ADMINISTRATIVE MINUTIAE

• HW3 and OQ3 out
• HW3 due next Friday
• OQ3 due next Wednesday
• HW4 out next week: on Datalog
• Midterm reminder: Feb 9th
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
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
\[ \pi_{sname}(\text{Supplier} \Join \text{Supply} \Join (\sigma_{psize>10}(\text{Part})) \]

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

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

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

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

Union $\cup$, intersection $\cap$, difference $-$
Selection $\sigma$
Projection $\pi$
Cartesian product $\times$, 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

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

Answer

\[
\Pi_{\text{city, q}} \left( T_2(\text{city, q, c}) \right)
\]

\[
\sigma_{c > 100} \left( T_1(\text{city, q, c}) \right)
\]

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

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

Answer

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

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

\[ \text{join condition} \]

\[ \text{join condition} \]

\[ \text{join condition} \]

\[ \text{R} \]

\[ \text{S} \]

\{ \]

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

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

\[
\begin{align*}
\sigma_{\text{having condition}} & \quad \vdash \quad \gamma_{\text{fields, sum/count/min/max(fields)}} \\
\pi_{\text{fields}} & \quad \vdash \quad \sigma_{\text{where condition}} \\
\end{align*}
\]

join condition

\[
\begin{align*}
\ldots & \quad \vdash \quad \ldots
\end{align*}
\]

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)

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)
```

EXCEPT = set difference
HOW ABOUT SUBQUERIES?

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

• Be able to get a query write the relational algebra expression equivalent to it
• 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

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

Rules = queries
**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).

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).
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'.

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

Rules = queries

Actors = tuples in the database

Facts:
- 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:
- 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).
**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)

Find Actors who acted in a Movie in 1940 and in one in 1910
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).

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
DATALOG: TERMINOLOGY

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

f, l = head variables
x,y,z= existential variables
MORE DATALOG TERMINOLOGY

R\textsubscript{i}(args\textsubscript{i}) called an \textit{atom}, or a \textit{relational predicate}

R\textsubscript{i}(args\textsubscript{i}) evaluates to true when relation R\textsubscript{i} contains the tuple described by args\textsubscript{i}.

• Example: Actor(344759, ‘Douglas’, ‘Fowley’) is true

In addition we can also have arithmetic predicates

• Example: z > ‘1940’.

Book uses AND instead of ,

Q(args) :- R1(args) \textbf{AND} R2(args) ....
SEMANTICS OF A SINGLE RULE
Meaning of a datalog rule = a logical statement!

\[ Q1(y) :\text{-} \text{Movie}(x,y,z), z='1940'. \]

- For all \( x, y, z \): if \( (x,y,z) \in \text{Movies} \) and \( z = '1940' \) then \( y \) is in \( Q1 \) (i.e. is part of the answer)
- \( \forall x \forall y \forall z \ [(\text{Movie}(x,y,z) \text{ and } z='1940') \Rightarrow Q1(y)] \)
- Logically equivalent:
  \( \forall y \ [(\exists x \exists z \text{Movie}(x,y,z) \text{ and } z='1940') \Rightarrow Q1(y)] \)
- Thus, non-head variables are called "existential variables"
- We want the smallest set \( Q1 \) with this property (why?)
A datalog program consists of several rules

Importantly, rules may be recursive!

Usually there is one distinguished predicate that’s the output

We will show an example first, then give the general semantics.
EXAMPLE
R encodes a graph

\[ R = \]

\begin{array}{cc}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
R encodes a graph

\[
\begin{align*}
R &= \{(x, y) : \neg R(x, y), T(x, y) : \neg R(x, z), T(z, y)\}
\end{align*}
\]

What does it compute?
What does it compute?

Initially: $T$ is empty.

$$R = \begin{array}{|c|c|} 
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}$$

$R$ encodes a graph

$T(x,y) :- R(x,y)$

$T(x,y) :- R(x,z), T(z,y)$
R encodes a graph

Initially:
T is empty.

First iteration:
T =

First rule generates this

Second rule generates nothing (because T is empty)

T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
**EXAMPLE**

R encodes a graph

\[
R(x,y) : - R(x,y), T(y,x)
\]

\[
T(x,y) : - R(x,z), T(z,y)
\]

Initially: T is empty.

First iteration:

\[
T =
\begin{array}{cc}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

Second iteration:

\[
T =
\begin{array}{cc}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
4 & 5 \\
1 & 1 \\
2 & 2 \\
1 & 3 \\
2 & 4 \\
1 & 5 \\
3 & 5 \\
\end{array}
\]

What does it compute?

First rule generates this

Second rule generates this

New facts
**EXAMPLE**

R encodes a graph

\[
R = \begin{array}{cccc}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

Initially: 
T is empty.

What does it compute?

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

First iteration: 
T =

Second iteration: 
T =

Third iteration: 
T =

R encodes a graph

T(x, y) :- R(x, y)

T(x, y) :- R(x, z), T(z, y)

First rule

Second rule

Both rules

New fact
EXAMPLE

R encodes a graph

R =

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Initially: T is empty.

T(x,y) :: R(x,y)
T(x,y) :: R(x,z), T(z,y)

First iteration:
T =

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Second iteration:
T =

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Third iteration:
T =

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fourth iteration:
T = (same)

No new facts.
DONE