Lecture 4 of 42

Relational Joins

Wednesday, 30 January 2008

William H. Hsu
Department of Computing and Information Sciences, KSU

KSOL course page: http://snipurl.com/va60
Course web site: http://www.kddresearch.org/Courses/Spring-2008/CIS560
Instructor home page: http://www.cis.ksu.edu/~bhsu

Reading for Next Class:
Chapter 2, Silberschatz et al., 5th edition

Chapter 2: Relational Model

- Structure of Relational Databases
- Fundamental Relational-Algebra-Operations
- Additional Relational-Algebra-Operations
- Extended Relational-Algebra-Operations
- Null Values
- Modification of the Database
Relational Algebra: Review

- Procedural language
- Six basic operators
  - select: \( \sigma \)
  - project: \( \Pi \)
  - union: \( \cup \)
  - set difference: \( - \)
  - Cartesian product: \( \times \)
  - rename: \( \rho \)
- The operators take one or two relations as inputs and produce a new relation as a result.

Review: Finding Max using Self-Join

- Find the largest account balance

  **Strategy:**
  - Find those balances that are *not* the largest
    - Rename account relation as \( d \) so that we can compare each account balance with all others
  - Use set difference to find those account balances that were not found in the earlier step.
  - The query is:

\[
\Pi_{balance}(account) - \Pi_{account.balance} \\
(\sigma_{account.balance < d.balance}(account \times \rho_{d}(account)))
\]
Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let $E_1$ and $E_2$ be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_P(E_1)$, $P$ is a predicate on attributes in $E_1$
  - $\Pi_S(E_1)$, $S$ is a list consisting of some of the attributes in $E_1$
  - $\rho_x(E_1)$, $x$ is the new name for the result of $E_1$

Additional Operations

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

- Set intersection
- Natural join
- Division
- Assignment
Set-Intersection Operation

- Notation: \( r \cap s \)
- Defined as:
  \[ r \cap s = \{ t \mid t \in r \text{ and } t \in s \} \]
- Assume:
  - \( r, s \) have the same arity
  - attributes of \( r \) and \( s \) are compatible
- Note: \( r \cap s = r - (r - s) \)

Set-Intersection Operation – Example

- Relation \( r, s \):
  \[
  \begin{array}{ccc}
  A & B \\
  \alpha & 1 \\
  \alpha & 2 \\
  \beta & 1 \\
  \end{array}
  \quad \begin{array}{ccc}
  A & B \\
  \alpha & 2 \\
  \beta & 3 \\
  \end{array}
  \]
  \( r \)
  \( s \)

- \( r \cap s \)
  \[
  \begin{array}{ccc}
  A & B \\
  \alpha & 2 \\
  \end{array}
  \]

- Assume:
  - \( r, s \) have the same arity
  - attributes of \( r \) and \( s \) are compatible
  - Note: \( r \cap s = r - (r - s) \)
Natural-Join Operation

- Notation: \( r \bowtie s \)
- Let \( r \) and \( s \) be relations on schemas \( R \) and \( S \) respectively. Then, \( r \bowtie s \) is a relation on schema \( R \cup S \) obtained as follows:
  - Consider each pair of tuples \( t_r \) from \( r \) and \( t_s \) from \( s \).
  - If \( t_r \) and \( t_s \) have the same value on each of the attributes in \( R \cap S \), add a tuple \( t \) to the result, where
    - \( t \) has the same value as \( t_r \) on \( r \)
    - \( t \) has the same value as \( t_s \) on \( s \)

- Example:
  - \( R = (A, B, C, D) \)
  - \( S = (E, B, D) \)
  - Result schema = \( (A, B, C, D, E) \)
  - \( r \bowtie s \) is defined as:
    \[ \Pi_{r,A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \land r.D = s.D} (r \times s)) \]
Division Operation

- Notation: \( r \div s \)
- Suited to queries that include the phrase “for all”. 
- Let \( r \) and \( s \) be relations on schemas \( R \) and \( S \) respectively where
  - \( R = (A_1, \ldots, A_m, B_1, \ldots, B_n) \)
  - \( S = (B_1, \ldots, B_n) \)

The result of \( r \div s \) is a relation on schema \( R - S = (A_1, \ldots, A_m) \)

\[
r \div s = \{ t \mid t \in \Pi_{R-S}(r) \land \forall u \in s \ (tu \in r) \}
\]

Where \( tu \) means the concatenation of tuples \( t \) and \( u \) to produce a single tuple

Division Operation – Example

- Relations \( r, s \):

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>1</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>2</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>3</td>
</tr>
<tr>
<td>( \beta )</td>
<td>1</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>1</td>
</tr>
<tr>
<td>( \delta )</td>
<td>1</td>
</tr>
<tr>
<td>( \delta )</td>
<td>3</td>
</tr>
<tr>
<td>( \varepsilon )</td>
<td>4</td>
</tr>
<tr>
<td>( \varepsilon )</td>
<td>6</td>
</tr>
<tr>
<td>( \beta )</td>
<td>2</td>
</tr>
</tbody>
</table>

- \( r \div s \):

<table>
<thead>
<tr>
<th>( A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
</tr>
<tr>
<td>( \beta )</td>
</tr>
</tbody>
</table>

\( s \)

\( r \)

\( r \div s \)
Another Division Example

- Relations \( r, s \):

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>a</td>
<td>α</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>α</td>
<td>a</td>
<td>γ</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>α</td>
<td>a</td>
<td>γ</td>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>β</td>
<td>a</td>
<td>γ</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>β</td>
<td>a</td>
<td>γ</td>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>γ</td>
<td>a</td>
<td>γ</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>γ</td>
<td>a</td>
<td>γ</td>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>γ</td>
<td>a</td>
<td>β</td>
<td>b</td>
<td>1</td>
</tr>
</tbody>
</table>

\( r \)

- \( r \div s \):

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>a</td>
<td>γ</td>
</tr>
<tr>
<td>γ</td>
<td>a</td>
<td>γ</td>
</tr>
</tbody>
</table>

Division Operation (Cont.)

- Property
  - Let \( q = r \div s \)
  - Then \( q \) is the largest relation satisfying \( q \times s \subseteq r \)
- Definition in terms of the basic algebra operation
  Let \( r(R) \) and \( s(S) \) be relations, and let \( S \subseteq R \)

\[
 r \div s = \Pi_{R \setminus S} (r) - \Pi_{R \setminus S} ( ( \Pi_{R \setminus S} (r) \times s ) - \Pi_{R \setminus S \cup S} (r) )
\]

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

- The assignment operation (←) provides a convenient way to express complex queries.
- Write query as a sequential program consisting of
  - a series of assignments
  - followed by an expression whose value is displayed as a result of the query.
- Assignment must always be made to a temporary relation variable.
- Example: Write \( r \div s \) as

  \[
  \begin{align*}
  temp1 & \leftarrow \Pi_{R,S}(r) \\
  temp2 & \leftarrow (\Pi_{R,S}(temp1 \times s) - \Pi_{R,S,S}(r)) \\
  result & = temp1 - temp2
  \end{align*}
  \]

- The result to the right of the ← is assigned to the relation variable on the left of the ←.
- May use variable in subsequent expressions.

Bank Example Queries

- Find the names of all customers who have a loan and an account at bank.

  \[
  \Pi_{\text{customer\_name}}(\text{borrower}) \cap \Pi_{\text{customer\_name}}(\text{depositor})
  \]

- Find the name of all customers who have a loan at the bank and the loan amount

  \[
  \Pi_{\text{customer\_name}, \text{loan\_number}, \text{amount}}(\text{borrower} \bowtie \text{loan})
  \]
Bank Example Queries

- Find all customers who have an account from at least the “Downtown” and the Uptown” branches.
- Query 1
  \[
  \Pi_{\text{customer\_name}}(\sigma_{\text{branch\_name} = \text{"Downtown"}}(\text{depositor} \bowtie \text{account})) \cap \\
  \Pi_{\text{customer\_name}}(\sigma_{\text{branch\_name} = \text{"Uptown"}}(\text{depositor} \bowtie \text{account}))
  \]
- Query 2
  \[
  \Pi_{\text{customer\_name}, \text{branch\_name}}(\text{depositor} \bowtie \text{account}) \\
  \div \Pi_{\text{branch\_name}}(\sigma_{\text{branch\_city} = \text{"Brooklyn"}}(\text{branch}))
  \]
  Note that Query 2 uses a constant relation.

Example Queries

- Find all customers who have an account at all branches located in Brooklyn city.
  \[
  \Pi_{\text{customer\_name}, \text{branch\_name}}(\text{depositor} \bowtie \text{account}) \\
  \div \Pi_{\text{branch\_name}}(\sigma_{\text{branch\_city} = \text{"Brooklyn"}}(\text{branch}))
  \]
Extended Relational-Algebra-Operations

- Generalized Projection
- Aggregate Functions
- Outer Join

Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

\[ \Pi_{F_1, F_2, \ldots, F_n}(E) \]

- \( E \) is any relational-algebra expression
- Each of \( F_1, F_2, \ldots, F_n \) are arithmetic expressions involving constants and attributes in the schema of \( E \).
- Given relation \( \text{credit\_info}(\text{customer\_name}, \text{limit}, \text{credit\_balance}) \), find how much more each person can spend:

\[ \Pi_{\text{customer\_name}, \text{limit} - \text{credit\_balance}}(\text{credit\_info}) \]
Aggregate Functions and Operations

- **Aggregation 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
  \[
  G_1, G_2, \ldots, G_n \left( F_1(A_1), F_2(A_2), \ldots, F_n(A_n) \right) E
  \]

  - \( E \) is any relational-algebra expression
  - \( G_1, G_2, \ldots, G_n \) is a list of attributes on which to group (can be empty)
  - Each \( F_j \) is an aggregate function
  - Each \( A_i \) is an attribute name

Aggregate Operation – Example

- Relation \( r \):
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>α</td>
<td>7</td>
</tr>
<tr>
<td>α</td>
<td>β</td>
<td>7</td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>3</td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>10</td>
</tr>
</tbody>
</table>

- \( g_{\text{sum}(c)}(r) \)
  
<table>
<thead>
<tr>
<th>sum(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
</tr>
</tbody>
</table>
Aggregate Operation – Example

- Relation account grouped by branch-name:

<table>
<thead>
<tr>
<th>branch_name</th>
<th>account_number</th>
<th>balance</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>branch_name</th>
<th>sum(balance)</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

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

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses null values:
  - null signifies that the value is unknown or does not exist
  - All comparisons involving null are (roughly speaking) false by definition.
  - We shall study precise meaning of comparisons with nulls later

Outer Join – Example

- Relation loan

<table>
<thead>
<tr>
<th>loan_number</th>
<th>branch_name</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-170</td>
<td>Downtown</td>
<td>3000</td>
</tr>
<tr>
<td>L-230</td>
<td>Redwood</td>
<td>4000</td>
</tr>
<tr>
<td>L-260</td>
<td>Perryridge</td>
<td>1700</td>
</tr>
</tbody>
</table>

- Relation borrower

<table>
<thead>
<tr>
<th>customer_name</th>
<th>loan_number</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>