CSE 344

JANUARY 24TH – RELATIONAL ALGEBRA
ADMINISTRATIVE MINUTIAE

• HW1 grades out today
• HW2 grades out soon
• HW3 and OQ3 out after class
• Azure setup
REAL LIFE BREAK

• Child welfare
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
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 $=$, $<$, $\leq$, $>$, $\geq$, $\neq$ combined with AND, OR, NOT
\[
\sigma_{\text{Salary} > 40000} (\text{Employee})
\]

<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:

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

Different semantics over sets or bags! Why?
<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>

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

<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>

Bag semantics

Set semantics

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

<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>

\[
\sigma_{\text{disease}=\text{‘heart’}}(\text{Patient})
\]

<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>

\[
\pi_{\text{zip,disease}}(\sigma_{\text{disease}=\text{‘heart’}}(\text{Patient}))
\]

<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 × 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>9999999999</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>9999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>9999999999</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 \nbowtie\ R2

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

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

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

\[ \mathbf{R} \bowtie \mathbf{S} = \Pi_{ABC}(\sigma_{R.B=S.B}(\mathbf{R} \times \mathbf{S})) \]
**NATURAL JOIN EXAMPLE 2**

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>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

\[ R_1 \bowtie_{\theta} R_2 = \sigma_{\theta} (R_1 \times R_2) \]

Here \( \theta \) can be any condition

No projection in this case!

For our voters/patients example:

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

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

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

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>

#### EQUIJOIN Operation

\[ P \bowtie_{\text{age} = \text{age}} V \]

<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>

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

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

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

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

\[ \sigma_{\text{psize}>10} \]

\[ \text{Part} \]

\[ \text{Supplier} \]

\[ \text{Supply} \]

\[ \pi_{sname} \]

\[ \text{Answer} \]
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
EXTENDED RA: OPERATORS ON BAGS

Duplicate elimination $\delta$

Grouping $\gamma$
- Takes in relation and a list of grouping operations (e.g., aggregates). Returns a new relation.

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

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

**Answer**

\[ \pi_{\text{city}, \text{q}} \]

\[ \sigma_{c > 100} \]

\[ \gamma_{\text{city}, \text{sum(quantity)} \rightarrow \text{q}, \text{count(*)} \rightarrow \text{c}} \]

sales(product, city, quantity)

T1, T2 = 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)

SELECT fields
FROM R, S, ...
WHERE condition
GROUP BY fields
HAVING condition
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)
```
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)
```

```
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)
```

De-Correlation
HOW ABOUT SUBQUERIES?

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

EXCEPT = set difference
HOW ABOUT SUBQUERIES?

\[
\begin{align*}
\text{(SELECT } & \text{ Q.sno} \\
\text{FROM } & \text{ Supplier Q} \\
\text{WHERE } & \text{ Q.sstate = 'WA')} \\
\text{EXCEPT} & \\
\text{(SELECT } & \text{ P.sno} \\
\text{FROM } & \text{ Supply P} \\
\text{WHERE } & \text{ P.price > 100)}
\end{align*}
\]

Finally…
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 → RA, then optimize RA
SUMMARY OF RA AND SQL

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

Example:

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

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