Lecture 20:
Query Execution: Relational Algebra

Wednesday, May 16, 2007
DBMS Architecture

How does a SQL engine work?

- SQL query → relational algebra plan
- Relational algebra plan → Optimized plan
- Execute each operator of the plan
Relational Algebra

• Formalism for creating new relations from existing ones

• Its place in the big picture:

![Diagram showing the relationship between declarative query language, algebra, and implementation. SQL, relational calculus, relational algebra, and Relational bag algebra are connected in the diagram.]
Relational Algebra

- Five operators:
  - Union: $\cup$
  - Difference: $-$
  - Selection: $\sigma$
  - Projection: $\Pi$
  - Cartesian Product: $\times$

- Derived or auxiliary operators:
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
  - Renaming: $\rho$
1. Union and 2. Difference

- $R_1 \cup R_2$
- Example:
  - ActiveEmployees $\cup$ RetiredEmployees

- $R_1 - R_2$
- Example:
  - AllEmployees -- RetiredEmployees
What about Intersection?

• It is a derived operator
• $R_1 \cap R_2 = R_1 - (R_1 - R_2)$
• Also expressed as a join (will see later)
• Example
  – $\text{UnionizedEmployees} \cap \text{RetiredEmployee}$
3. Selection

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Examples
  - $\sigma_{\text{Salary} > 40000}(\text{Employee})$
  - $\sigma_{\text{name} = \text{“Smith”}}(\text{Employee})$
- The condition $c$ can be $=, <, \leq, >, \geq, <>$
\[ \sigma_{\text{Salary} > 40000} (\text{Employee}) \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>2000000</td>
</tr>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>6000000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>5000000</td>
</tr>
</tbody>
</table>
4. Projection

• Eliminates columns, then removes duplicates
• Notation: $\Pi_{A_1, \ldots, A_n}(R)$
• Example: project social-security number and names:
  – $\Pi_{\text{SSN, Name}}(\text{Employee})$
  – Output schema: Answer(\text{SSN, Name})
\[
\Pi_{\text{Name},\text{Salary}}(\text{Employee})
\]

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>200000</td>
</tr>
<tr>
<td>5423341</td>
<td>John</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>John</td>
<td>200000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>200000</td>
</tr>
<tr>
<td>John</td>
<td>600000</td>
</tr>
</tbody>
</table>
5. Cartesian Product

- Each tuple in R1 with each tuple in R2
- Notation: $R_1 \times R_2$
- Example:
  - Employee $\times$ Dependents
- Very rare in practice; mainly used to express joins
## Cartesian Product Example

### Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>

### Dependents

<table>
<thead>
<tr>
<th>EmployeeSSN</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

### Employee x Dependents

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmployeeSSN</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>9999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>9999999999</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>
Relational Algebra

• Five operators:
  – Union: ∪
  – Difference: -
  – Selection: σ
  – Projection: Π
  – Cartesian Product: ×

• Derived or auxiliary operators:
  – Intersection, complement
  – Joins (natural,equi-join, theta join, semi-join)
  – Renaming: ρ
Renaming

• Changes the schema, not the instance
• Notation: \( \rho_{B_1,\ldots,B_n}(R) \)
• Example:
  
  – \( \rho_{\text{LastName}, \text{SocSocNo}}(\text{Employee}) \)
  – Output schema: \( \text{Answer(LastName, SocSocNo)} \)
## Renaming Example

<table>
<thead>
<tr>
<th>Employee</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>SSN</td>
</tr>
<tr>
<td>John</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>

$$\rho_{LastName, SocSocNo} (Employee)$$

<table>
<thead>
<tr>
<th>LastName</th>
<th>SocSocNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>
Natural Join

• Notation: $R_1 \times R_2$

• Meaning: $R_1 \times R_2 = \Pi_A(\sigma_C(R_1 \times R_2))$

• Where:
  – The selection $\sigma_C$ checks equality of all common attributes
  – The projection eliminates the duplicate common attributes
### Natural Join Example

#### Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>

#### Dependents

<table>
<thead>
<tr>
<th>SSN</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

\[
\text{Employee} \Join \text{Dependents} = \\
\Pi_{\text{Name}, \text{SSN}, \text{Dname}}(\sigma_{\text{SSN}=\text{SSN}_2}(\text{Employee} \times \rho_{\text{SSN}_2, \text{Dname}}(\text{Dependents})))
\]

| Name   | SSN          | Dname |
|--------|--------------|
| John   | 9999999999   | Emily |
| Tony   | 7777777777   | Joe   |
Natural Join

- **R=**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
</tr>
</tbody>
</table>

- **S=**

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
</tr>
</tbody>
</table>

- **R \times S=**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
<td>W</td>
</tr>
</tbody>
</table>
Natural Join

• Given the schemas $R(A, B, C, D)$, $S(A, C, E)$, what is the schema of $R \times S$ ?

• Given $R(A, B, C)$, $S(D, E)$, what is $R \times S$ ?

• Given $R(A, B)$, $S(A, B)$, what is $R \times S$ ?
 Theta Join

- A join that involves a predicate
- $R_1 \ |\times|_\theta \ R_2 = \sigma_\theta (R_1 \times R_2)$
- Here $\theta$ can be any condition
Eq-join

- A theta join where $\theta$ is an equality
- $R_1 \mid \times \mid_{A=B} R_2 = \sigma_{A=B} (R_1 \times R_2)$
- Example:
  - Employee $\mid \times \mid_{SSN=SSN}$ Dependents

- Most useful join in practice
Semijoin

- \( R \Join S = \Pi_{A_1, \ldots, A_n} (R \Join S) \)
- Where \( A_1, \ldots, A_n \) are the attributes in \( R \)
- Example:
  - Employee \( \Join \) Dependents
Semijoins in Distributed Databases

- Semijoins are used in distributed databases

**Employee**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Dependents**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Dname</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Employee $\times_{ssn=ssn} (\sigma_{age>71} (\text{Dependents})))$

$R = \text{Employee} \times T$

$T = \Pi_{SSN} \sigma_{age>71} (\text{Dependents})$

Answer = $R \times \text{Dependents}$
Complex RA Expressions

\[ \Pi_{\text{name}} \]

\[ \sigma_{\text{name}=\text{fred}} \]

\[ \Pi_{\text{ssn}} \]

\[ \sigma_{\text{name}=\text{gizmo}} \]

\[ \Pi_{\text{pid}} \]

\[ \Pi_{\text{buyer-ssn}=\text{ssn}} \]

\[ \Pi_{\text{seller-ssn}=\text{ssn}} \]

Person  Purchase  Person  Product
Operations on Bags

A bag = a set with repeated elements
All operations need to be defined carefully on bags
• \( \{a,b,b,c\} \cup \{a,b,b,b,e,f,f\} = \{a,a,b,b,b,b,b,c,e,f,f\} \)
• \( \{a,b,b,b,c,c\} - \{b,c,c,c,d\} = \{a,b,b,d\} \)
• \( \sigma_C(R) \): preserve the number of occurrences
• \( \Pi_A(R) \): no duplicate elimination
• Cartesian product, join: no duplicate elimination

Important ! Relational Engines work on bags, not sets !

Reading assignment: 5.3 – 5.4
Note: RA has Limitations!

- Cannot compute “transitive closure”

<table>
<thead>
<tr>
<th>Name1</th>
<th>Name2</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Mary</td>
<td>Father</td>
</tr>
<tr>
<td>Mary</td>
<td>Joe</td>
<td>Cousin</td>
</tr>
<tr>
<td>Mary</td>
<td>Bill</td>
<td>Spouse</td>
</tr>
<tr>
<td>Nancy</td>
<td>Lou</td>
<td>Sister</td>
</tr>
</tbody>
</table>

- Find all direct and indirect relatives of Fred
- Cannot express in RA !!! Need to write C program
From SQL to RA

Purchase(buyer, product, city)
Person(name, age)

\[
\text{SELECT DISTINCT } P.\text{buyer} \\
\text{FROM Purchase P, Person Q} \\
\text{WHERE } P.\text{buyer}=Q.\text{name AND} \\
P.\text{city}=\text{‘Seattle’ AND} \\
Q.\text{age} > 20
\]
Also...

Purchase(buyer, product, city)
Person(name, age)

```
SELECT DISTINCT P.buyer
FROM Purchase P, Person Q
WHERE P.buyer=Q.name AND
  P.city='Seattle' AND
  Q.age > 20
```
Non-monontone Queries (in class)

Purchase(buyer, product, city)
Person(name, age)

```sql
SELECT DISTINCT P.product
FROM Purchase P
WHERE P.city='Seattle' AND
not exists (select *
from Purchase P2, Person Q
where P2.product = P.product
and P2.buyer = Q.name
and Q.age > 20)
```
Extended Logical Algebra Operators
(operate on Bags, not Sets)

- Union, intersection, difference
- Selection $\sigma$
- Projection $\Pi$
- Join $|x|$ 
- Duplicate elimination $\delta$
- Grouping $\gamma$
- Sorting $\tau$
Logical Query Plan

SELECT city, count(*)
FROM sales
GROUP BY city
HAVING sum(price) > 100

T1, T2, T3 = temporary tables
Logical v.s. Physical Algebra

• We have seen the logical algebra so far:
  – Five basic operators, plus group-by, plus sort

• The Physical algebra refines each operator into a concrete algorithm
Physical Plan

SELECT DISTINCT P.buyer FROM Purchase P, Person Q WHERE P.buyer=Q.name AND P.city='Seattle' AND Q.age > 20

- Sequential scan
- Index-join
- Hash-based dup. elim
Physical Plans Can Be Subtle

```
SELECT * 
FROM Purchase P 
WHERE P.city='Seattle'
```

Where did the join come from?