Relational Algebra

Lecture 7

Outline

• Relational Algebra (Section 6.1)
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

• Five operators:
  – Union: ∪
  – Difference: -
  – Selection: s
  – Projection: P
  – Cartesian Product: ×

• Derived or auxiliary operators:
  – Intersection, complement
  – Joins (natural, equi-join, theta join, semi-join)
  – Renaming: r
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: \( s_c(R) \)
• Examples
  – \( s_{\text{Salary} > 40000} \) (Employee)
  – \( s_{\text{name} = "Smith"} \) (Employee)
• The condition \( c \) can be \( =, \leq, >, \geq, <> \)

[in SQL: SELECT * FROM Employee WHERE Salary > 40000]

Selection Example

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>DepartmentID</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999999999</td>
<td>John</td>
<td>1</td>
<td>30,000</td>
</tr>
<tr>
<td>7777777777</td>
<td>Tony</td>
<td>1</td>
<td>32,000</td>
</tr>
<tr>
<td>8888888888</td>
<td>Alice</td>
<td>2</td>
<td>45,000</td>
</tr>
</tbody>
</table>

Find all employees with salary more than $40,000.
\( s_{\text{Salary} > 40000} \) (Employee)

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>DepartmentID</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>8888888888</td>
<td>Alice</td>
<td>2</td>
<td>45,000</td>
</tr>
</tbody>
</table>
4. Projection

- Eliminates columns, then removes duplicates
- Notation: $P_{A_1,\ldots,A_n}(R)$
- Example: project to social-security number and names:
  - $P_{SSN, Name}(Employee)$
  - Output schema: Answer(SSN, Name)

[In SQL: SELECT DISTINCT SSN, Name FROM Employee]
5. Cartesian Product

- Combine each tuple in R1 with each tuple in R2
- Notation: $R_1 \times R_2$
- Example:
  - $\text{Employee} \times \text{Dependents}$
- Very rare in practice; mainly used to express joins

[In SQL: SELECT * FROM R1, R2]
Relational Algebra

• Five operators:
  – Union: ∪
  – Difference: -
  – Selection: s
  – Projection: P
  – Cartesian Product: ×

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

Renaming

• Changes the schema, not the instance
• Schema: R(A₁, …, Aₙ)
• Notation: r(B₁,…,Bₙ)(R)
• Example:
  – r_LastName, SocSocNo(Employee)
  – Output schema: Answer(LastName, SocSocNo)

[In SQL: SELECT Name AS LastName, SSN AS SocSocNo FROM Employee]
Renaming Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>

\[ r_{\text{LastName, SocSocNo}}(\text{Employee}) \]

<table>
<thead>
<tr>
<th>LastName</th>
<th>SocSocNo</th>
</tr>
</thead>
<tbody>
<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: \( R_1 \bowtie R_2 = P_A(s_C(R_1 \times R_2)) \)

- Where:
  - The selection \( s_C \) checks equality of all common attributes
  - The projection eliminates the duplicate common attributes

[In SQL:
SELECT DISTINCT R1.A, R1.B, R2.C FROM R1, R2
WHERE R1.B = R2.B
Schema: R1(A,B), R2(B,C)]
Natural Join Example

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

\[ \text{Employee} \bowtie \text{Dependents} = \]

\[ \text{P}_{\text{Name, SSN, Dname}}(s \text{SSN} = \text{SSN}_2(\text{Employee} \bowtimes \text{Dependents})) \]

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

**Natural Join**

- **R**
  \[
  \begin{array}{ccc}
  A & B \\
  X & Y \\
  X & Z \\
  Y & Z \\
  Z & V \\
  \end{array}
  \]

- **S**
  \[
  \begin{array}{ccc}
  B & C \\
  Z & U \\
  V & W \\
  Z & V \\
  \end{array}
  \]

- **R \bowtie S**
  \[
  \begin{array}{ccc}
  A & B & C \\
  X & Z & U \\
  X & Z & V \\
  Y & Z & U \\
  Y & Z & V \\
  Z & V & W \\
  \end{array}
  \]
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
• $R1 \bowtie_{q} R2 = s_{q} (R1 \times R2)$
• Here $q$ can be any condition
Eq-join

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

• Most useful join in practice
  (difference to natural join?)

Semijoin

• \( R \bowtie S = P_{A_1, \ldots, A_n} (R \bowtie S) \)
• 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

![Diagram of Employee and Dependents networks]

\[
\text{Employee} \Join_{\text{ssn}=\text{ssn}} (\text{Dependents})
\]

\[
R = \text{Employee} \times T
\]

\[
T = \text{P}_{\text{SSN}} \text{S}_{\text{age}>71} (\text{Dependents})
\]

\[
\text{Answer} = R \bowtie \text{Dependents}
\]

---

Complex RA Expressions

![Diagram of Person, Purchase, and Product relationships]
Application:
Query Rewriting for Optimization

The earlier we process selections, less tuples we need to manipulate higher up in the tree (predicate pushdown)

Disadvantages?

Algebraic Laws (Examples)

• Commutative and Associative Laws
  – $R \cap S = S \cap R$, $R \cap (S \cap T) = (R \cap S) \cap T$
  – $R \cup S = S \cup R$, $R \cup (S \cup T) = (R \cup S) \cup T$

• Laws involving selection
  – $s_{\mathit{C \land C'}}(R) = s_{\mathit{C}}(s_{\mathit{C'}}(R)) = s_{\mathit{C}}(R) \cap s_{\mathit{C'}}(R)$
  – $s_{\mathit{C}}(R \cup S) = s_{\mathit{C}}(R) \cup S$
    • When $C$ involves only attributes of $R$

• Laws involving projections
  – $P_{M}(P_{N}(R)) = P_{M,N}(R)$
Operations on Bags

A **bag** = a set with repeated elements

All operations need to be defined carefully on bags

- \(\{a,b,b,c\} \cup \{a,b,b,e,f,f\} = \{a,a,b,b,b,c,e,f,f\}\)
- \(\{a,b,b,b,c,c\} - \{b,c,c,c,d\} = \{a,b,b\}\)
- \(s_C(R)\): preserve the number of occurrences
- \(P_A(R)\): no duplicate elimination
- Cartesian product, join: no duplicate elimination

Important ! Relational Engines work on bags, not sets !

---

Finally: 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