ADMINISTRATIVE MINUTIAE

• WQ3 due tonight

• HW3 out last night
  • more complex queries, better DBMS
  • you should have Azure token via email
  • you should be able to create a DB in Azure
REVIEW

• SQL & RA
  • languages for querying relational data
  • SQL is more declarative
  • RA used internally by DBMS as query plan

• Neither can express all queries that we might want to write...
  • (essential missing element is “recursion”)
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

• intersection of DBs with AI
**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).

**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
Rules = queries

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

Thought of as an inference rule:
if there exists x, y, z, f, l such that
Actor(z, f, l) and Casts(z, x) and Movie(x, y, 1940)
then we infer the fact Q2(f, l)

So far, still easily writable in SQL:

```
SELECT A.fname, A.lname
FROM Actor A, Casts C, Movie M
WHERE A.id = C.pid and C.mid = M.id AND M.year = 1940
```
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
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) ....
Meaning of a datalog rule = a logical statement !

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

- 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)
- \(\forall x \forall y \forall z \left[ (\text{Movie}(x,y,z) \land z='1940') \Rightarrow Q1(y) \right] \)
- Logically equivalent: 
\(\forall y \left[ (\exists x \exists z \; \text{Movie}(x,y,z) \land z='1940') \Rightarrow Q1(y) \right] \)
- Thus, non-head variables are called "existential variables"
- We want the \textit{smallest} set \(Q1\) with this property (why?)
DATALOG PROGRAM

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 =

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

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>

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

What does it compute?
**EXAMPLE**

R encodes a graph

\[ R = \]

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

Initially: T is empty.

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

What does it compute?
EXAMPLE
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 =

<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.

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

First iteration:
\[ \text{T} = \]

Second iteration:
\[ \text{T} = \]

First rule generates this
Second rule generates this

New facts

What does it compute?
EXAMPLE
R encodes a graph

R =

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

Initially:
T is empty.

First iteration:
T =

1 2
2 1
2 3
1 4
3 4

Second iteration:
T =

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

Third iteration:
T =

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

What does it compute?

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

New fact

Both rules
First rule
Second rule

EXAMPLE
R encodes a graph

R =

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

Initially:
T is empty.

First iteration:
T =

1 2
2 1
2 3
1 4
3 4

Second iteration:
T =

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

Third iteration:
T =

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

What does it compute?

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

New fact

Both rules
First rule
Second rule

EXAMPLE
R encodes a graph

R =

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

Initially:
T is empty.

First iteration:
T =

1 2
2 1
2 3
1 4
3 4

Second iteration:
T =

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

Third iteration:
T =

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

What does it compute?

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

New fact

Both rules
First rule
Second rule

EXAMPLE
R encodes a graph

R =

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

Initially:
T is empty.

First iteration:
T =

1 2
2 1
2 3
1 4
3 4

Second iteration:
T =

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

Third iteration:
T =

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

What does it compute?

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

New fact

Both rules
First rule
Second rule

EXAMPLE
R encodes a graph

R =

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

Initially:
T is empty.

First iteration:
T =

1 2
2 1
2 3
1 4
3 4

Second iteration:
T =

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

Third iteration:
T =

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

What does it compute?

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

New fact

Both rules
First rule
Second rule
**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) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)

First iteration: T =

<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>

Second iteration: T =

<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>

Third iteration: T =

<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>

Fourth iteration: T = (same)

What does it compute?

No new facts. DONE
DATALOG SEMANTICS

Fixpoint semantics

Start:

\[ \text{IDB}_0 = \text{empty relations} \]
\[ t = 0 \]

Repeat:

\[ \text{IDB}_{t+1} = \text{Compute Rules(EDB, IDB}_t) \]
\[ t = t+1 \]

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

Remark: since rules are monotone:

\[ \emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq \cdots \]

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

2) Is the smallest IDB satisfying (1)

Theorem: there exists a smallest IDB satisfying (1)
DATALOG SEMANTICS

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 -- you get the same thing
SAFE DATALOG RULES

Here are unsafe datalog rules. What’s “unsafe” about them?

\[
U_1(x, y) :- \text{ParentChild(“Alice”,} x), \ y \neq \ “Bob”
\]

\[
U_2(x) :- \text{ParentChild(“Alice”,} x), \ \neg \text{ParentChild(x,y)}
\]
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?

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

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

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).
SAFE DATALOG RULES

Here are *unsafe* datalog rules. What’s “unsafe” about them?

\[
U_1(x, y) :- \text{ParentChild}(\text{“Alice”}, x), y \neq \text{“Bob”}
\]

\[
U_2(x) :- \text{ParentChild}(\text{“Alice”}, x), \neg \text{ParentChild}(x, y)
\]

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

- Datalog can express things RA cannot
  - Recursive Queries

- Can Datalog express all queries in RA?
RELATIONAL ALGEBRA OPERATORS

Union \( \cup \), difference \(-\)
Selection \( \sigma \)
Projection \( \pi \)
Cartesian product \( \times \), join \( \bowtie \)
OPERATORS IN DATALOG

• Suppose we want $Q_1(\ldots)$ to contain all the values from $F_1(\ldots)$ and $F_2(\ldots)$
OPERATORS IN DATALOG

• Suppose we want Q1(...) to contain all the values from F1(...) and F2(...)
  • Q1(...) :- F1(...)
  • Q1(...) :- F2(...)

• What about for difference?
OPERATORS IN DATALOG

• Suppose we want Q1(...) to contain all the values from F1(...) and F2(...)
  • Q1(...) :- F1(...) 
  • Q1(...) :- F2(...) 

• What about for difference?
  • Q1(...) :- F1(...), !F2(...)


OPERATORS IN DATALOG

• Suppose we want Q1(…) to contain all the values from F1(…) and F2(…)
  • Q1(…) :- F1(…)
  • Q1(…) :- F2(…)

• What about for difference?
  • Q1(…) :- F1(…), !F2(…)
  • The variables (...) in F1 and F2 must be the same or else we have an unsafe rule
OPERATORS IN DATALOG

• Projection, from the variables $R_1, R_2, \ldots, R_k$
  select some subset of the variables
OPERATORS IN DATALOG

• Projection, from the variables $R_1, R_2, \ldots R_k$
  select some subset of the variables
  • $Q_1$(subset) :- Original(all_attributes)

• Selection: only return certain records from our knowledge base
OPERATORS IN DATALOG

• Projection, from the variables $R_1, R_2, \ldots R_k$
  select some subset of the variables
  • $Q_1(\text{subset}) : - \text{Original(all\_attributes)}$

• Selection: only return certain records from our knowledge base
  • $Q_1(\ldots) : - \text{Original(\ldots), selection\_criteria}$
OPERATORS IN DATALOG

• Cross product: find all the pairs between $R(a_1,a_2...) \text{ and } S(b_1,b_2...)\)
OPERATORS IN DATALOG

• Cross product: find all the pairs between \( R(a_1, a_2 \ldots) \) and \( S(b_1, b_2 \ldots) \)
  
  • \( Q1(a_1, b_1, a_2, b_2 \ldots) :- R(a_1, a_2 \ldots), S(b_1, b_2 \ldots) \)

• Joins?
  
  • Natural
• Cross product: find all the pairs between $R(a_1,a_2...)$ and $S(b_1,b_2...)$
  • $Q1(a_1,b_1,a_2,b_2...) :- R(a_1,a_2...), S(b_1,b_2...)$

• Joins?
  • Natural: $Q1(a,b,c) :- R(a,b), S(b,c)$
  • Theta
OPERATORS IN DATALOG

• Cross product: find all the pairs between \( R(a_1,a_2...) \) and \( S(b_1,b_2...) \)
  • \( Q1(a_1,b_1,a_2,b_2...) :- R(a_1,a_2...), S(b_1,b_2...) \)

• Joins?
  • Natural: \( Q1(a,b,c) :- R(a,b), S(b,c) \)
  • Theta: Cross product with selection
  • Equijoin: subset of Theta join
EXAMPLE

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

\[
\begin{align*}
\text{σ}_{\text{sstate='WA'}} & \text{ (Supplier)} \\
\text{σ}_{\text{Price > 100}} & \text{ (Supply)}
\end{align*}
\]
EXAMPLE

Datalog:

\[
\begin{align*}
\text{σ}_{\text{sstate} = 'WA'} & \text{σ}_{\text{Price} > 100} \\
\text{π}_{\text{sno}} & \text{π}_{\text{sno}} \\
\text{Supplier}(\text{sno}, \text{sname}, \text{scity}, \text{sstate}) & \text{Part}(\text{pno}, \text{pname}, \text{psize}, \text{pcolor}) & \text{Supply}(\text{sno}, \text{pno}, \text{price})
\end{align*}
\]
**EXAMPLE**

Datalog:

\[
\begin{align*}
Q1(\text{no}, \text{name}, \text{city}, \text{state}) & : - \\
& \phantom{: -} \text{Supplier}(\text{sno}, \text{sname}, \text{scity}, \text{sstate}), \\
& \phantom{: -} \text{sstate} = 'WA' \\
Q2(\text{sno}) & : - \text{Q1(\text{sno}, \text{n}, \text{c}, \text{s})} \\
Q3(\text{no}, \text{pno}, \text{price}) & : - \text{Supply}(\text{s}, \text{pn}, \text{pr}), \ \text{pr} > 100 \\
Q4(\text{sno}) & : - \text{Q3(\text{sno}, \text{pn}, \text{pr})} \\
\text{Result}(\text{sno}) & : - \text{Q2(\text{sno})}, \ !\text{Q4(\text{sno})}
\end{align*}
\]

\[
\begin{aligned}
\text{Supplier}(\text{sno}, \text{sname}, \text{scity}, \text{sstate}) \\
\text{Part}(\text{pno}, \text{pname}, \text{psize}, \text{pcolor}) \\
\text{Supply}(\text{sno}, \text{pno}, \text{price})
\end{aligned}
\]
MORE EXAMPLES W/O RECURSION

Find Joe's friends, and Joe's friends of friends.

\[
\begin{align*}
A(x) & \text{ :- Friend('Joe', x)} \\
A(x) & \text{ :- Friend('Joe', z), Friend(z, x)} \\
\end{align*}
\]

Friend(name1, name2)
Enemy(name1, name2)
MORE EXAMPLES W/O RECURSION

Find all of Joe's friends who do not have any friends except for Joe:

\[
\begin{align*}
\text{JoeFriends}(x) & : \text{Friend('Joe',x)} \\
\text{NonAns}(x) & : \text{JoeFriends}(x), \text{Friend}(x,y), y \neq 'Joe' \\
A(x) & : \text{JoeFriends}(x), \text{NOT NonAns}(x)
\end{align*}
\]
MORE EXAMPLES W/O RECURSION

Find all people such that all their enemies' enemies are their friends

Q: if someone doesn't have any enemies, do we want them in the answer?

A: Yes!

\[
\begin{align*}
\text{Everyone}(x) & : \text{Friend}(x,y) \\
\text{Everyone}(x) & : \text{Friend}(y,x) \\
\text{Everyone}(x) & : \text{Enemy}(x,y) \\
\text{Everyone}(x) & : \text{Enemy}(y,x) \\
\text{NonAns}(x) & : \text{Enemy}(x,y), \text{Enemy}(y,z), \text{NOT} \ \text{Friend}(x,z) \\
\text{A}(x) & : \text{Everyone}(x), \text{NOT} \ \text{NonAns}(x)
\end{align*}
\]
MORE EXAMPLES W/O RECURSION

Find all persons x that have a friend all of whose enemies are x's enemies.

Everyone(x) :- Friend(x,y)
...
NonAns(x) :- Friend(x,y), Enemy(y,z), NOT Enemy(x,z)
A(x) :- Everyone(x), NOT NonAns(x)
MORE EXAMPLES W/ RECURSION

Two people are in the *same generation* if they are siblings, or if they have parents in the same generation

Find all persons in the same generation with Alice
MORE EXAMPLES W/ RECURSION

Find all persons in the same generation with Alice
Let’s compute $SG(x,y) = \text{“}x,y \text{ are in the same generation}\text{”}$

\[
SG(x,y) :- \text{ParentChild}(p,x), \text{ParentChild}(p,y) \\
SG(x,y) :- \text{ParentChild}(p,x), \text{ParentChild}(q,y), SG(p,q) \\
Answer(x) :- SG(\text{“Alice”}, x)
\]
DATALOG SUMMARY

EDB (base relations) and IDB (derived relations)

Datalog program = set of rules

Datalog allows recursion

Some reminders about semantics:

- Multiple atoms in a rule mean join (or intersection)
- Variables with the same name are equi-join variables
- Multiple rules with same head mean union
CLASS OVERVIEW

Unit 1: Intro
Unit 2: Relational Data Models and Query Languages
Unit 3: Non-relational data
  • NoSQL
  • Json
  • SQL++
Unit 4: RDMBS internals and query optimization
Unit 5: Parallel query processing
Unit 6: DBMS usability, conceptual design
Unit 7: Transactions
Unit 8: Advanced topics (time permitting)