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: RDBMS 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

Why bother with yet another relational query language?

• 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
Example: storing FB friends

<table>
<thead>
<tr>
<th>Person1</th>
<th>Person2</th>
<th>is_friend</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>

We will learn the tradeoffs of different data models later this quarter

Compute your friends graph

```
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
```

As a graph

As a relation

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(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).

When does it end???

Find Movies made in 1940

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

```
Q(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.

Note: Order of variable matters!

Find 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.

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).

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

Order of variable matters!

Find Movies made in 1940
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(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).

Q3(f, l) :- Actor(z, f, l), Casts(z, x1), Movie(x1, y1, 1910), Casts(z, x2), Movie(x2, y2, 1940).

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

More Datalog Terminology

- $R_i^j$ called an atom, or a relational predicate.
- $R_i^j$ evaluates to true when relation $R_i$ contains the tuple described by args.
  - Example: Actor(344759, 'Douglas', 'Fowley') is true.
- In addition we can also have arithmetic predicates
  - Example: $z > 1948$.
- Book uses AND instead of $\land$.

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

Example

\begin{verbatim}
R =
1  2
2  1
2  3
1  4
3  4
4  5
\end{verbatim}

Example

\begin{verbatim}
R =
1  2
2  1
2  3
1  4
3  4
4  5
\end{verbatim}

Example

\begin{verbatim}
R =
1  2
2  1
2  3
1  4
3  4
4  5
\end{verbatim}

What does it compute?

R encodes a graph e.g., connected cities

Example

\begin{verbatim}
T(x,y) :- R(x,y),
T(x,y) :- R(x,z), T(z,y).
\end{verbatim}

What does it compute?

First iteration:

T =
1  2
2  1
2  3
1  4
3  4
4  5

First rule generates this

Second rule generates nothing (because T is empty)

First rule generates this

New facts

Second rule generates this

New facts

Third iteration:

T =
1  2
2  1
2  3
1  4
3  4
4  5

Both rules

First rule

Second rule
Example

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

Initially:

T is empty.

First iteration:

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

Second iteration:

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

Third iteration:

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

Fourth iteration (same)

What does it compute?

No new facts.

DONE

Datalog Semantics

Fixpoint semantics

- Start:
  \( \text{IDB}_0 = \text{empty relations} \)
- 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:

\( \emptyset = \text{IDB}_0 \subseteq \text{IDB}_1 \subseteq \text{IDB}_2 \subseteq \ldots \)

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

Three Equivalent Programs

R encodes a graph e.g., connected cities

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

Right linear

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

Left linear

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

Non-linear

Question: which terminates in fewest iterations?

More Features

- Aggregates
- Grouping
- Negation

Aggregates

\[\text{[aggregate name]} < \text{var}> : \{ \text{relation to compute aggregate on} \}\]

- \( \min x : \{ \text{Actor}(x, y, _), y = 'John' \} \)
- \( \text{assign variable to the value of the aggregate} \)

<table>
<thead>
<tr>
<th>Aggregate in Souffle:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
</tr>
<tr>
<td>Min</td>
</tr>
<tr>
<td>Max</td>
</tr>
<tr>
<td>Sum</td>
</tr>
</tbody>
</table>

Aggregates

\[\text{[aggregate name]} < \text{var}> : \{ \text{relation to compute aggregate on} \}\]

- \( \min x : \{ \text{Actor}(x, y, _), y = 'John' \} \)
- \( \text{assign variable to the value of the aggregate} \)

What does this even mean???

Can't use variable that are not aggregated in the outer /head atoms
Counting

\[ Q(c) : c = \text{count} : (\text{Actor}_1, y, _) y = 'John' \]

Meaning (in SQL assuming no NULLs)

\[
\text{SELECT count(*) as c }
FROM \text{Actor as a}
\text{WHERE a.name = 'John'}
\]

Grouping

\[ Q(c, y) : \text{Movie}(x, y), c = \text{count} : (\text{Movie}(x, y)) \]

Meaning (in SQL)

\[
\text{SELECT m.year, count(*)}
FROM \text{Movie as m}
\text{GROUP BY m.year}
\]

Example

For each person, compute the total number of descendants

\[
// \text{for each person, compute his/her descendants}
D(x, y) := \text{ParentChild}(x, y).
D(x, z) := D(x, y), \text{ParentChild}(y, z).
\]

Example

For each person, compute the total number of descendants

\[
// \text{for each person, compute his/her descendants}
D(x, y) := \text{ParentChild}(x, y).
D(x, z) := D(x, y), \text{ParentChild}(y, z).
// \text{for each person, count the number of descendants}
\]
Example

For each person, compute the total number of descendants

// for each person, compute his/her descendants
D(x,y) :: ParentChild(x,y).
D(x,z) :: D(x,y), ParentChild(y,z).

// For each person, count the number of descendants
T(p,c) :: D(p,_), c = count : { D(p,y) }.

// Find the number of descendants of Alice
Q(d) :: -D("Alice",x), !D("Bob",x).

Negation: use “!”

Find all descendants of Alice, who are not descendants of Bob