CSE 344

APRIL 9TH – DATALOG
ADMINISTRATIVE MINUTIAE

• Midterm exam
  • Piazza poll
• OQ 2/3 Due Friday
• HW2 Due Wednesday
• HW3 Out 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: σ, π, δ, 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, surname, 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
\[ \pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{\text{psize}>10} (\text{Part}) \cup \sigma_{\text{pcolor}='red'} (\text{Part}) )) \]
\[ \pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{\text{psize}>10} \lor \text{pcolor}='red' (\text{Part})) ) \]

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

Union $\cup$, intersection $\cap$, difference $-$
Selection $\sigma$
Projection $\pi$
Cartesian product $\times$, join $\bowtie$
(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.
USING EXTENDED RA OPERATORS

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

T1, T2 = temporary tables

Answer

\[ \Pi_{\text{city}, q} (T2) \]
\[ \sigma_{c > 100} (T1) \]
\[ \gamma_{\text{city, sum(quantity)} \rightarrow q, \text{count(*)} \rightarrow c} \text{sales(product, city, quantity)} \]
TYPICAL PLAN FOR A QUERY (1/2)

Answer

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

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

\[ \text{join condition} \]

\[ \text{join condition} \]

\[ \text{...} \]

\[ \text{SELECT PROJECT-JOIN Query} \]

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

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)
```

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

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

**Un-nesting**

\[
\begin{align*}
&\text{EXCEPT} = \text{set difference}
\end{align*}
\]
HOW ABOUT SUBQUERIES?

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

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
RELATIONAL ALGEBRA TAKEAWAYS

• 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
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(29445, ‘Ave Maria’, 1940).
DATALOG: FACTS AND RULES

Facts = tuples in the database

Actor(344759, ‘Douglas’, ‘Fowley’).
Cast(344759, 29851).
Cast(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).
- 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

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').

Find Actors who acted in Movies made in 1940
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).
**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

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).

**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_i(args_i) called an **atom**, or a **relational predicate**

R_i(args_i) evaluates to true when relation R_i contains the tuple described by args_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), R2(args), ....

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

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

• For all x, y, z: if (x,y,z) ∈ Movies and z = ‘1940’ then y is in Q1 (i.e. is part of the answer)
• ∀x ∀y ∀z [(Movie(x,y,z) and z='1940') ⇒ Q1(y)]
• Logically equivalent:
  ∀y [( ∃x ∃z Movie(x,y,z) and z='1940') ⇒ 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.
R encodes a graph

\[ R = \begin{array}{cccccc}
1 & 2 & 1 & 3 & 4 & 4 & 5 \\
2 & 2 & 1 & 4 & 1 & 3 & 4 \\
5 & 4 & 4 & 5 \\
\end{array} \]
R encodes a graph

\[ R = \]

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

What does it compute?

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

R encodes a graph

\[ R = \]

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

Initially:
T is empty.

\[ T(x, y) \leftarrow R(x, y) \]
\[ T(x, y) \leftarrow R(x, z), T(z, y) \]

What does it compute?
EXAMPLE

R encodes a graph

\[
\begin{align*}
\text{R} = & \\
\begin{array}{cc}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\end{align*}
\]

Initially: T is empty.

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

First iteration:

\[
\begin{array}{c}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

First rule generates this

Second rule generates nothing (because T is empty)

What does it compute?
EXAMPLE

R encodes a graph

What does it compute?

First iteration:

T =

```
1 2
2 1
2 3
1 4
3 4
4 5
```

Second iteration:

```
1 2
2 1
2 3
3 4
4 5
1 1
2 2
1 3
2 4
1 5
3 5
```

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

First rule generates this
Second rule generates this

New facts

Initially:
T is empty.
R encodes a graph

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

First iteration:
T =

Second iteration:
T =

Third iteration:
T =

What does it compute?

R =

Initially: T is empty.

New fact

Both rules

First rule

Second rule
**EXAMPLE**

R encodes a graph

R =

<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>

Initially: T is empty.

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

First iteration:
T =

<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>

Second iteration:
T =

<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>

Third iteration:
T =

<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>

Fourth iteration:
T =

<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>

What does it compute?

No new facts.

DONE
DATACLASSIC SEMANTICS

Fixpoint semantics

Start:
\[
\begin{align*}
\text{IDB}_0 &= \text{empty relations} \\
\text{t} &= 0
\end{align*}
\]

Repeat:
\[
\begin{align*}
\text{IDB}_{t+1} &= \text{Compute Rules(EDB, IDB}_t) \\
\text{t} &= \text{t+1}
\end{align*}
\]

Until \(\text{IDB}_t = \text{IDB}_{t-1}\)

Remark: since rules are monotone:
\(\emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq \ldots\)

It follows that a datalog program w/o functions (+, *, ...) always terminates. (Why? In what time?)
DATALOG SEMANTICS

Minimal model semantics:

Return the IDB that

1) For every rule,
   \[ \forall \text{vars} \ (\text{Body}(\text{EDB}, \text{IDB}) \Rightarrow \text{Head}(\text{IDB})) \]

2) Is the smallest IDB satisfying (1)

Theorem: there exists a smallest IDB satisfying (1)
The fixpoint semantics tells us how to compute a datalog query.

The minimal model semantics is more declarative: only says what we get.

The two semantics are equivalent meaning: you get the same thing.
THREE EQUIVALENT PROGRAMS

R encodes a graph

\[
R(1,2) \\
R(2,1) \\
R(2,3) \\
R(1,4) \\
R(3,4) \\
R(4,5)
\]

\[
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), T(z,y)
\]
Here are *unsafe* datalog rules. What’s “unsafe” about them?

\[
\text{U1}(x,y) \ :- \ \text{ParentChild}(\text{“Alice”}, x), \ y \neq \text{“Bob”}
\]

\[
\text{U2}(x) \ :- \ \text{ParentChild}(\text{“Alice”}, x), \ !\text{ParentChild}(x, y)
\]
Here are *unsafe* datalog rules. What’s “unsafe” about them?

\[
\text{U1}(x,y) :- \text{ParentChild(“Alice”,x), } y \neq “Bob”
\]

\[
\text{U2}(x) :- \text{ParentChild(“Alice”,x), } \neg \text{ParentChild(x,y)}
\]

Holds for every y other than “Bob”

U1 = infinite!
Here are *unsafe* datalog rules. What’s “unsafe” about them?

**U1**\((x, y)\) :- ParentChild(“Alice”, x), y != “Bob”

Holds for every y other than “Bob”

**U2**(x) :- ParentChild(“Alice”, x), !ParentChild(x,y)

Want Alice’s childless children, but we get all children x (because there exists some y that x is not parent of y)
Here are *unsafe* datalog rules. What’s “unsafe” about them?

**U1(x,y)** :- ParentChild(“Alice”,x), y != “Bob”

**U2(x)** :- ParentChild(“Alice”,x), !ParentChild(x,y)

A datalog rule is *safe* if every variable appears in some positive relational atom.

Holds for every y other than “Bob”

U1 = infinite!

Want Alice’s childless children, but we get all children x (because there exists some y that x is not parent of y)