
</tr>
</tbody>
</table>

5. \( \pi_{(R-S)}(R) - \pi_{(R-S)} [(\pi_{(R-S)}(R) \times S) - R] \)

<table>
<thead>
<tr>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
</tr>
<tr>
<td>Eugene</td>
</tr>
<tr>
<td>Sarah</td>
</tr>
</tbody>
</table>

6. \( R/S = \pi_{(R-S)}(R) - \pi_{(R-S)} [(\pi_{(R-S)}(R) \times S) - R] \)

<table>
<thead>
<tr>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
</tr>
<tr>
<td>Sarah</td>
</tr>
</tbody>
</table>
Theta Join

\[ R \bowtie_C S = \sigma_C (R \times S) \]

- Start with the product \( R \times S \)
- Then apply the selection operator with a condition \( C \)
Natural Join

• $R \bowtie S = \pi_L(\sigma_C (R \times S))$

  • Start with the product $R \times S$

  • Then apply the selection operator with a condition $C$ of the form

    $R. A_1 = S. A_1 \ AND \ R. A_2 = S. A_2 \ AND \ R. A_3 = S. A_3 \ldots \ AND \ R. A_n = S. A_n$

    where $A_1,\ldots, A_n$ are all the attributes appearing in the schemas of both $R$ and $S$

  • Then we project out one copy of each of the equated attributes.

    $L$: list of attributes in $R$ followed by attributes in $S$ that are not in $R$
Equijoin

• Theta join
• Join condition C consists only of equalities

\[ R \bowtie_C S = \sigma_C (R \times S) \]

• Start with the product \( R \times S \)
• Then apply the selection operator with a condition \( C \)
• \( C \) consists only of equalities
Joins

• Inner Join
  • Includes only those tuples with matching attributes and the rest are discarded in the resulting relation
    • Natural Join
    • Theta Join
    • Equijoin

• Outer Join
  • Include unmatched tuples from the participating relations
    • Full outer join
    • Left outer join
    • Right outer join
Outer Join

• The outer join (full outer join) $R \bowtie^{outer}_C S$

  • Schema contains union of the (possibly renamed) attributes in $R$ and $S$

  • Tuples consist of
    • The tuples in the regular join
    • The tuples of $R$ that do not join with any tuple in $S$, and padded with NULL instead
    • The tuples of $S$ that do not join with any tuple in $R$, and padded with NULL instead
Left Outer Join

• Left outer join $R \bowtie_{C} S$

  • Schema contains union of the (possibly renamed) attributes in $R$ and $S$

  • Tuples consist of
    • The tuples in the regular join
    • The tuples of $R$ that do not join with any tuple in $S$, and padded with NULL instead
Right Outer Join

- Right outer join $R \bowtie^\text{right} C S$

- Schema contains union of the (possibly renamed) attributes in $R$ and $S$

- Tuples consist of
  - The tuples in the regular join
  - The tuples of $S$ that do not join with any tuple in $R$, and padded with NULL instead
Outer Join Example

The relation $R$:

<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>

The relation $S$:

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

The relation $R \bowtie_{R.B=S.B}^\text{outer} S$:

<table>
<thead>
<tr>
<th>A</th>
<th>R.B</th>
<th>S.B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
## Left Outer Join Example

The relation $R$:

<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>

The relation $S$:

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

The relation $R \bowtie_{R.B=S.B}^{left} S$:

<table>
<thead>
<tr>
<th>A</th>
<th>R.B</th>
<th>S.B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Right Outer Join Example

The relation \( R \):  

<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>

The relation \( S \):  

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

The relation \( R \bowtie_{R.B=S.B}^\text{right} S \):  

<table>
<thead>
<tr>
<th>A</th>
<th>R.B</th>
<th>S.B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Relational Algebra as a Constraint Language

• Two ways to express
  • \( R \) is an expression of relational algebra
    \( R = 0 \) is a constraint
    **Meaning:** The value of \( R \) must be empty (no tuples in the result of \( R \))

  • \( R \) and \( S \) are expressions of relational algebra
    \( R \subseteq S \) is a constraint
    **Meaning:** Every tuple in the result of \( R \) must also be in the result of \( S \)
    The result of \( S \) may contain additional tuples not produced by \( R \).
Referential Integrity Constraints

• Asserting a value appearing in one context also appears in another, related context

• Example
  • Movies(title, year, length, genre, studioName, producerC#)
  • StarsIn(movieTitle, movieYear, starName)

• If we want to assert that any movie mentioned in the relation StartsIn also appears in the relation Movies

\[ \pi_{\text{movieTitle, movieYear}}(\text{StarsIn}) \subseteq \pi_{\text{title, year}}(\text{Movies}) \]
Key Constraint

• Example
  • The relation movie star with name attribute as key
    \text{MovieStar}(\text{name, address, gender, birthdate})

  • If we construct all pairs of \text{MovieStar} tuples \((t_1, t_2)\), we must not find a pair that agree in the name component and disagree in address components
  • To construct the pairs we use a Cartesian product
  • To search for pairs that violate the condition we use a selection
  • We then assert the constraint by equating the result to 0.
Key Constraint

$$\sigma_{MS1.name=MS2.name \text{ AND } MS1.address\neq MS2.address}(MS1 \times MS2)=0$$

- MS1 in the product $MS1 \times MS2$ is shorthand for the renaming:
  $$\rho_{MS1(name, address, gender, birthdate)}(MovieStar)$$
Additional Operators

• There are additional relational algebra operators
  • Usually used in the context of **query optimization**

• Duplicate elimination – $\delta$
  • Used to turn a bag into a set

• Aggregation operators
  • e.g. sum, average

• Grouping – $\gamma$
  • Used to partition tuples into groups
    • Typically used with aggregation
Relational Algebra (2)

• Review Discussed Operators

• More on Derived Operators

✓ Combining Operators

• Equivalent Expressions

• Operations on Multisets
Combining Operators to Form Queries

• Expressions by applying operations to the result of other operations.
  • Parentheses and precedence rules

• Three notations
  • Expression trees
  • Expressions with several operators
  • Sequences of assignment statements
Expressions

• Example

What are the titles and years of movies made by **Fox** that are at least **100** minutes long?

1. Select those Movies tuples that have `length>= 100`

2. Select those Movies tuples that have `studioName = ‘Fox’`

3. Compute the intersection of (1) and (2)

4. Project the relation from (3) onto attributes title and year
Tree Expression

• Evaluated bottom-up
  • Applying the operator at an interior node to the arguments, which are the results of its children.
• Example
  • The two selection nodes: steps (1) and (2)
  • The intersection node: step (3)
  • The projection node is step (4)
Linear Expression

• More than one relational algebra expression that represents the same computation

\[ \pi_{\text{title,year}}\left(\sigma_{\text{lengh} \geq 100}(Movies) \cap \sigma_{\text{studioName} = \text{‘Fox’}}(Movies)\right) \]

\[ \pi_{\text{title,year}}\left(\sigma_{\text{lengh} \geq 100 \ \text{AND} \ \text{studioName} = \text{‘Fox’}}(Movies)\right) \]
Linear Notation

• Names for temporary relations corresponding to interior nodes of the tree

• Sequence of assignments that create value for each temporary relation

• Flexible order as long as value ready
Linear Notation

• A relation name and parenthesized list of attributes for that relation. The name answer will be used conventionally for the result of the final step
  • i.e. the name of the relation at the root of the expression tree

• The assignment symbol ( := )

• Any algebraic expression on the right
Linear Notation

• \( R(t,y,l,i,s,p) := \sigma_{\text{length} \geq 100}(\text{Movies}) \)

• \( S(t,y,l,i,s,p) := \sigma_{\text{studioName} = 'Fox'}(\text{Movies}) \)

• \( \text{Answer}(\text{title}, \text{year}) := \pi_{t,y}(R \cap S) \)
Relational Algebra (2)

• Review Discussed Operators

• More on Derived Operators

• Combining Operators

✓ Equivalent Expressions

• Operations on Multisets
Equivalent Expressions & Query Optimization

• Query based on expressions similar to relational algebra

• Query asked may have many equivalent expressions
  • Some much more quickly evaluated

• Query optimization: replace one expression of relational algebra by an equivalent expression that is more efficiently evaluated
Equivalent Expressions & Query Optimization

• Semantic equivalence: results are always the same

\[ \pi_{\text{title,year}}(\sigma_{\text{length} \geq 100}(\text{Movies}) \cap \sigma_{\text{studioName} = 'Fox'}(\text{Movies})) \]

\[ \pi_{\text{title,year}}(\sigma_{\text{length} \geq 100 \ \text{AND} \ \text{studioName} = 'Fox'}(\text{Movies})) \]

• Questions to ask
  • Are they equivalent?
  • Which one is more efficient?
  • Can you make it even more efficient?
Relational Algebra (2)

- Review Discussed Operators
- More on Derived Operators
- Combining Operators
- Equivalent Expressions

✓ Operations on Multisets
Operations on Multisets

• A bag (or multiset) is like a set, but an element may appear more than once.
  
• Example: \(\{1,2,1,3\}\) is a bag.
• Example: \(\{1,2,3\}\) is also a bag that happens to be a set.
Operations on Multisets

• SQL, the most important query language for relational databases, is actually a bag language.

• Some operations, like projection, are more efficient on bags than sets
Operations on Multisets

• Selection applies to each tuple, so its effect on bags is like its effect on sets.

• Projection also applies to each tuple, but as a bag operator, we do not eliminate duplicates.

• Products and joins are done on each pair of tuples, so duplicates in bags have no effect on how we operate.
Bag Union

• An element appears in the union of two bags the sum of the number of times it appears in each bag.

• Example: \( \{1,2,1\} \cup \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\} \)
Bag Intersection

• An element appears in the intersection of two bags the minimum of the number of times it appears in either.

• Example: \( \{1,2,1,1\} \cap \{1,2,1,3\} = \{1,1,2\} \).
Bag Difference

• An element appears in the difference $A - B$ of bags as many times as it appears in $A$, minus the number of times it appears in $B$.

• But never less than 0 times.

• Example: $\{1,2,1,1\} - \{1,2,3\} = \{1,1\}$. 
Bag Laws are not Set Laws

• Some, but not all algebraic laws that hold for sets also hold for bags.

• Example: the commutative law for union \((R \cup S = S \cup R)\) does hold for bags.

• Since addition is commutative, adding the number of times \(x\) appears in \(R\) and \(S\) does not depend on the order of \(R\) and \(S\).
Law that Fails

• Set union is idempotent, meaning that $S \cup S = S$.

• However, for bags, if $x$ appears $n$ times in $S$, then it appears $2n$ times in $S \cup S$.

• Thus $S \cup S \neq S$ in general.

• e.g., $\{1\} \cup \{1\} = \{1,1\} \neq \{1\}$. 
Summary

• Relational Algebra operators
  • Five core operators: selection, projection, cross-product, union and set difference
  • Additional operators are defined in terms of the core operators
    • E.g. Intersection, join, rename

• SQL and Relational Algebra can express the same class of queries

• Multiple Relational Algebra queries can be equivalent
  • Same semantics but difference performance
  • Form basis for optimizations
Acknowledgements

I have used materials from the following resources in preparation of this course:

• Database Systems: The Complete Book (and slides)
• Database Systems (Kifer, Bernstein, Lewis)
• Database System Concepts: https://www.db-book.com
• Course offerings
  • W 4111 (Eugene Wu - Columbia): https://w4111.github.io/
  • CS 245 (Matei Zaharia - Stanford): http://web.stanford.edu/class/cs245/
  • CS 186 (Joe Hellerstein - Berkeley): https://sites.google.com/site/cs186fall17/