

# Introduction to Data Management

## CSE 344

### Lecture 11: Datalog

# Announcements

- Homework 3 due next Tuesday, 11 pm
  - Don't wait until the last minute – you don't want to be part of the experiment to see if the server bogs down when everyone tries to use it at once (!)

# Datalog

- Book: 5.3, 5.4
- Query Language primer on CSE344 web
- Initially designed for recursive queries
- Today:
  - Some companies use datalog for data analytics, e.g. LogicBlox
  - Popular notation for many CS problems
- We discuss only recursion-free or non-recursive datalog, and add negation

# Datalog

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

- Download DLV from <http://www.dbai.tuwien.ac.at/proj/dlv/>
- Run DLV on this file:

```
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(id, fname, lname)  
Casts(pid, mid)  
Movie(id, 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,z), z='1940'.

Find Movies made in 1940

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, 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,z), z='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(id, fname, lname)

Casts(pid, mid)

Movie(id, 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,z), z='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(id, fname, lname)  
Casts(pid, mid)  
Movie(id, 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,z), z='1940'.

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

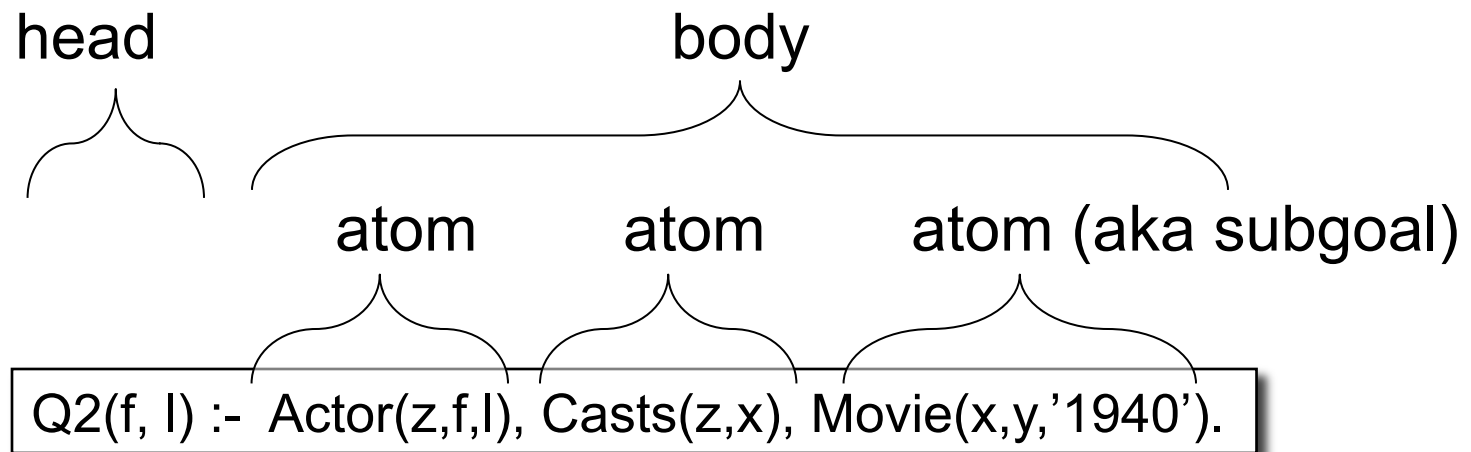
Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),  
Casts(z,x2), Movie(x2,y2,1940)

**Extensional Database Predicates = EDB** = Actor, Casts, Movie

**Intensional Database Predicates = IDB** = Q1, Q2, Q3



# Datalog: Terminology



f, l = head variables

x, y, z = existential variables

# More Datalog Terminology

$Q(\text{args}) \text{ :- } R_1(\text{args}), R_2(\text{args}), \dots$

Book writes:

$Q(\text{args}) \text{ :- } R_1(\text{args}) \text{ AND } R_2(\text{args}) \text{ AND } \dots$

- $R_i(\text{args}_i)$  is called an atom, or a relational predicate
- $R_i(\text{args}_i)$  evaluates to true when relation  $R_i$  contains the tuple described by  $\text{args}_i$ .
  - Example:  $\text{Actor}(344759, \text{'Douglas'}, \text{'Fowley'})$  is true
- In addition to relational predicates, we can also have arithmetic predicates
  - Example:  $z = \text{'1940'}$ .

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

# Semantics

- Meaning of a datalog rule = a logical statement !

$Q1(y) :- \text{Movie}(x,y,z), z='1940'.$

- 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".

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  $\leq 2$

```
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 is the union of B0, B1, and B2

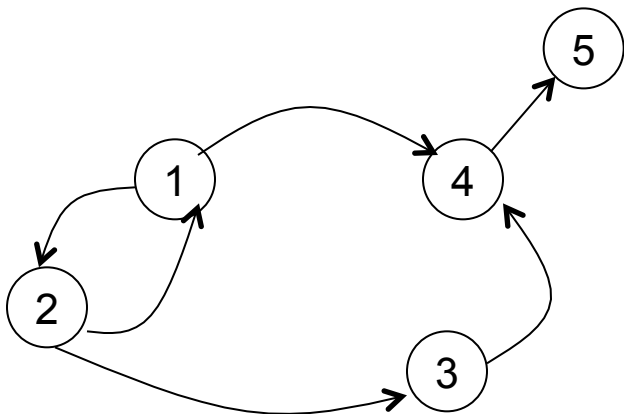
# Non-recursive Datalog

- In datalog, rules can be recursive

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

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

- We focus only 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  $< 2$

$B0(x) :- \text{Actor}(x, \text{'Kevin'}, \text{'Bacon'})$

$B1(x) :- \text{Actor}(x, f, l), \text{Casts}(x, z), \text{Casts}(y, z), B0(y)$

$Q6(x) :- \text{Actor}(x, f, l), \text{not } B1(x), \text{not } B0(x)$

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

# Safe Datalog Rules

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

$U1(x,y) :- \text{Movie}(x,z,1994), y > 1910$

$U2(x) :- \text{Movie}(x,z,1994), \text{not Casts}(u,x)$

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

# Datalog v.s. Relational Algebra

- Every expression in the basic 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

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

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

# RA to Datalog by Examples

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

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

Selection  $x>100$  **or**  $y=\text{'some string'}$

$L(x,y,z) :- R(x,y,z), x > 100$

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

# RA to Datalog by Examples

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

$J(x,y,z,q) :- R(x,y,z), S(x,y,q)$

# RA to Datalog by Examples

Projection

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

# RA to Datalog by Examples

To express difference, 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:  $\Pi_A(\sigma_{B=3}(R))$



# 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:  $\Pi_A(\sigma_{B=3}(R) \bowtie_{R.A=S.D} \sigma_{E=5}(S))$

# 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.

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:

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) :- JoeFriends(x), Friend(x,y), 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



Friend(name1, name2)

Enemy(name1, name2)

## More Examples

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?

Friend(name1, name2)

Enemy(name1, name2)

## More Examples

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!

Friend(name1, name2)

Enemy(name1, name2)

## More Examples

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!

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 a friend all of whose enemies are x's enemies.

Friend(name1, name2)

Enemy(name1, name2)

## More Examples

Find all persons  $x$  that have a friend all of whose enemies are  $x$ 's enemies.

Everyone( $x$ ) :- Friend( $x,y$ )

NonAns( $x$ ) :- Friend( $x,y$ ) Enemy( $y,z$ ), NOT Enemy( $x,z$ )

A( $x$ ) :- Everyone( $x$ ), NOT NonAns( $x$ )

# Why Do We Learn Datalog?

- A query language that is closest to mathematical logic
- Datalog can be translated to SQL (practice at home !)
- Can also translate back and forth between datalog and relational algebra
- Bottom line: relational algebra, non-recursive datalog with negation, and relational calculus all have the same expressive power!

# Why Do We Learn Datalog?

Datalog, Relational Algebra, and Relational Calculus (next lecture) are of fundamental importance in DBMSs because

1. Sufficiently expressive to be useful in practice yet
2. Sufficiently simple to be efficiently implementable