Introduction to Database Systems
CSE 444

Lecture 17: Relational Algebra
Outline

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

• Read Sections 2.4, 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 = z.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, . . .

Customer

Product

Purchase

T1(pid, name, price, pid, cid, store)
T2( . . . )
T3( . . . )
T4(name, name)

σ price>100 and city='Seattle'

Π x.name, z.name

δ

Final answer

σ cid = cid

pid = pid
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 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
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: $<$, $\leq$, $=$, $\neq$, $\geq$, or $>$
- **Projection**: $\pi_{\text{list-of-attributes}}(S)$
- **Cross-product or cartesian product** ($\times$)
Derived or auxiliary operators:

- **Intersection** (\( \cap \)), **Division** (\( R/S \))

- **Join**: \( R \bowtie_{\theta} S = \sigma_{\theta}(R \times S) \)

- **Variations of joins**
  - Natural, equijoin, theta-join
  - Outer join and semi-join

- **Rename** \( \rho_{B_1, \ldots, B_n}(S) \)
Relational Algebra (3/3)

Extensions for bags

• Duplicate elimination: $\delta$
• Group by: $\gamma$ [Same symbol as aggregation]
  – Partitions tuples of a relation into “groups”
• Sorting: $\tau$

Other extensions

• Aggregation: $\gamma$ (min, max, sum, average, count)
Union and Difference

- $R_1 \cup R_2$
- Example:
  - $\text{ActiveEmployees} \cup \text{RetiredEmployees}$

- $R_1 - R_2$
- Example:
  - $\text{AllEmployees} - \text{RetiredEmployees}$

Be careful when applying to bags!
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
Selection

• Returns all tuples that satisfy a condition
• Notation: $\sigma_c(R)$
• Examples
  – $\sigma_{\text{Salary} > 40000}$ (Employee)
  – $\sigma_{\text{name} = "\text{Smith}"}$ (Employee)
• The condition $c$ can be
  – Boolean combination $(\land, \lor)$ of terms
  – Term is: attribute op constant, attr. op attr.
  – Op is: <, <=, =, \neq, >=, 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>

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

<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
• 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})

Semantics differs over set or over bags
\( \Pi_{\text{Name}, \text{Salary}} (\text{Employee}) \)

<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>

Set semantics: duplicate elimination automatic
\[ \Pi_{\text{Name,Salary}} \text{(Employee)} \]

<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>
<tr>
<td>John</td>
<td>200000</td>
</tr>
</tbody>
</table>

Bag semantics: no duplicate elimination; need explicit \( \delta \)
Cartesian Product

- Each tuple in R1 with each tuple in R2
- Notation: $R1 \times R2$
- 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>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>

### Dependents

<table>
<thead>
<tr>
<th>EmployeeSSN</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>777777777</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>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
• Notation: $\rho_{B_1,\ldots,B_n}(R)$
• Example:
  - $\rho_{\text{LastName, SocSocNo}}(\text{Employee})$
  - Output schema: $\text{Answer(LastName, SocSocNo)}$
Different Types of Join

- **Theta-join**: \( R \bowtie_\theta S = \sigma_\theta(R \times S) \)
  - Join of \( R \) and \( S \) with a join condition \( \theta \)
  - Cross-product followed by selection \( \theta \)

- **Equijoin**: \( R \bowtie_\theta S = \pi_A(\sigma_\theta(R \times S)) \)
  - Join condition \( \theta \) consists only of equalities
  - Projection \( \pi_A \) drops all redundant attributes
  - By far most used join in practice

- **Natural join**: \( R \bowtie S = \pi_A(\sigma_\theta(R \times S)) \)
  - Equijoin
  - Equality on all common attributes (names) in \( R \) and in \( S \)
  - Projection drops duplicate common attributes
**Theta-Join Example**

### AnonPatient P

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

### AnonJob J

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

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

<table>
<thead>
<tr>
<th>P\text{.age}</th>
<th>P\text{.zip}</th>
<th>disease</th>
<th>J\text{.job}</th>
<th>J\text{.age}</th>
<th>J\text{.zip}</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>
Equijoin Example

<table>
<thead>
<tr>
<th>AnnonPatient P</th>
<th>AnnonJob J</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>zip</td>
</tr>
<tr>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

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

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

AnonPatient \( P \)

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

AnnonJob \( J \)

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

\( P \bowtie J \)

<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 $R \bowtie S$ we usually mean an equijoin, but we often omit the equality predicate when it is clear from the context.
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
## Outer Join Example

### AnonPatient P

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

### AnnonJob J

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

### Outer Join

<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>

CSE 444 - Summer 2010
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:
  - Employee $\bowtie$ Dependents

- Particularly useful in distributed databases
  - Compute the query with minimum amount of data transfer
Semijoins in Distributed Databases

- Semijoins are used in distributed databases

\[
\text{Employee} \bowtie_{\text{ssn}=\text{ssn}} (\sigma_{\text{age}>71} (\text{Dependents}))
\]

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

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

Answer = R \bowtie \text{Dependents}
Operators on Bags

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

- 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

\[ \Pi_{\text{name}} \]
\[ \text{buyer-ssn}=\text{ssn} \]
\[ \Pi_{\text{ssn}} \]
\[ \sigma_{\text{name}}=\text{fred} \]
\[ \Pi_{\text{pid}} \]
\[ \sigma_{\text{name}}=\text{gizmo} \]
RA = Dataflow Program

• 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