Database Systems
CSE 414

Lecture 13: Datalog
(Ch 5.3–5.4)

Announcements

• HW3 is due Tomorrow
• WQ4 moved to Sunday
  – it will be useful review for the midterm
  – finish it early if you have time
• Midterm on Friday, April 28th, in class…

Midterm

• Content
  – Lectures 1 through 13 (today / Wednesday)
  – HW 1–3, WQ 1–4
• Closed book. No computers, phones, watches, etc.!
• Can bring one letter-sized piece of paper with notes, but…
  – test will not be about memorization
  – formulas provided for join algorithms & selectivity
  – can ask me during test about anything you could look up
• Similar in format & content to CSE 414 16sp midterm
  – CSE 344 tests include some things we did not cover

What is Datalog?

• Another query language for relational model
  – Simple and elegant
  – Initially designed for recursive queries
  – Some companies use datalog for data analytics
    • e.g. LogicBlox
  – Increased interest due to recursive analytics
• We discuss only recursion-free or non-recursive datalog and add negation

Datalog

• See book: 5.3 – 5.4

• See also: Query Language primer
  – article by Dan Suciu
  – covers relational calculus as well

Why Do We Learn Datalog?

• Datalog can be translated to SQL
  – Helps to express complex queries…
SQL Query vs Datalog (which would you rather write?)

### Why Do We Learn Datalog?

- Datalog can be translated to SQL
  - Helps to express complex queries
- Increase in datalog interest due to recursive analytics
- A query language that is closest to mathematical logic
  - Good language to reason about query properties
  - Can show that:
    1. Non-recursive datalog & RA have equivalent power
    2. Recursive datalog is strictly more powerful than RA
    3. Extended RA & SQL92 is strictly more powerful than datalog

### Some History

Early database history:
- 60s: network data models
- 70s: relational DBMSs
- 80s: OO-DBMSs

Ullman (1988) predicts KBMSs will replace DBMSs as they replaced what came before
- KBMS: knowledge-base
- combines data & logic (inferences)

Actually... relational DBMSs still dominate

Datalog: Facts and Rules

**Facts** = tuples in the database

<table>
<thead>
<tr>
<th>Fact</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>344759, 'Douglas', 'Fowley'</td>
</tr>
<tr>
<td>Actor</td>
<td>344759, 'Douglas', 'Fowley'</td>
</tr>
<tr>
<td>Casts</td>
<td>355713, 29000</td>
</tr>
<tr>
<td>Movie</td>
<td>7909, 'A Night in Armour', 1910</td>
</tr>
<tr>
<td>Movie</td>
<td>29000, 'Arizona', 1940</td>
</tr>
<tr>
<td>Movie</td>
<td>29445, 'Ave Maria', 1940</td>
</tr>
<tr>
<td>Movie</td>
<td>97909, 'A Night in Armour', 1910</td>
</tr>
<tr>
<td>Movie</td>
<td>29000, 'Arizona', 1940</td>
</tr>
<tr>
<td>Movie</td>
<td>29445, 'Ave Maria', 1940</td>
</tr>
</tbody>
</table>

**Rules** = queries

- Q1(y) :- Movie(x,y,'1940')

Find Movies made in 1940

### Datalog: Facts and Rules

**Facts** = tuples in the database

<table>
<thead>
<tr>
<th>Fact</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>344759, 'Douglas', 'Fowley'</td>
</tr>
<tr>
<td>Actor</td>
<td>344759, 'Douglas', 'Fowley'</td>
</tr>
<tr>
<td>Casts</td>
<td>355713, 29000</td>
</tr>
<tr>
<td>Movie</td>
<td>7909, 'A Night in Armour', 1910</td>
</tr>
<tr>
<td>Movie</td>
<td>29000, 'Arizona', 1940</td>
</tr>
<tr>
<td>Movie</td>
<td>29445, 'Ave Maria', 1940</td>
</tr>
<tr>
<td>Movie</td>
<td>97909, 'A Night in Armour', 1910</td>
</tr>
<tr>
<td>Movie</td>
<td>29000, 'Arizona', 1940</td>
</tr>
<tr>
<td>Movie</td>
<td>29445, 'Ave Maria', 1940</td>
</tr>
</tbody>
</table>

**Rules** = queries

- Q1(y) :- Movie(x,y,'1940')
- Q2(f, l) :- Actor(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, 29445).
- Movie(29445, 'Ave Maria', 1940).
- Movie(29000, 'Arizona', 1940).
- Casts(355713, 29000).
- Casts(344759, 29851).
- Actor(344759, 'Douglas', 'Fowley').

**Rules** = queries

- Q1(y) = Movie(x, '1940').
- Q2(f, l) = Actor(z, f), Casts(z, x), Movie(x, '1940').
- Q3(f, l) = Actor(z, f), Casts(z, x1), Movie(x1, y1, 1910), Casts(z, x2), Movie(x2, y2, 1940).
- Q4(x) = Movie(x, '1940').

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

### Datalog: Terminology

**Head**

- f, l = head variables

**Body**

- x, y, z = existential variables

- atom

- atom (aka subgoal)

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

### More Datalog Terminology

- **R(args)** is called an atom, or a relational predicate
- **R(args)** evaluates to true when relation **R** contains the tuple described by args.
  - Example: Actor(344759, 'Douglas', 'Fowley') is true

- In addition to relational predicates, we can also have arithmetic predicates:
  - Example: z = '1940'.

### Semantics

- **Meaning of a datalog rule** = a logical statement!

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

- **Means:**
  - \( \forall x. \forall y. \forall z. \left[ \text{Movie}(x, y, z) \land z = '1940' \right] \Rightarrow Q1(y) \)
  - and Q1 is the smallest relation that has this property

- **Note:** logically equivalent to:
  - \( \forall y. \left[ \exists x \exists z. \text{Movie}(x, y, z) \land z = '1940' \right] \Rightarrow Q1(y) \)
  - That’s why vars not in head are called “existential variables”.

### Datalog program

A datalog program is a collection of one or more rules

Each rule expresses the idea that, from certain combinations of tuples in certain relations, we may infer that some other tuple must be in some other relation or in the query answer.

Example: Find all actors with Bacon number ≤ 2

- B0(x) = Actor(x, 'Kevin', 'Bacon')
- B1(x) = Actor(x, f), Casts(x, z), Casts(y, z), B0(y)
- B2(x) = Actor(x, f), Casts(x, z), Casts(y, z), B1(y)
- Q4(x) = B0(x)
- Q4(x) = B1(x)
- Q4(x) = B2(x)

**Note:** Q4 means the union of B0, B1, & B2.
Recursive Datalog

- In Datalog, rules can be recursive
  
  \[
  \text{Path}(x, y) : \neg \text{Edge}(x, y), \\
  \text{Path}(x, y) : \text{Path}(x, z), \text{Edge}(z, y).
  \]

- We'll focus on non-recursive Datalog

Path encodes a graph
Path finds all paths

Datalog with negation

Find all actors who do not have a Bacon number < 2

\[
\begin{align*}
\text{B0}(x) & : \text{Actor}(x, \text{'Kevin'}, \text{'Bacon'}) \\
\text{B1}(x) & : \text{Actor}(x, f, l), \text{Casts}(x, z), \text{Casts}(y, z), \text{B0}(y) \\
\text{Q6}(x) & : \text{Actor}(x, f, l), \neg \text{B0}(x), \neg \text{B1}(x)
\end{align*}
\]

Safe Datalog Rules

Here are unsafe Datalog rules. What's "unsafe" about them?

\[
\begin{align*}
\text{U1}(x, y) & : \text{Movie}(x, z, 1994), y > 1910 \\
\text{U2}(x) & : \text{Movie}(x, z, 1994), \neg \text{Casts}(u, x)
\end{align*}
\]

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

Datalog vs Relational Algebra

- Every expression in standard relational algebra can be expressed as a Datalog query

- But operations in the extended relational algebra (grouping, aggregation, and sorting) have no corresponding features in the version of Datalog that we discussed today

- Similarly, Datalog can express recursion, which relational algebra cannot

RA to Datalog by Examples

Schema for our examples:

\[
\begin{align*}
R(A, B, C) \\
S(D, E, F) \\
T(G, H)
\end{align*}
\]
Union \( R(A,B,C) \cup S(D,E,F) \)

- \( U(x,y,z) :- R(x,y,z) \)
- \( U(x,y,z) :- S(x,y,z) \)

Intersection \( R(A,B,C) \cap S(D,E,F) \)

- \( I(x,y,z) :- R(x,y,z), S(x,y,z) \)

Selection: \( \sigma_{x>100 \text{ and } y=\text{some string}} (R) \)

- \( L(x,y,z) :- R(x,y,z), x > 100, y=\text{some string} \)

Selection \( x>100 \) or \( y=\text{some string} \)

- \( L(x,y,z) :- R(x,y,z), x > 100 \)
- \( L(x,y,z) :- R(x,y,z), y=\text{some string} \)

Equi-join: \( R \bowtie_{R.A=S.D \text{ and } R.B=S.E} S \)

- \( J(x,y,z,u,v,w) :- R(x,y,z), S(u,v,w), x=u, y=v \)
- \( J(x,y,z,w) :- R(x,y,z), S(x,y,w) \)

Projection \( \pi_x (R) \)

- \( P(x) :- R(x,y,z) \)

To express set difference \( R - S \),
we add negation

- \( D(x,y,z) :- R(x,y,z), \text{not } S(x,y,z) \)
Examples

R(A,B,C)
S(D,E,F)
T(G,H)

Translate: $\Pi_b c_b = 3 \; (R)$
$\forall a, b, c \; R(a, b, c), \; b = 3$
$\forall a \; B(a, b, c)

Underscore used to denote an "anonymous variable", a variable that appears only once.

Examples

R(A,B,C)
S(D,E,F)
T(G,H)

Translate: $\Pi_b c_b = 3 \; (R)$
$\forall a \; R(a, 3, _)$

More Examples

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

$\forall x \; \text{Friend('Joe', x)}$
$\forall x \; \text{Friend('Joe', z)}, \; \text{Friend(z, x)}$

More Examples

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

$\forall x \; \text{Friend('Joe', x)}$
$\forall x \; \text{Friend(y, x)}, \; y \neq 'Joe'$
$\forall x \; \text{JoeFriends(x)}$
$\forall x \; \text{NonAns(x)}$

More Examples

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

$\forall x \; \text{NonAns(x)}$
$\forall x \; \text{Enemy(x, y)}, \; \text{Enemy(y, z)}, \; \neg \text{Friend(x, z)}$
$\forall x \; \text{Everyone(x)}$
$\forall x \; \neg \text{NonAns(x)}$

$\forall x \; \text{Everyone(x)}$
$\forall x \; \text{Friend(x, y)}$
$\forall x \; \text{Enemy(x)}$
$\forall x \; \text{Enemy(y, x)}$
$\forall x \; \text{Enemy(x, y)}$
More Examples

Find all persons $x$ that have only friends all of whose enemies are $x$’s enemies.

NonAns($x$) : $\neg$Friend($x, y$), Enemy($y, z$), $\neg$Enemy($x, z$)

A($x$) :
$\neg$NonAns($x$)

Datalog Summary

- facts (extensional relations) and rules (intensional relations)
  - rules can use relations, arithmetic, union, intersect, ...
- As with SQL, existential quantifiers are easier
  - use negation to handle universal
- Everything expressible in RA is expressible in non-recursive datalog and vice versa
  - recursive datalog can express more than (extended) RA
  - extended RA can express more than recursive datalog

Midterm Concept Review I

- relational data model
  - set semantics vs bag semantics
  - primary & secondary keys
  - foreign keys
  - schemas
- SQL
  - CREATE TABLE
  - SELECT-FROM-WHERE (SFW)
  - joins: inner vs outer, natural
  - group by & aggregation
  - ordering
  - CREATE INDEX

Midterm Concept Review II

- relational queries
  - languages for writing them:
    - standard relational algebra
    - datalog (even without recursion)
    - SQL (even without grouping / aggregation)
  - monotone queries are a proper subset
  - SFW queries (i.e., w/out subqueries) are monotone

Midterm Concept Review III

- types of indexes
  - B+ tree vs hash
    - hash indexes use at most 2 disk accesses
    - B+ tree can be used for $=$ predicates
    - B+ tree index on ($X, Y$) allows searching for $X=x$ matches
  - clustered vs non-clustered
    - selectivity above 1-2% $\Rightarrow$ not helped by non-clustered indexes
- cost-based query optimization
  - consider choices over logical and physical query plans
    - most important choice in latter is choice of join algorithm
    - those include nested loop, sorted merge, hash, and indexed joins
    - primary goal of the optimizer is to avoid really bad plans