Introduction to Database Systems
CSE 444

Lecture 20:
Query Execution: Relational Algebra

May 21, 2008

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:

  - Declarative query language
  - Algebra
  - Implementation

  - SQL, relational calculus
  - Relational algebra
  - Relational bag algebra

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
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
  - UnionizedEmployees $\cap$ RetiredEmployees

3. Selection

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

5. Cartesian Product

- Each tuple in $R_1$ with each tuple in $R_2$
- Notation: $R_1 \times R_2$
- Example:
  - $\text{Employee} \times \text{Dependents}$
- Very rare in practice; mainly used to express joins
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: $\rho$

Renaming

- Changes the schema, not the instance
- Notation: $\rho_{B_1,\ldots,B_n} (R)$
- Example:
  - $\rho_{\text{LastName, SocSocNo}} (\text{Employee})$
  - 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>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LastName</td>
<td>SocSocNo</td>
</tr>
<tr>
<td>John</td>
<td>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>

Natural Join

- Notation: $R_1 \bowtie R_2$
- Meaning: $R_1 \bowtie R_2 = \Pi_A (\sigma_C (R_1 \times R_2))$
- 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>99999999999</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>

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

Natural Join

- Given the schemas \( R(A, B, C, D) \), \( S(A, C, E) \), what is the schema of \( R \bowtie S \) ?
- Given \( R(A, B, C) \), \( S(D, E) \), what is \( R \bowtie S \) ?
- Given \( R(A, B) \), \( S(A, B) \), what is \( R \bowtie S \) ?

**Theta Join**

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

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

• Most useful join in practice

Semijoin

• $R \bowtie S = \Pi_{A_1, \ldots, A_n} (R \bowtie \leq S)$
• Where $A_1, \ldots, A_n$ are the attributes in $R$
• Example:
  – Employee $\bowtie$ Dependents

Semijoins in Distributed Databases

• Semijoins are used in distributed databases

Complex RA Expressions

$\Pi_{name}$

$\sigma_{name=fred}$

$\Pi_{ssn}$

$\sigma_{name=gizmo}$

$\Pi_{pid}$

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\} ∪ \{a,b,b,b,c,c,f,f\} = \{a,a,b,b,b,b,c,c,f,f\}
• \{a,b,b,b,c,e\} – \{b,c,c,d\} = \{a,b\}
• \(\sigma_c(R)\): 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)

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

Also...

```
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)
```

```
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 $\Join$
- Duplicate elimination $\delta$
- Grouping $\gamma$
- Sorting $\tau$

Logical Query Plan

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

```
T3(city, c)
\Pi_{city, c} \quad T2(city,p,c)
\sigma_{p > 100} \quad T1(city,p,c)
\gamma_{city, sum(price)\rightarrow p, count(*)\rightarrow c}
\quad sales(product, city, price)
```

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
```

Physical Plans Can Be Subtle

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

Where did the join come from?