Lecture 16: Relational Algebra

Monday, May 10, 2010
Outline

Relational Algebra:
• Chapters 5.1 and 5.2
The WHAT and the HOW

• In SQL we write WHAT we want to get form the data

• The database system needs to figure out HOW to get the data we want

• The passage from WHAT to HOW goes through the Relational Algebra
SQL = WHAT

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
  x.price > 100 and z.city = ‘Seattle’

It’s clear WHAT we want, unclear HOW to get it
Relational Algebra = HOW

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Temporary tables
T1, T2, ...

Final answer

δ

Π

σ

price>100 and city='Seattle'

cid=cid

T2(. . . )

T1(pid,name,price,pid,cid,store)

T4(name,name)

T3(. . . )
Relational Algebra = HOW

The order is now clearly specified:

Iterate over PRODUCT…
…join with PURCHASE…
…join with CUSTOMER…
…select tuples with Price>100 and City=‘Seattle’…
…eliminate duplicates…
…and that’s the final answer!
Sets v.s. Bags

- Sets: \{a,b,c\}, \{a,d,e,f\}, \{\}\, . . .
- Bags: \{a, a, b, c\}, \{b, b, b, b, b\}, . . .

Relational Algebra has two semantics:
- Set semantics
- Bag semantics
Extended Algebra Operators

- Union $\cup$, intersection $\cap$, difference $-$
- Selection $\sigma$
- Projection $\Pi$
- Join $\bowtie$
- Rename $\rho$
- Duplicate elimination $\delta$
- Grouping and aggregation $\gamma$
- Sorting $\tau$
Relational Algebra (1/3)

The Basic Five operators:
• Union: $\cup$
• Difference: $-$
• Selection: $\sigma$
• Projection: $\Pi$
• Join: $\bowtie$
Relational Algebra (2/3)

Derived or auxiliary operators:

- Renaming: $\rho$
- Intersection, complement
- Variations of joins
  - natural, equi-join, theta join, semi-join, cartesian product
Relational Algebra (3/3)

Extensions for bags:
• Duplicate elimination: $\delta$
• Group by: $\gamma$
• Sorting: $\tau$
Union and Difference

\[ R_1 \cup R_2 \]
\[ R_1 - R_2 \]

What do they mean over bags?
What about Intersection?

- Derived operator using minus
  \[ R_1 \cap R_2 = R_1 - (R_1 - R_2) \]
- Derived using join (will explain later)
  \[ R_1 \cap R_2 = R_1 \bowtie R_2 \]
Selection

• Returns all tuples which satisfy a condition

\[ \sigma_c(R) \]

• Examples
  – \( \sigma_{\text{Salary} > 40000} \) (Employee)
  – \( \sigma_{\text{name} = \text{“Smith”}} \) (Employee)

• The condition \( c \) can be \( =, <, \le, >, \ge, <> \)
### 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>Smith</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>500000</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{Salary > 40000}} (\text{Employee}) \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>600000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>500000</td>
</tr>
</tbody>
</table>
Projection

• Eliminates columns

\[ \Pi_{A_1, \ldots, A_n}(R) \]

• Example: project social-security number and names:
  - \( \Pi_{\text{SSN, Name}}(\text{Employee}) \)
  - Answer(\( \text{SSN, Name} \))

Semantics differs over set or over bags
### 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>

\[ \Pi_{\text{Name}, \text{Salary}} (\text{Employee}) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>John</td>
<td>60000</td>
</tr>
<tr>
<td>John</td>
<td>20000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>John</td>
<td>60000</td>
</tr>
</tbody>
</table>

**Bag semantics**

**Set semantics**

Which is more efficient to implement?
Cartesian Product

• Each tuple in R1 with each tuple in R2

\[ R_1 \times R_2 \]

• Very rare in practice; mainly used to express joins
## Employee

<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>

## Dependent

<table>
<thead>
<tr>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

## Employee $\times$ Dependent

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>999999999</td>
<td>777777777</td>
<td>Joe</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
<td>777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>
Renaming

• Changes the schema, not the instance

\[ \rho_{B_1, \ldots, B_n} (R) \]

• Example:
  – \( \rho_{N, S}(\text{Employee}) \rightarrow \text{Answer}(N, S) \)
Natural Join

\[ R_1 \bowtie R_2 \]

• Meaning: \( R_1 \bowtie R_2 = \Pi_A(\sigma(R_1 \times R_2)) \)

• Where:
  – The selection \( \sigma \) checks equality of all common attributes
  – The projection eliminates the duplicate common attributes
Natural Join

\[ R \Join S = \Pi_{ABC}(\sigma_{R.B=S.B}(R \times S)) \]

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

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td></td>
<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 $R \bowtie S$ ?

• 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

\[ R_1 \Join_\theta R_2 = \sigma_\theta (R_1 \times R_2) \]

• Here \( \theta \) can be any condition
Eq-join

- A theta join where $\theta$ is an equality

$$R_1 \bowtie_{A=B} R_2 = \sigma_{A=B} (R_1 \times R_2)$$

- This is by far the most used variant of join in practice
So Which Join Is It?

• When we write \( R \bowtie S \) we usually mean an eq-join, but we often omit the equality predicate when it is clear from the context.
Semijoin

\[
R \bowtie_C S = \Pi_{A_1, \ldots, A_n} (R \bowtie_C S)
\]

- Where \(A_1, \ldots, A_n\) are the attributes in \(R\)
Semijoins in Distributed Databases

Employee \bowtie_{\text{SSN} = \text{EmpSSN}} (\sigma_{\text{age} > 71} (\text{Dependent}))

Task: compute the query with minimum amount of data transfer
Semijoins in Distributed Databases

$\text{Employee} \bowtie_{\text{SSN}=\text{EmpSSN}} (\sigma_{\text{age}>71} (\text{Dependent}))$

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

Employee

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

Dependent

<table>
<thead>
<tr>
<th>EmpSSN</th>
<th>DepName</th>
<th>Age</th>
<th>Stuff</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Employee \Join_{SSN=EmpSSN} (σ \text{age}>71 (Dependent))

T(SSN) = \Pi_{SSN} \sigma \text{age}>71 (Dependants)

R = Employee \Join_{SSN=EmpSSN} T

= Employee \Join_{SSN=EmpSSN} (σ \text{age}>71 (Dependants))
Semijoins in Distributed Databases

Employee \bowtie_{SSN=EmpSSN} (\sigma_{age>71} (Dependent))

T(SSN) = \Pi_{SSN} \sigma_{age>71} (Dependants)

R = Employee \bowtie_{SSN=EmpSSN} T

Answer = R \bowtie_{SSN=EmpSSN} Dependents
Joins R US

• The join operation in all its variants (eq-join, natural join, semi-join, outer-join) is at the heart of relational database systems

• WHY ?
Operators on Bags

• Duplicate elimination $\delta$
  \[ \delta(R) = \text{select distinct * from } R \]

• Grouping $\gamma$
  \[ \gamma_{A,\text{sum}(B)} = \text{select } A,\text{sum}(B) \text{ from } R \text{ group by } A \]

• Sorting $\tau$
Complex RA Expressions

\[ \gamma \text{ u.name, count(*)} \]

\[ \Pi \text{ ssn} \quad \Pi \text{ pid} \]

\[ \sigma \text{ name=fred} \quad \sigma \text{ name=gizmo} \]

\[ \sigma \text{ name=fred} \quad \sigma \text{ name=gizmo} \]

\[ x.ssn=y.buyer-ssn \quad y.pid=u.pid \quad y.seller-ssn=z.ssn \]

Person x  Purchase y  Person z  Product u
RA = Dataflow Program

- Several operations, plus strictly specified order

- In RDBMS the dataflow graph is always a tree

- Novel applications (s.a. PIG), dataflow graph may be a DAG
Limitations of RA

- Cannot compute “transitive closure”
- Find all direct and indirect relatives of Fred
- Cannot express in RA !!! Need to write Java program
- Remember *the Bacon number* ? Needs TC too !

<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>