Introduction to Database Systems
CSE 414

Lecture 8: Datalog
Announcements

• HW3 posted (1 week)
  – Same dataset, more challenging queries
  – We have sent out all Azure codes if you filled out the form earlier
  – Make sure you use the cheapest tier
    • aka READ THE HW INSTRUCTIONS

– You should first run on sqlite in any case!
Class Overview

- Unit 1: Intro
- Unit 2: Relational Data Models and Query Languages
  - Data models, SQL, Datalog, Relational Algebra
- Unit 3: Non-relational data
- Unit 4: RDMBS internals and query optimization
- Unit 5: Parallel query processing
- Unit 6: DBMS usability, conceptual design
- Unit 7: Transactions
What is Datalog?

• Another query language for relational model
  – Designed in the 80’s
  – Simple, concise, elegant
  – Extends relational queries with recursion

• Today is a hot topic:
  – Souffle (we will use in HW4)
  – Eve http://witheve.com/
  – Differential datalog
    https://github.com/frankmcsherry/differential-dataflow
  – Beyond databases in many research projects: network protocols, static program analysis
• Open-source implementation of Datalog DBMS
• Under active development
• Commercial implementations are available
  – More difficult to set up and use
• “sqlite” of Datalog
  – Set-based rather than bag-based

• Install in your VM
  – Run `sudo yum install souffle in terminal`
  – More details in upcoming HW4
Why bother with yet another relational query language?
Example: storing FB friends

As a graph

As a relation

We will learn the tradeoffs of different data models later this quarter
Compute your friends graph

Friends(p1, p2, isFriend)

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>isFriend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>John</td>
<td>1</td>
</tr>
<tr>
<td>John</td>
<td>Mary</td>
<td>0</td>
</tr>
<tr>
<td>Mary</td>
<td>Phil</td>
<td>1</td>
</tr>
<tr>
<td>Phil</td>
<td>Peter</td>
<td>1</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

SELECT f.p2
FROM Friends as f
WHERE f.p1 = 'me' AND f.isFriend = 1

My own friends

SELECT f1.p2
FROM Friends as f1,
(SELECT f.p2
 FROM Friends as f
 WHERE f.p1 = 'me' AND f.isFriend = 1) as f2
WHERE f1.p1 = f2.p2 AND f1.isFriend = 1

My FoF

My FoFoF... My FoFoFoF...

Datalog allows us to write
recursive queries easily

When does it end???
Datalog: Facts and Rules

Facts = tuples in the database

Rules = queries
Datalog: Facts and Rules

Facts = tuples in the database

Rules = queries

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

.decl Actor(id:number, fname:symbol, lname:symbol)
.decl Casts(id:number, mid:number)
.decl Movie(id:number, name:symbol, year:number)

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

Table declaration

Types in Souffle:
number
symbol (aka varchar)

Insert data
Datalog: Facts and Rules

Facts = tuples in the database

Rules = queries

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

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

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

Find Movies made in 1940
**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.

**SQL**

```
SELECT name
FROM Movie
WHERE year = 1940
```
Datalog: Facts and Rules

Facts = tuples in the database

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Rules = queries

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

Order of variable matters!

Find Movies made in 1940

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

**Rules** = queries

Q1(y) :- Movie(iDontCare, 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(_, y, z), z=1940.

_ = “don’t care” variables

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(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Rules = queries

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

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).
Datalog: Facts and Rules

**Facts** = tuples in the database

- Actor(id, fname, lname)
- Casts(pid, mid)
- Movie(id, name, year)

**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(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

\[ Q(\text{args}) :- R1(\text{args}), R2(\text{args}), \ldots \]

- \( R_i(\text{args}_i) \) called an *atom*, or a *relational predicate*
- \( R_i(\text{args}_i) \) evaluates to true when relation \( R_i \) contains the tuple described by \( \text{args}_i \).
  - Example: \( \text{Actor}(344759, \text{‘Douglas’, ‘Fowley’}) \) is true
- In addition we can also have arithmetic predicates
  - Example: \( z > 1940 \).
- Book uses AND instead of
  \[ Q(\text{args}) :- R1(\text{args}) \text{ AND } R2(\text{args}) \ldots \]
Datalog program

• A Datalog program consists of several rules
• Importantly, rules may be recursive!
  – Recall CSE 143!
• 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 e.g., connected cities

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

R encodes a graph e.g., connected cities

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

What does it compute?

Multiple rules for the same IDB means OR
Example

\[ R = \{(x, y) : \neg R(x, y), T(x, y) : \neg R(x, z), T(z, y)\}. \]

Initially:

T is empty.

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

R encodes a graph e.g., connected cities

What does it compute?
Example

R encodes a graph e.g., connected cities

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

Initially:
T is empty.

First iteration:
T =

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

First rule generates this

Second rule generates nothing (because T is empty)

What does it compute?

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

R encodes a graph e.g., connected cities
R encodes a graph e.g., connected cities

Example

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

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

First iteration:
\[
T = \begin{array}{c|c}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
1 & 1 \\
2 & 2 \\
1 & 3 \\
2 & 4 \\
1 & 5 \\
3 & 5 \\
\end{array}
\]

Second iteration:

First rule generates this
Second rule generates this

What does it compute?

New facts
Example

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

What does it compute?

Initially:
\[ T = \]

First iteration:
\[ T = \]

Second iteration:
\[ T = \]

Third iteration:
\[ T = \]

R encodes a graph e.g., connected cities

<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>

New fact

Both rules
First rule
Second rule

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
Example

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

Initially: T is empty.

First iteration:

\[
R = \{(x, y) : T(x, y)\}
\]

Second iteration:

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

Third iteration:

T =

Fourth iteration

T =

No new facts.

DONE

What does it compute?

R encodes a graph e.g., connected cities

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

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

What does it compute?