Introduction to Database Systems
CSE 444

Lecture 18: Relational Algebra

Outline

• Motivation and sets v.s. bags
• Relational Algebra
• Translation from SQL to the Relational Algebra

• Read Sections 2.4, 5.1, and 5.2
  – [Old edition: 5.1 through 5.4]
  – These book sections go over relational operators

The WHAT and the HOW

• In SQL, we write WHAT we want to get from 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 = z.cid and
x.price > 100 and z.city = 'Seattle'

It's clear WHAT we want, unclear HOW to get it

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!
Relations

• A relation is a set of tuples
  – Sets: \{a, b, c\}, \{a, d, e, f\}, \{\}\ldots
• But, commercial DBMS’s implement relations that are bags rather than sets
  – Bags: \{a, a, b, c\}, \{b, b, b, b, b\}, \ldots

Sets v.s. Bags

Relational Algebra has two flavors:
• Over sets: theoretically elegant but limited
• Over bags: needed for SQL queries + more efficient
  – Example: Compute average price of all products

We discuss set semantics
• We mention bag semantics only where needed

Outline

• Motivation and sets v.s. bags
• Relational Algebra
• Translation from SQL to the Relational Algebra
• Read Sections 2.4, 5.1, and 5.2
  – [Old edition: 5.1 through 5.4]
  – These book sections go over relational operators

Relational Algebra

• Query language associated with relational model
• Queries specified in an operational manner
  – A query gives a step-by-step procedure
• Relational operators
  – Take one or two relation instances as argument
  – Return one relation instance as result
  – Easy to compose into relational algebra expressions

Relational Algebra (1/3)

Five basic operators:
• Union (\cup) and Set difference (\neg)
• Selection: \sigma_{\text{condition}}(S)
  – Condition is Boolean combination (\land, \lor) of terms
  – Term is: attribute op constant, attr. op attr.
  – Op is: <, <=, =, \neq, >=, or >
• Projection: \pi_{\text{list-of-attributes}}(S)
• Cross-product or cartesian product (\times)

Relational Algebra (2/3)

Derived or auxiliary operators:
• Intersection (\cap), Division (R/S)
• Join: R \bowtie S = \sigma_{\theta}(R \times S)
• Variations of joins
  – Natural, equijoin, theta-join
  – Outer join and semi-join
• Rename \rho_{B1,...,Bn}(S)
Relational Algebra (3/3)

Extensions for bags
- Duplicate elimination: δ
- Group by: γ [Same symbol as aggregation]
  - Partitions tuples of a relation into "groups"
- Sorting: τ

Other extensions
- Aggregation: γ (min, max, sum, average, count)

Union and Difference
- R1 ∪ R2
- Example:
  - ActiveEmployees ∪ RetiredEmployees
- R1 – R2
- Example:
  - AllEmployees – RetiredEmployees

Be careful when applying to bags!

What about Intersection ?
- It is a derived operator
- R1 ∩ R2 = R1 – (R1 – R2)
- Also expressed as a join (will see later)
- Example
  - UnionizedEmployees ∩ RetiredEmployees

Relational Algebra (1/3)

Five basic operators:
- Union (∪) and Set difference (–)
- Selection: σcondition(R)
  - Condition is Boolean combination (∧, ∨) of terms
  - Term is: attribute op constant, attr. op attr.
  - Op is: <, <=, =, ≠, >=, or >
- Projection: πlist-of-attributes(R)
- Cross-product or cartesian product (×)

Selection
- Returns all tuples that satisfy a condition
- Notation: σc(R)
- Examples
  - σSalary > 40000(Employee)
  - σName = "Smith"(Employee)
- The condition c can be
  - Boolean combination (∧, ∨) of terms
  - Term is: attribute op constant, attr. op attr.
  - Op is: <, <=, =, ≠, >=, or >

<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>

σSalary > 40000(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
- Notation: \( \Pi A_1, \ldots, A_n (R) \)
- Example: project social-security number and names:
  - \( \Pi \text{SSN}, \text{Name} (\text{Employee}) \)
  - Output schema: Answer(\text{SSN}, \text{Name})

**Semantics differs over set or over bags**

Selection & Projection Examples

<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>

\( \Pi \text{Name, Salary} (\text{Employee}) \)

**Set semantics: duplicate elimination automatic**

<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: no duplicate elimination; need explicit \( \delta \)**

Relational Algebra (1/3)

- Each tuple in R1 with each tuple in R2
- Notation: \( R_1 \times R_2 \)
- Example:
  - Employee \( \times \) Dependents
- Rare in practice; mainly used to express joins

**Cartesian Product**

Five basic operators:
- Union (\( \cup \)) and Set difference (\( \cdot \))
- Selection: \( \sigma_{\text{condition}}(S) \)
  - Condition is Boolean combination (\( \land, \lor \)) of terms
  - Term is: attribute op constant, attr. op attr.
  - Op is: <, \( \leq \), =, \( \neq \), \( \geq \), or >
- Projection: \( \pi_{\text{list-of-attributes}}(S) \)
- Cross-product or cartesian product (\( \times \))
Relational Algebra (2/3)

Derived or auxiliary operators:
- Intersection \((\cap)\), Division \((R/S)\)
- Join: \(R \ast_{\theta} S = \sigma_{\theta}(R \times S)\)
- Variations of joins
  - Natural, equijoin, theta-join
  - Outer join and semi-join
- Rename \(p_{B_1, \ldots, B_n}(S)\)

**Cartesian Product 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>??????????</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>SSN</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>??????????</td>
<td>Joe</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>employee</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>??????????</td>
<td>999999999</td>
<td>Joe</td>
</tr>
</tbody>
</table>

**Renaming**

- Changes the schema, not the instance
- Notation: \(p_{B_1, \ldots, B_n}(R)\)
- Example:
  - \(p_{\text{LastName}, \text{SocSocNo}}(\text{Employee})\)
  - Output schema:
    - Answer(\text{LastName}, \text{SocSocNo})

**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>??????????</td>
</tr>
</tbody>
</table>

\(p_{\text{LastName}, \text{SocSocNo}}(\text{Employee})\)

<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>??????????</td>
</tr>
</tbody>
</table>

**Different Types of Join**

- Theta-join: \(R \ast_{\theta_i} S = \sigma_{\theta_i}(R \times S)\)
  - Join of R and S with a join condition \(\theta_i\)
  - Cross-product followed by selection \(\sigma_{\theta_i}\)
- Equijoin: \(R \ast_{\theta_i} S = \pi_{\theta_i}(R \times S)\)
  - Join condition \(\theta_i\) consists only of equalities
  - Projection \(\pi_{\theta_i}\) drops all redundant attributes
- Natural join: \(R \ast S = \pi_{\theta_i}(R \times S)\)
  - Equijoin
  - Equality on all fields with same name in R and in S
Theta-Join Example

\[ P \Join_{\text{age} = J\text{.age} \land \text{zip} = J\text{.zip} \land \text{P.age} < 50} J \]

<table>
<thead>
<tr>
<th>P.age</th>
<th>P.zip</th>
<th>P.disease</th>
<th>P.job</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>lawyer</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
</tr>
</tbody>
</table>

Equijoin Example

\[ P \Join_{\text{age} = J\text{.age}} J \]

<table>
<thead>
<tr>
<th>P.age</th>
<th>P.zip</th>
<th>P.disease</th>
<th>P.job</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>lawyer</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
</tr>
</tbody>
</table>

Natural Join Example

\[ P \bowtie J \]

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
<th>job</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>lawyer</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
</tr>
</tbody>
</table>

So Which Join Is It?

- When we write \( R \Join S \) we usually mean an equijoin, but we often omit the equality predicate when it is clear from the context.

Outer Join Example

\[ P \bowtie \pi V \]

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
<th>job</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>lawyer</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
<td>null</td>
</tr>
</tbody>
</table>

More Joins

- **Outer join**
  - Include tuples with no matches in the output
  - Use NULL values for missing attributes

- **Variants**
  - Left outer join
  - Right outer join
  - Full outer join
Semijoin

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

Example of Algebra Queries

Q1: Jobs of patients who have heart disease

$\pi_{\text{job}} (\text{AnnonJob} \sigma_{\text{disease} = \text{'heart'}} (\text{AnonPatient}))$

Q2: Name of supplier of parts with size greater than 10

$\pi_{\text{sname}} (\text{Supplier} \bowtie_{\text{psize}} \text{Supply} \sigma_{\text{psize} > 10} (\text{Part}))$

Q3: Name of supplier of red parts or parts with size greater than 10

$\pi_{\text{sname}} (\text{Supplier} \bowtie_{\text{pcolor}} \text{Supply} \sigma_{\text{pcolor} = \text{'red'}} (\text{Part}) \cup \sigma_{\text{psize} > 10} (\text{Part}))$

RA Expressions v.s. Programs

- An Algebra Expression is like a program
  - Several operations
  - Strictly specified order
- But Algebra expressions have limitations
RA and Transitive Closure

• 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 Java program

Outline

• Motivation and sets v.s. bags
• Relational Algebra
• Translation from SQL to the Relational Algebra

• Read Sections 2.4, 5.1, and 5.2
  – [Old edition: 5.1 through 5.4]
  – These book sections go over relational operators

From SQL to RA

\[
\begin{align*}
\text{SELECT} & \text{DISTINCT x.name, z.name} \\
\text{FROM} & \text{Product x, Purchase y, Customer z} \\
\text{WHERE} & x.pid = y.pid \text{ and y.cid = z.cid and} \\
& x.price > 100 \text{ and z.city = 'Seattle'}
\end{align*}
\]

Operators on Bags

• Duplicate elimination δ
• Grouping γ
• Sorting τ
Logical Query Plan

\[
\begin{align*}
\text{SELECT} & \text{ city, count(*) } \\
\text{FROM} & \text{ sales } \\
\text{GROUP BY} & \text{ city } \\
\text{HAVING} & \text{ sum(price) > 100 } \\
\end{align*}
\]

\[
\begin{align*}
T3(\text{city, c}) & \rightarrow \Pi_{\text{city}, \text{c}} \\
T2(\text{city, p, c}) & \rightarrow \sigma_{\text{p} > 100} \\
T1(\text{city, p, c}) & \rightarrow \pi_{\text{city, c}} \\
\end{align*}
\]

T1, T2, T3 = temporary tables

Non-monontone Queries
(at home !)

\[
\begin{align*}
\text{SELECT DISTINCT} & \text{ z.store } \\
\text{FROM} & \text{ Customer z } \\
\text{WHERE} & \text{ z.city = 'Seattle' AND } \\
& \text{ not exists (select * } \\
& \text{from Product x, Purchase y } \\
& \text{where x.pid = y.pid } \\
& \text{and y.cid = z.cid } \\
& \text{and x.price < 100) } \\
\end{align*}
\]