CSE 344

APRIL 6TH – RELATIONAL ALGEBRA
ASSORTED MINUTIAE

- HW2 out (Due Wednesday)
  - git pull upstream master
- OQ1 due tonight
- OQ2/3 Out
  - Both due next Friday
- Azure accounts will be created over the weekend
  - Needed for HW3
• Remember from last week
  • SQL queries are combinations of functions on tables
  • Each one receives tables as input and has a table as an output
RELATIONAL ALGEBRA

Set-at-a-time algebra, which manipulates relations

In SQL we say *what* we want

In RA we can express *how* to get it

Every DBMS implementations converts a SQL query to RA in order to execute it

An RA expression is called a *query plan*
BASICS

• Relations and attributes

• Functions that are applied to relations
  – Return relations
  – Can be composed together
  – Often displayed using a tree rather than linearly
  – Use Greek symbols: $\sigma$, $\pi$, $\delta$, etc
SETS V.S. BAGS

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

Relational Algebra has two flavors:
Set semantics = standard Relational Algebra
Bag semantics = extended Relational Algebra

DB systems implement bag semantics (Why?)
RELATIONAL ALGEBRA OPERATORS

Union $\cup$, intersection $\cap$, difference $-$
Selection $\sigma$
Projection $\pi$
Cartesian product $\times$, join $\Join$
(Rename $\rho$)
Duplicate elimination $\delta$
Grouping and aggregation $\gamma$
Sorting $\tau$

All operators take in 1 or more relations as inputs and return another relation
UNION AND DIFFERENCE

R1 \( \cup \) R2
R1 – R2

Only make sense if R1, R2 have the same schema

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

\[ R_1 \cap R_2 = R_1 \Join R_2 \]
SELECTION

Returns all tuples which satisfy a condition

Examples

- $\sigma_{\text{Salary} > 40000}$ (Employee)
- $\sigma_{\text{name} = \text{"Smith"}}$ (Employee)

The condition $c$ can be $=, <, <=, >, >=, <$ combined with AND, OR, NOT
### Employee

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>50000</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>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>50000</td>
</tr>
</tbody>
</table>
PROJECTION

Eliminates columns

$\pi_{A_1,\ldots,A_n}(R)$

Example: project social-security number and names:

- $\pi_{\text{SSN, Name}}(\text{Employee}) \rightarrow \text{Answer(SSN, Name)}$

Different semantics over sets or bags! Why?
### Employee

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>5423341</td>
<td>John</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>John</td>
<td>20000</td>
</tr>
</tbody>
</table>

The set of names with their salaries can be represented as:

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

**Bag semantics**

<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>

**Set semantics**

<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>

**Which is more efficient?**
COMPOSING RA OPERATORS

Patient

<table>
<thead>
<tr>
<th>no</th>
<th>name</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p1</td>
<td>98125</td>
<td>flu</td>
</tr>
<tr>
<td>2</td>
<td>p2</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>3</td>
<td>p3</td>
<td>98120</td>
<td>lung</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>

$\sigma_{\text{disease}=\text{\textquoteleft heart\textquoteright}}(\text{Patient})$

<table>
<thead>
<tr>
<th>no</th>
<th>name</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>p2</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>4</td>
<td>p4</td>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>

$\pi_{\text{zip, disease}}(\text{Patient})$

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

$\pi_{\text{zip, disease}}(\sigma_{\text{disease}=\text{\textquoteleft heart\textquoteright}}(\text{Patient}))$

<table>
<thead>
<tr>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>98120</td>
<td>heart</td>
</tr>
</tbody>
</table>
CARTESIAN PRODUCT

Each tuple in R1 with each tuple in R2

R1 \times R2

Rare in practice; mainly used to express joins
## CROSS-PRODUCT 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>

### Dependent

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

### Employee X Dependent

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>99999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>99999999999</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>
NATURAL JOIN

\[ R1 \Join R2 \]

Meaning: \( R1 \Join R2 = \Pi_A(\sigma_\theta (R1 \times R2)) \)

Where:

- Selection \( \sigma_\theta \) checks equality of all common attributes (i.e., attributes with same names)
- Projection \( \Pi_A \) eliminates duplicate common attributes
NATURAL JOIN EXAMPLE

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></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></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>

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

<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

### EXAMPLE 2

<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>

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>Bob</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

\[ P \bowtie V \]

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>Alice</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>Bob</td>
</tr>
</tbody>
</table>
THETA JOIN

A join that involves a predicate

\[
R1 \bowtie_\theta R2 = \sigma_\theta (R1 \times R2)
\]

Here \( \theta \) can be any condition
No projection in this case!

For our voters/patients example:

\[
P \bowtie \quad \text{P.zip} = \text{V.zip} \text{ and } \text{P.age} \geq \text{V.age} - 1 \text{ and } \text{P.age} \leq \text{V.age} + 1
\]
EQUIJOIN

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

$$R1 \bowtie_\theta R2 = \sigma_\theta (R1 \times R2)$$

By far the most used variant of join in practice

What is the relationship with natural join?
## EQUIJOIN 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>

### Voters V

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>P.age</th>
<th>P.zip</th>
<th>P.disease</th>
<th>V.name</th>
<th>V.age</th>
<th>V.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>
JOIN SUMMARY

**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 \( \theta \)
- No projection

**Equijoin:** \( R \bowtie_{\theta} S = \sigma_{\theta} (R \times S) \)
- Join condition \( \theta \) consists only of equalities
- No projection

**Natural join:** \( R \bowtie S = \pi_A (\sigma_{\theta} (R \times S)) \)
- Equality on all fields with same name in \( R \) and in \( S \)
- Projection \( \pi_A \) drops all redundant attributes
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
- Does not eliminate duplicate columns

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>

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>

The outer join operation is illustrated by the following: P ⋈ J

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

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)

Name of supplier of parts with size greater than 10
\[ \pi_{sname}(Supplier \Join Supply \Join (\sigma_{psize>10}(Part))) \]

Name of supplier of red parts or parts with size greater than 10
\[ \pi_{sname}(Supplier \Join Supply \Join (\sigma_{psize>10}(Part) \cup \sigma_{pcolor='red'}(Part))) \]
\[ \pi_{sname}(Supplier \Join Supply \Join (\sigma_{psize>10 \vee pcolor='red'}(Part))) \]

Can be represented as trees as well
REPRESENTING RA QUERIES AS TREES

\[
\pi_{sname} \left( \text{Supplier} \bowtie \text{Supply} \bowtie \left( \sigma_{\text{psize}>10} \left( \text{Part} \right) \right) \right)
\]

Answer

\[
\pi_{sname}
\]

\[
\text{Supplier}
\]

\[
\text{Supply}
\]

\[
\text{Part}
\]
RELATIONAL ALGEBRA OPERATORS

Union $\bigcup$, intersection $\bigcap$, difference $-$
Selection $\sigma$
Projection $\pi$
Cartesian product $\times$, join $\Join$
(Rename $\rho$)
Duplicate elimination $\delta$
Grouping and aggregation $\gamma$
Sorting $\tau$

All operators take in 1 or more relations as inputs and return another relation
EXTENDED RA: OPERATORS ON BAGS

Duplicate elimination $\delta$
  - Turns bags into sets (no other arguments)

Grouping $\gamma$
  - Takes in relation and a list of grouping operations (e.g., aggregates). Returns a new relation.
  - Can also perform renames at the same time

Sorting $\tau$
  - Takes in a relation, a list of attributes to sort on, and an order. Returns a new relation.
**USING EXTENDED RA OPERATORS**

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

\[ T1, T2 \text{ = temporary tables} \]
TYPICAL PLAN FOR A QUERY (1/2)

Answer

\[ \pi_{\text{fields}} \]

\[ \sigma_{\text{selection condition}} \]

\[ \text{JOIN condition} \]

\[ \text{JOIN condition} \]

R

S

SELECT fields
FROM R, S, ...
WHERE condition

SELECT-PROJECT-JOIN
Query
TYPICAL PLAN FOR A QUERY (1/2)

\[ \sigma_{\text{having condition}} \]
\[ \gamma_{\text{fields, sum/count/min/max(fields)}} \]
\[ \pi_{\text{fields}} \]
\[ \sigma_{\text{where condition}} \]
\[ \text{join condition} \]

\[
\text{SELECT fields}
\]
\[
\text{FROM R, S, ...}
\]
\[
\text{WHERE condition}
\]
\[
\text{GROUP BY fields}
\]
\[
\text{HAVING condition}
\]
HOW ABOUT SUBQUERIES?

```sql
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
  and not exists
    (SELECT *
     FROM Supply P
     WHERE P.sno = Q.sno
       and P.price > 100)
```
 HOW ABOUT SUBQUERIES? 

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
    and not exists
    (SELECT *
     FROM Supply P
     WHERE P.sno = Q.sno
          and P.price > 100)
```
HOW ABOUT SUBQUERIES?

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
  and not exists
  (SELECT *
   FROM Supply P
   WHERE P.sno = Q.sno
   and P.price > 100)
```

De-Correlation

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
  and Q.sno not in
  (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```
HOW ABOUT SUBQUERIES?

```
(SELECT Q.sno
 FROM Supplier Q
 WHERE Q.sstate = 'WA')
 EXCEPT
(SELECT P.sno
 FROM Supply P
 WHERE P.price > 100)
```
HOW ABOUT SUBQUERIES?

\[
\begin{align*}
\text{(SELECT } & \text{ Q.sno} \\
\text{FROM Supplier } & \text{ Q} \\
\text{WHERE } & \text{ Q.sstate} = \text{ 'WA'} \\
\text{EXCEPT} & \end{align*}
\]

\[
\begin{align*}
\text{(SELECT } & \text{ P.sno} \\
\text{FROM Supply } & \text{ P} \\
\text{WHERE } & \text{ P.price > 100)}
\end{align*}
\]
SUMMARY OF RA AND SQL

SQL = a declarative language where we say *what* data we want to retrieve

RA = an algebra where we say *how* we want to retrieve the data

Theorem: SQL and RA can express exactly the same class of queries

RDBMS translate SQL \(\rightarrow\) RA, then optimize RA
SUMMARY OF RA AND SQL

SQL (and RA) cannot express ALL queries that we could write in, say, Java

Example:
- Parent(p,c): find all descendants of ‘Alice’
- No RA query can compute this!
- This is called a recursive query

Next lecture: Datalog is an extension that can compute recursive queries