Introduction to Data Management
CSE 344

Lecture 13: Datalog
(Ch 5.3–5.4)
Announcements

• HW3 is due Tomorrow

<table>
<thead>
<tr>
<th>Spent</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>7</td>
</tr>
<tr>
<td>$1</td>
<td>7</td>
</tr>
<tr>
<td>$2</td>
<td>10</td>
</tr>
<tr>
<td>$3</td>
<td>7</td>
</tr>
<tr>
<td>$4</td>
<td>9</td>
</tr>
<tr>
<td>$6</td>
<td>1</td>
</tr>
<tr>
<td>$21</td>
<td>1</td>
</tr>
</tbody>
</table>
Announcements

• HW3 is due Tomorrow

• WQ4 is due next Monday
  – it will be useful review for the midterm
  – finish it early if you have time

• Midterm on Friday, July 21st, in class…
  – All the web quizzes are open if that helps you study
Midterm

• Content
  – Lectures 1 through 13 (today / Monday)
  – 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 344 17wi midterm
  – Previous midterms on course webpage
Domain independence

- $Q(x) = \forall y. \text{ Likes}(x,y)$ is domain dependent
  - Suppose Likes = \{(d1,b1), (d1,b2)\}
  - What if we evaluate y over \{ b1, b2 \}?
  - What about \{ b1, b2, b3 \}? \\

- $Q(x) = \exists y. \text{ Likes}(x,y)$ is domain independent
  - What if we evaluate y over \{ b1, b2 \}?
  - What about \{ b1, b2, b3 \}?

- $Q(x) = \text{ IsBar}(x) \land \forall y. \text{ Serves}(x,y) \Rightarrow \text{ IsBeer}(y)$ is domain independent
  - Let IsBeer = \{ b1, b2 \}, IsBar = \{ bar1 \}, and Serves = \{(bar1, b1), (bar1, b2)\}
  - What if we evaluate y over \{ b1, b2 \}? \{ b1, b2, b3 \}?
Today: YAQL
(Yet Another Query Language)

Datalog
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
USE AdventureWorks2008R2;
GO
WITH DirectReports (ManagerID, EmployeeID, Title, DeptID, Level)
AS
(
  -- Anchor member definition
  SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID, 0 AS Level
  FROM dbo.MyEmployees AS e
  INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
    ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
  WHERE ManagerID IS NULL
  UNION ALL
  -- Recursive member definition
  SELECT e.ManagerID, e.EmployeeID, e.Title, edh.DepartmentID, Level + 1
  FROM dbo.MyEmployees AS e
  INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
    ON e.EmployeeID = edh.BusinessEntityID AND edh.EndDate IS NULL
  INNER JOIN DirectReports AS d
    ON e.ManagerID = d.EmployeeID
)

-- Statement that executes the CTE
SELECT ManagerID, EmployeeID, Title, DeptID, Level
FROM DirectReports
INNER JOIN HumanResources.Department AS dp
  ON DirectReports.DeptID = dp.DepartmentID
WHERE dp.GroupName = N'Sales and Marketing' OR Level = 0;

GO

DirectReports(eid, 0) :-
  Employee(eid),
  not Manages(_, eid)
DirectReports(eid, level+1) :-
  DirectReports(mid, level),
  Manages(mid, eid)

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
Datalog

We do not run datalog in 344; to try out on you own:

• Download DLV (http://www.dbai.tuwien.ac.at/proj/dlv/)
  Run DLV on this file

• Can also try Flix
  (http://flix.github.io/try/)

• Or pydatalog
  (https://sites.google.com/site/pydatalog/home)

• Or DrRacket
  http://www.racket-lang.org/

Similar to Prolog - logical programming language
Datalog: Terminology

\[ Q2(f, l) :- \text{Actor}(z,f,l), \text{Casts}(z,x), \text{Movie}(x,y,’1940’). \]

- \( f, l \) = head variables
- \( x,y,z \) = existential variables
More Datalog Terminology

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

Book writes:
Q(args) :- R1(args) AND R2(args) AND ....

• $R_i(args_i)$ is 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 to relational predicates, we can also have arithmetic predicates
  – Example: $z$=‘1940’.
Datalog: Facts and Rules

Facts = tuples in the database
Rules = queries

Find Movies made in 1940

Q1(y) :- Movie(x,y,’1940’).

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

Actor(344759,‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).
Datalog: Facts and Rules

Find Actors who acted in Movies made in 1940

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

\[ \text{Q2}(f, l) = \text{Movie}(x, y, '1940') \land \text{Casts}(z, x) \land \text{Actor}(z, f, l) \]
Datalog: Facts and Rules

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

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, '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

<table>
<thead>
<tr>
<th>Actor(pid, fname, lname)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Casts(pid, mid)</td>
</tr>
<tr>
<td>Movie(mid, name, year)</td>
</tr>
</tbody>
</table>

**Rules** = queries

- Q1(y) :- Movie(x,y,’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
Semantics

• Meaning of a datalog rule = a logical statement!

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

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

• Note: logically equivalent to:
  – \( \forall y. [(\exists x. \exists z. \text{Movie}(x,y,z) \text{ and } z='1940') \Rightarrow Q1(y)] \)
  – 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,l), Casts(x,z), Casts(y,z), B0(y)
B2(x) :- Actor(x,f,l), 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

\[
\text{Path}(x, y) :- \text{Edge}(x, y).
\]

\[
\text{Path}(x, y) :- \text{Path}(x, z), \text{Edge}(z, y).
\]

• We’ll focus on non-recursive datalog

Edge encodes a graph
Path finds all paths
Datalog with negation

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

B0(x) :- Actor(x,'Kevin', 'Bacon')
B1(x) :- Actor(x,f,l), Casts(x,z), Casts(y,z), B0(y)
Q6(x) :- Actor(x,f,l), not B1(x), not B0(x)
Safe Datalog Rules

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

- $\text{U0}(x) \ :- \ y > 1910$
- $\text{U1}(x, y) \ :- \ \text{Movie}(x, z, 1994), \ y > 1910$
- $\text{U2}(x) \ :- \ \text{Movie}(x, z, 1994), \ \text{not \ Casts}(u, x)$

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
Datalog vs Relational Algebra

- Standard RA
- Datalog + neg

- Extended RA
- Grouping & aggregation
- Datalog + neg + recursion
RA to Datalog by Examples

Schema for our examples:

R(A,B,C)
S(D,E,F)
T(G,H)
RA to Datalog by Examples

Union $R(A,B,C) \cup S(D,E,F)$

$U(x,y,z) \leftarrow R(x,y,z)$
$U(x,y,z) \leftarrow S(x,y,z)$
RA to Datalog by Examples

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

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

What does this do?
$I(x, y, z) :- R(x, y, z), S(a, b, y)$

Cross Product
RA to Datalog by Examples

Selection: \( \sigma_{x>100 \text{ and } y='\text{some string}'}(R) \)
L(x,y,z) :- R(x,y,z), x > 100, y='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='some string'
RA to Datalog by Examples

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

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

\[ J(x,y,z,w) \ :- \ R(x,y,z), S(x,y,w) \]
RA to Datalog by Examples

Projection $\pi_x(R)$

$P(x) :- R(x,y,z)$
RA to Datalog by Examples

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: \[ \Pi_A (\sigma_{B=3} (R) ) \]
\[ B(a,b,c) :- R(a,b,c), b=3 \]
\[ A(a) :- B(a,b,c) \]
Examples

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

Translate: $\Pi_A(\sigma_{B=3} (R) )$
A(a) :- R(a,3,\_)

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: \( \Pi_A(\sigma_{B=3}(R) \bowtie_{R.A=S.D} \sigma_{E=5}(S)) \)
A(a) :- R(a,3,\_), S(a,5,\_)
Friend(name1, name2)
Enemy(name1, name2)

More Examples

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

\[ A(x) := \text{Friend('Joe', } x) \]
\[ A(x) := \text{Friend('Joe', } y) \text{ Friend}(y, x) \]

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

x \neq \text{Joe}
More Examples

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

\[
\begin{align*}
\text{JoeFriends}(x) & : \text{Friend}('Joe',x) \\
\text{NonAns}(x) & : \text{Friend}(y,x), y \neq 'Joe' \\
A(x) & : \text{JoeFriends}(x), \text{not NonAns}(x)
\end{align*}
\]
Find all people such that all their enemies' enemies are their friends

- Q: if someone doesn't have any enemies nor friends, do we want them in the answer?
- A: Yes!

```prolog
Everyone(x) :- Friend(x,y)
Everyone(x) :- Friend(y,x)
Everyone(x) :- Enemy(x,y)
Everyone(x) :- Enemy(y,x)
NonAns(x) :- Enemy(x,y),Enemy(y,z), NOT Friend(x,z)
A(x) :- Everyone(x), NOT NonAns(x)
```
Friend(name1, name2)
Enemy(name1, name2)

More Examples

Find all persons x that have **only** friends **all** of whose enemies are x's enemies.

```prolog
NonAns(x) :- Friend(x,y), Enemy(y,z), not Enemy(x,z)
A(x) :- not NonAns(x)
```

```prolog
NonAns(x) :- Friend(x,y), Enemy(y,z), not Enemy(x,z)
A(x) :- Everyone(x), not NonAns(x)
```

what's wrong with this?
Datalog Summary

• facts (extensional relations - EDBs) and rules (intensional relations - IDBs)
  – rules can use relations, arithmetic, union, intersect, …

• As with SQL, existential quantifiers are easier
  – use negation to handle universal
Datalog Summary

• 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

• Some reminders about semantics:
  – Multiple atoms in a rule mean join (or intersection)
  – Variables with the same name are join variables
  – Multiple rules with same head mean union
Using what we have learned

How to write a complex SQL query:
• Write it in RC
• Translate RC to datalog
• Translate datalog to SQL

Take shortcuts when you know what you’re doing
From RC to Datalog\textsuperscript{-} to SQL

**Query:** Find drinkers that like some beer so much that they frequent all bars that serve it

\[
Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\text{Serves}(z, y) \Rightarrow \text{Frequents}(x, z))
\]
Query: Find drinkers that like some beer so much that they frequent all bars that serve it

Step 1: Replace $\forall$ with $\exists$ using de Morgan’s Laws

$$Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\text{Serves}(z, y) \Rightarrow \text{Frequents}(x, z))$$

$P \Rightarrow Q$ same as $\neg P \lor Q$

$\forall x \ P(x)$ same as $\neg \exists x \ \neg P(x)$

$\neg (\neg P \lor Q)$ same as $P \land \neg Q$
From RC to Datalog\(^{-}\) to SQL

**Query:** Find drinkers that like some beer so much that they frequent all bars that serve it

\[
Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\text{Serves}(z, y) \Rightarrow \text{Frequents}(x, z))
\]

**Step 1:** Replace \( \forall \) with \( \exists \) using de Morgan’s Laws

\[
Q(x) = \exists y. \text{Likes}(x, y) \land \neg \exists z. (\text{Serves}(z, y) \land \neg \text{Frequents}(x, z))
\]

**Step 2:** Make sure the query is domain independent

\[
Q(x) = \exists y. \text{Likes}(x, y) \land \neg \exists z. (\text{Likes}(x, y) \land \text{Serves}(z, y) \land \neg \text{Frequents}(x, z))
\]
From RC to Datalog to SQL

\[
Q(x) = \exists y. \text{Likes}(x, y) \land \neg \exists z. (\text{Likes}(x, y) \land \text{Serves}(z, y) \land \neg \text{Frequents}(x, z))
\]

\[
H(x, y)
\]

Step 3: Create a datalog rule for each subexpression;
(shortcut: only for “important” subexpressions)

\[
H(x, y) \ :- \ \text{Likes}(x, y), \text{Serves}(z, y), \neg \text{Frequents}(x, z)
\]

\[
Q(x) \ :- \ \text{Likes}(x, y), \neg H(x, y)
\]
From RC to Datalog™ to SQL

H(x,y) :- Likes(x,y), Serves(z,y), not Frequents(x,z)
Q(x) :- Likes(x,y), not H(x,y)

Step 4: Write it in SQL

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Likes L2, Serves S
   WHERE … … …)
```
Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to Datalog⁻ to SQL

\[
\begin{align*}
H(x,y) & \quad :\quad \text{Likes}(x,y), \text{Serves}(z,y), \text{not Frequents}(x,z) \\
Q(x) & \quad :\quad \text{Likes}(x,y), \text{not } H(x,y)
\end{align*}
\]

**Step 4: Write it in SQL**

```
SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
  (SELECT * FROM Likes L2, Serves S
   WHERE L2.drinker=L.drinker and L2.beer=L.beer
   and L2.beer=S.beer
   and not exists (SELECT * FROM Frequents F
      WHERE F.drinker=L2.drinker
      and F.bar=S.bar))
```
From RC to Datalog to SQL

Unsafe rule

H(x,y) :- Likes(x,y), Serves(z,y), not Frequents(x,z)
Q(x) :- Likes(x,y), not H(x,y)

Improve the SQL query by using an unsafe datalog rule

SELECT DISTINCT L.drinker FROM Likes L
WHERE not exists
(SELECT * FROM Serves S
WHERE L.beer=S.beer
  and not exists (SELECT * FROM Frequents F
    WHERE F.drinker=L.drinker
      and F.bar=S.bar)))
Summary: all these formalisms are equivalent!

- We have seen these translations:
  - RA $\rightarrow$ datalog$^\neg$
  - RC $\rightarrow$ datalog$^\neg$

- Practice at home, and read *Query Language Primer*:
  - Nonrecursive datalog$^\neg$ $\rightarrow$ RA
  - RA $\rightarrow$ RC

- Summary:
  - RA, RC, and non-recursive datalog$^\neg$ can express the same class of queries, called Relational Queries