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)
  - Beyond databases in many research projects: network protocols, static program analysis

Why bother with yet another relational query language?

Example: storing FB friends

As a graph

<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>
</tbody>
</table>

As a relation

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

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

My own friends

```sql
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

Datalog allows us to write recursive queries easily

Datalog: Facts and Rules

Facts = tuples in the database

Rules = queries

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

Find Movies made in 1940

```sql
SELECT name FROM Movie WHERE year = 1940
```
Facts = tuples in the database

Rules = queries

**Find Movies made in 1940**

- `Actor(344759, 'Douglas', 'Fowley').
  Casts(344759, 29851).
  Movie(7909, 'A Night in Armour', 1910).
  Movie(29000, 'Arizona', 1940).
  Movie(29445, 'Ave Maria', 1940).

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

Order of variable matters!

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

- `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(f,l) :
- Actor(z,f,l), Casts(z,x), Movie(x,y,1910).

Order of variable matters!

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

= 'don't care' variables
Facts = tuples in the database
Rules = queries

Datalog: Facts and Rules

CSE 414 - Autumn 2018

19

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

Datalog: Terminology

head body

atom atom atom (aka subgoal)

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

More Datalog Terminology

Q(args) :- R1(args), R2(args), ...

• R1(args1) called an atom or a relational predicate
• R1(args1) evaluates to true when relation R1 contains the tuple described by args1.
  – Example: Actor(344759, 'Douglas', 'Fowley') is true
• In addition we can also have arithmetic predicates
  – Example: z > 1940
• Book uses AND instead of ...

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

T(x,y) ⊨ R(x,y),
T(x,y) ⊨ R(x,z), T(z,y).

What does it compute?

Example

T(x,y) ⊨ R(x,y),
T(x,y) ⊨ R(x,z), T(z,y).

What does it compute?

Example

T(x,y) ⊨ R(x,y),
T(x,y) ⊨ R(x,z), T(z,y).

What does it compute?

Example

T(x,y) ⊨ R(x,y),
T(x,y) ⊨ R(x,z), T(z,y).

What does it compute?
More Features

- Aggregates
- Grouping
- Negation
// 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) := T(p,d), p = "Alice".
Negation: use “!”

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

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

// Compute the answer: notice the negation
Q(x) :- D("Alice",x), !D("Bob",x).

Safe Datalog Rules

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)

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

Holds for every y other than ‘Bob’
U1 = infinite!

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

Safe Datalog Rules

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

Safe
Datalog
Rules

Stratified Datalog

• Recursion does not cope well with aggregates or negation
• Example: what does this mean?
  A() :- !B().
  B() :- !A().

• A datalog program is stratified if it can be partitioned into strata
  – Only IDB predicates defined in strata 1, 2, ..., n may appear under ! or agg in stratum n+1.
• Many Datalog DBMSs (including souffle) accepts only stratified Datalog.
Stratified Datalog

- If we don’t use aggregates or negation, then the Datalog program is already stratified.

- If we do use aggregates or negation, it is usually quite natural to write the program in a stratified way.