ADMINISTRATIVE MINUTIAE

• No class Wednesday

• HW2 Due Wednesday
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 as shorthand:
    • union $\cup$ and difference –
    • selection $\sigma$
    • projection $\pi$
    • cartesian product $\times$
    • natural join $\bowtie$
NATURAL JOIN

\[ R_1 \bowtie R_2 \]

**Meaning:** \( R_1 \bowtie R_2 = \Pi_A(\sigma_\theta(R_1 \times R_2)) \)
- Returns a relation where all attribute names are unambiguous

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

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

<table>
<thead>
<tr>
<th>R</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>S</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>

<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

#### 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

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

Here \( \theta \) can be any condition

No projection in this case!

For our voters/patients example:

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

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

<table>
<thead>
<tr>
<th>P\text{.age}</th>
<th>P\text{.zip}</th>
<th>P\text{.disease}</th>
<th>V\text{.name}</th>
<th>V\text{.age}</th>
<th>V\text{.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
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
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
\[ \pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10} \text{Part})) \]

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

Can be represented as trees as well
\[ \pi_{\text{sname}}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{\text{psize}>10} (\text{Part}))) \]
RELATIONAL ALGEBRA OPERATORS

Union \( \cup \), intersection \( \cap \), difference -
Selection \( \sigma \)
Projection \( \pi \)
Cartesian product \( X \), join \( \bowtie \)
(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.
SELECT city, sum(quantity) 
FROM sales 
GROUP BY city 
HAVING count(*) > 100

T1, T2 = temporary tables

Answer

\[ \Pi_{\text{city, q}} \]
\[ \sigma_{c > 100} \]
\[ \gamma_{\text{city, sum(quantity) \rightarrow q, count(*) \rightarrow c}} \]

\[ \text{sales(product, city, quantity)} \]
TYPICAL PLAN FOR A QUERY (1/2)

Answer

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

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

\[ \text{join condition} \]

\[ \text{join condition} \]

\[ R \]

\[ S \]

\[ \ldots \]

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?

```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)
```

De-Correlation

```sql
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)
```

**EXCEPT** = set difference

```
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)
```

**Un-nesting**

**Table**
- **Supplier(sno,sname,scity,sstate)**
- **Part(pno,pname,psize,pcolor)**
- **Supply(sno,pno,price)**
HOW ABOUT SUBQUERIES?

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

Finally...

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
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
RELATIONAL ALGEBRA TAKEAWAYS

• For a given query, be able write the equivalent relational algebra expression
• Given a relational algebra expression, write the equivalent query
• Understand what each are trying to get semantically
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

Datalog is an extension that can compute recursive queries
WHAT IS DATALOG?

Another query language for relational model

- Designed in the 80’s
- Simple, concise, elegant
- Extends relational queries with recursion

Relies on a logical framework for "record" selection
**DATALOG: FACTS AND RULES**

**Facts** = tuples in the database

**Rules** = queries

Schema

- Actor(id, fname, lname)
- Casts(pid, mid)
- Movie(id, name, year)
DATALOG: FACTS AND RULES

Facts = tuples in the database

Rules = queries

Actor(344759, 'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).
**DATLOG: FACTS AND RULES**

**Facts = tuples in the database**

Actor(344759, ‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).

**Rules = queries**

Q1(y) :- Movie(x,y,z), z=’1940’.
DATALOG: FACTS AND RULES

**Facts** = tuples in the database

- Actor(344759, ‘Douglas’, ‘Fowley’).
- Casts(344759, 29851).
- Casts(355713, 29000).

**Rules** = queries

Q1(y) :- Movie(x,y,z), z='1940'.

Find Movies made in 1940
DATALOG: FACTS AND RULES

Facts = tuples in the database

Actor(344759, 'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).

Rules = queries

Q1(y) :- Movie(x, y, z), z='1940'.
Q2(f, l) :- Actor(z, f, l), Casts(z, x),
         Movie(x, y, '1940').
DATALOG: FACTS AND RULES

Facts = tuples in the database

Actor(344759, 'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).

Rules = queries

Q1(y) :- Movie(x, y, z), z='1940'.
Q2(f, l) :- Actor(z, f, l), Casts(z, x),
Movie(x, y, '1940').

Find Actors who acted in Movies made in 1940
**DATALOG: FACTS AND RULES**

**Facts** = tuples in the database

- Actor(344759, 'Douglas', 'Fowley')
- Casts(344759, 29851)
- Casts(355713, 29000)
- Movie(7909, 'A Night in Armour', 1910)
- Movie(29000, 'Arizona', 1940)
- Movie(29445, 'Ave Maria', 1940)

**Rules** = queries

- Q1(y) :- Movie(x,y,z), z='1940'.
- Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,'1940')
- Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910), Casts(z,x2), Movie(x2,y2,1940)
DATAPLOG: FACTS AND RULES

Facts = tuples in the database

Actor(344759, ‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).

Rules = queries

Q1(y) :- Movie(x,y,z), z='1940’.
Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,’1940’).
Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910), Casts(z,x2), Movie(x2,y2,1940)

Find Actors who acted in a Movie in 1940 and in one in 1910
**DATALOG: FACTS AND RULES**

**Facts** = tuples in the database

- Actor(344759, ‘Douglas’, ‘Fowley’).
- Casts(344759, 29851).
- Casts(355713, 29000).

**Rules** = queries

- Q1(y) :- Movie(x, y, z), z='1940'.
- Q2(f, l) :- Actor(z, f, l), Casts(z, x), Movie(x, y, '1940').
- Q3(f, l) :- Actor(z, f, l), Casts(z, x1), Movie(x1, y1, 1910), Casts(z, x2), Movie(x2, y2, 1940).

**Extensional Database Predicates = EDB** = Actor, Casts, Movie

**Intensional Database Predicates = IDB** = Q1, Q2, Q3