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
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

Some History

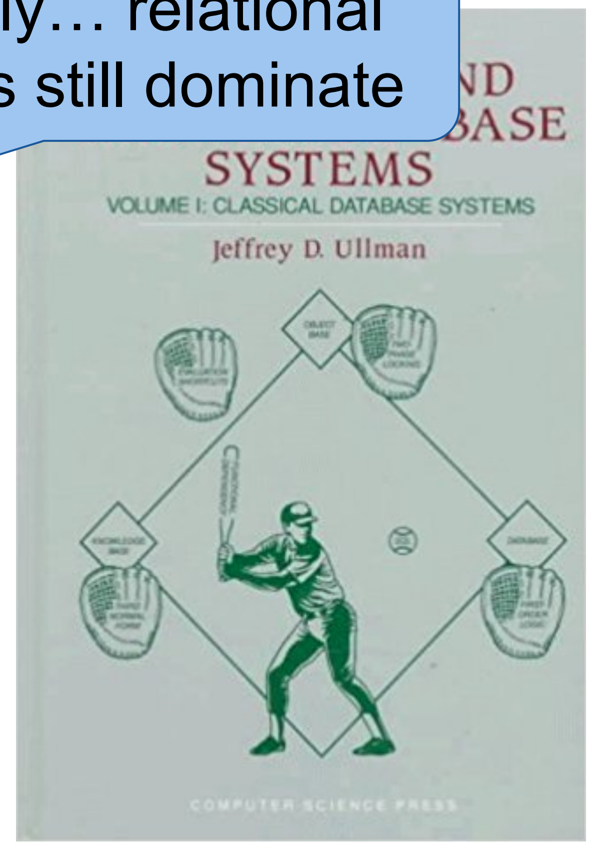
Early database history:

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

Actually… relational DBMSs still dominate

Ullman (1988) predicts KBMSs will replace DBMSs as they replaced what came before

- KBMS: knowledge-base
- combines data & logic (inferences)



Datalog

We won't run datalog in 414. Try out on you own:

- Download DLV (http://www.dlvsystem.com/dlv/)
- Run DLV on this file
- Can also try IRIS

  (http://www.iris-reasoner.org/demo)

```
parent(william, john).
parent(john, james).
parent(james, bill).
parent(sue, bill).
parent(james, carol).
parent(sue, carol).

male(john).
male(james).
female(sue).
male(bill).
female(carol).

grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
father(X, Y) :- parent(X, Y), male(X).
mother(X, Y) :- parent(X, Y), female(X).
brother(X, Y) :- parent(P, X), parent(P, Y), male(X), X != Y.
sister(X, Y)  :- parent(P, X), parent(P, Y), female(X), X != Y.
```

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

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,'1940').
```

Find Movies made in 1940

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

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,‘1940’).

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

Find Actors who acted in Movies made in 1940

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

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,'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

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

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,'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

$$\overbrace{\text{Q2(f, l)}}^{\text{head}} \text{ :- } \overbrace{\overbrace{\text{Actor(z,f,l)}}^{\text{atom}}\text{, }\overbrace{\text{Casts(z,x)}}^{\text{atom}}\text{, }\overbrace{\text{Movie(x,y,'1940')}}^{\text{atom (aka subgoal)}}}^{\text{body}}\text{.}$$

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'.

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

Semantics

- Meaning of a datalog rule = a logical statement !

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

- Means:
  - $\forall$x. $\forall$y. $\forall$z. [(Movie(x,y,z) 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. Movie(x,y,z) and z='1940') $\Rightarrow$ Q1(y)]
  - That's why vars not in head are called "existential variables".

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

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)
```

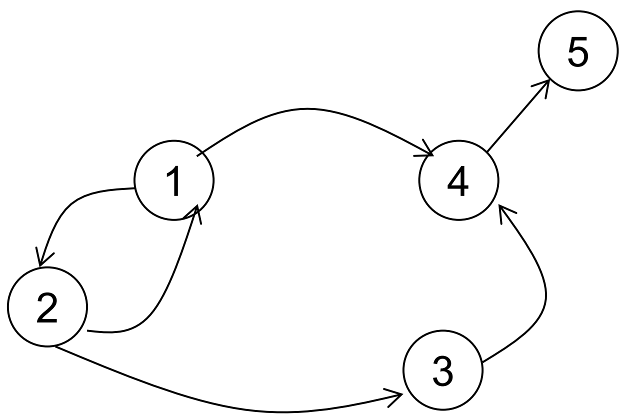
Note: Q4 means the *union* of B0, B1, & B2

Recursive Datalog

- In datalog, rules can be recursive

```
Path(x, y) :- Edge(x, y).
Path(x, y) :- Path(x, z), Edge (z, y).
```

- We'll focus on non-recursive datalog



Edge encodes a graph
Path finds all paths

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

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)
```

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

Safe Datalog Rules

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

U1(x,y) :- Movie(x,z,1994), y>1910

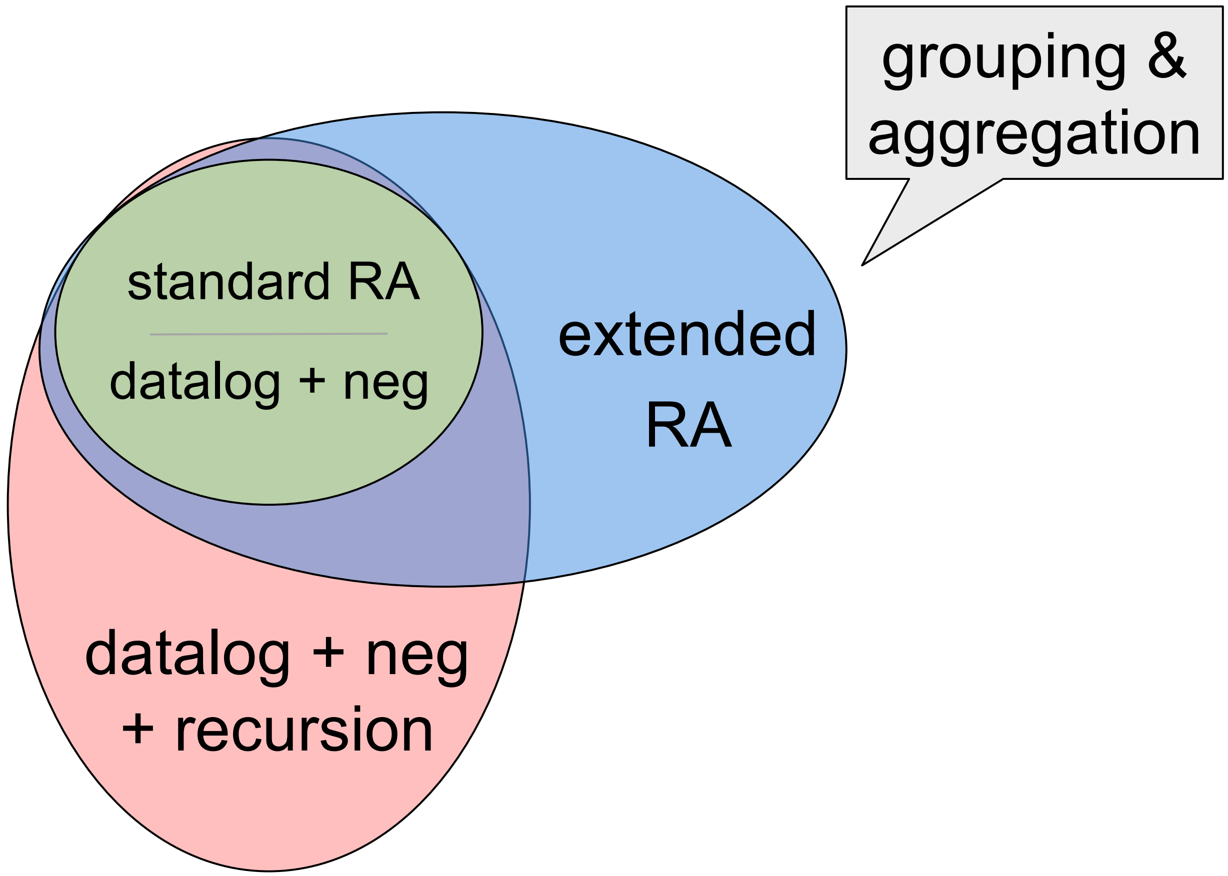
U2(x)   :- Movie(x,z,1994), not Casts(u,x)

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



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

RA to Datalog by Examples

Intersection R(A,B,C) ∩ S(D,E,F)

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

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=‘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: $R \bowtie_{R.A=S.D \text{ and } R.B=S.E} S$

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)

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), 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) :- Friend('Joe', x)
A(x) :- Friend('Joe', z), Friend(z, x)
```

Friend(name1, name2)
Enemy(name1, name2)

More Examples

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

```
JoeFriends(x) :- Friend('Joe',x)
NonAns(x) :- Friend(y,x), y != ‘Joe’
A(x) :- JoeFriends(x), not NonAns(x)
```

Friend(name1, name2)
Enemy(name1, name2)

More Examples

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

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

Everyone(x) :- Friend(x,y)
Everyone(x) :- Friend(y,x)
Everyone(x) :- Enemy(x,y)
Everyone(x) :- Enemy(y,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.

what's wrong with this?

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

```
NonAns(x) :- Friend(x,y), Enemy(y,z), not Enemy(x,z)
A(x) :- Everyone(x), not 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) also allows searching for X=a matches
  - clustered vs non-clustered
    - selectivity above 1-2% => 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 algoirthm
    - those include nested loop, sorted merge, hash, and indexed joins
  - primary goal of the optimizer is to avoid really bad plans