CSE 344

JANUARY 29TH – DATALOG

ADMINISTRATIVE MINUTIAE

- **HW3 due Friday**
- **OQ due Wednesday**
- **HW4 out Wednesday**
- **Exam next Friday**
  - 3:30 - 5:00

WHAT IS DATALOG?

Another query language for relational model

- Designed in the 80’s
- Simple, concise, elegant
- Extends relational queries with *recursion*

Relies on a logical framework for ”record” selection

DATALOG: FACTS AND RULES

Facts = tuples in the database

Rules = queries

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

Schema

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).
```

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)
```

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,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

head

body

atom

atom

atom (aka subgoal)

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

f, l   = head variables
x,y,z= existential variables

MORE DATALOG TERMINOLOGY

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

R_i($args_i$) 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 we can also have arithmetic predicates

- Example: z > ‘1940’.

Book uses AND instead of ,

```
Q(args) :- R1(args) AND R2(args) ....
```

SEMANTICS OF A SINGLE RULE

Meaning of a datalog rule = a logical statement !

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

- For all x, y, z: if (x,y,z) $\in$ Movies and z = '1940' then y is in Q1 (i.e. is part of the answer)
- $\forall x \forall y \forall z$ [(Movie(x,y,z) and z='1940') $\Rightarrow$ Q1(y)]
- Logically equivalent:
  $\forall y$ [($\exists x \exists z$ Movie(x,y,z) and z='1940') $\Rightarrow$ Q1(y)]
- Thus, non-head variables are called "existential variables"
- We want the *smallest* set Q1 with this property (why?)

DATALOG PROGRAM

A datalog program consists of several rules

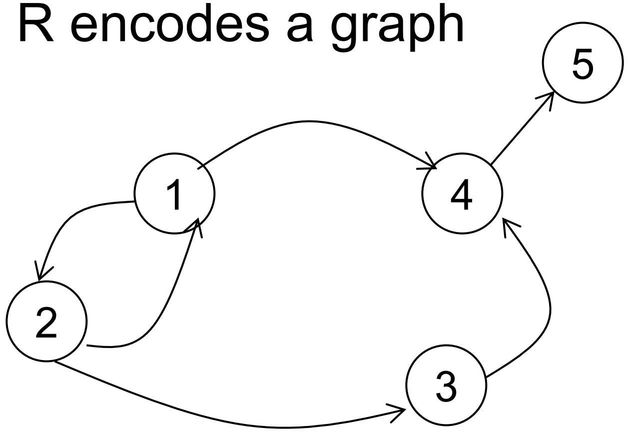
Importantly, rules may be recursive!

Usually there is one distinguished predicate that's the output

We will show an example first, then give the general semantics.

EXAMPLE

R encodes a graph

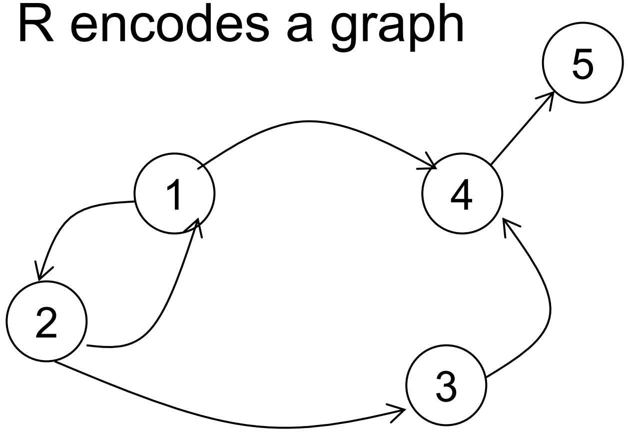


R=

1	2
2	1
2	3
1	4
3	4
4	5

EXAMPLE

R encodes a graph



R=

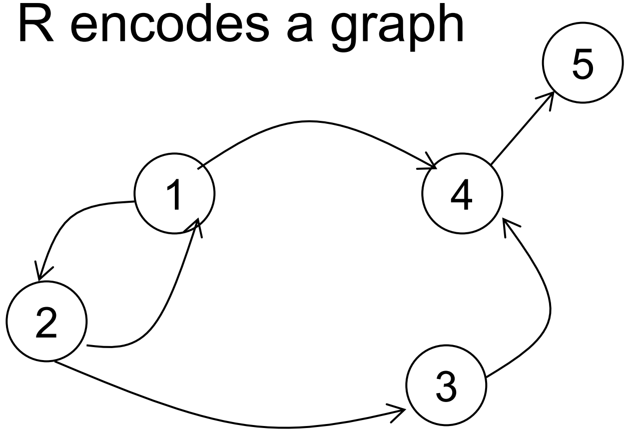
1	2
2	1
2	3
1	4
3	4
4	5

T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)

What does it compute?

EXAMPLE

R encodes a graph



T(x,y) :- R(x,y)	
T(x,y) :- R(x,z), T(z,y)	

What does it compute?

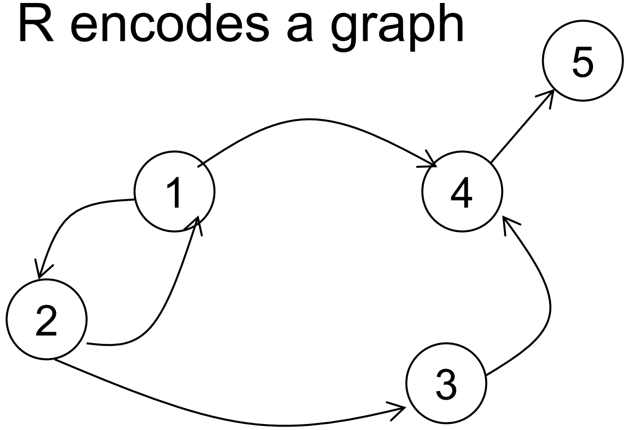
R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:
T is empty.

EXAMPLE

R encodes a graph



T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:
T is empty.

First iteration:
T =

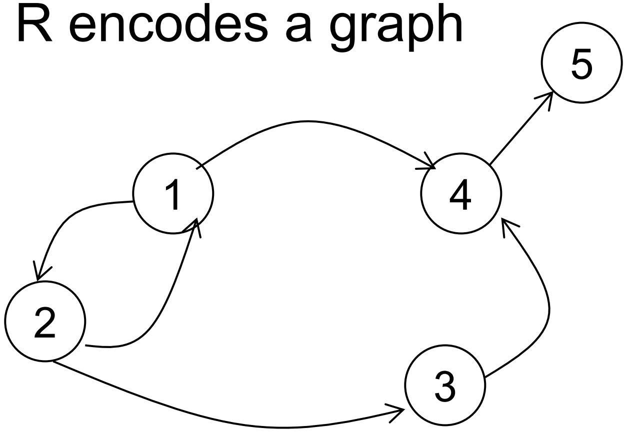
1	2
2	1
2	3
1	4
3	4
4	5

First rule generates this

Second rule generates nothing (because T is empty)

EXAMPLE

R encodes a graph



$$T(x,y) \text{ :- } R(x,y)$$
$$T(x,y) \text{ :- } R(x,z),\ T(z,y)$$

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:
T is empty.

First iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5

Second iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

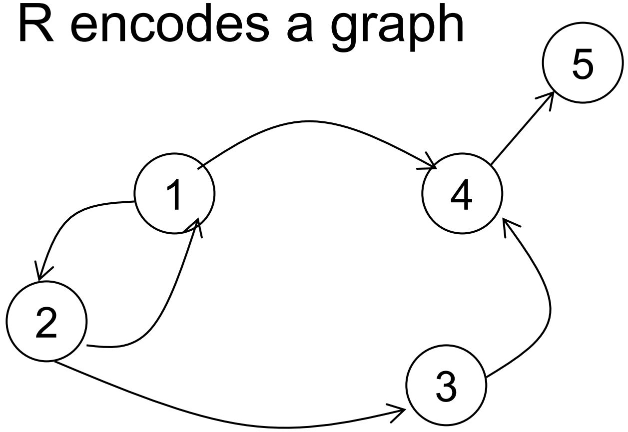
First rule generates this (rows 1 2 through 4 5)

Second rule generates this (rows 1 1 through 3 5)

New facts

EXAMPLE

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:
T is empty.

$$T(x,y) \text{ :- } R(x,y)$$
$$T(x,y) \text{ :- } R(x,z), T(z,y)$$

What does it compute?

First iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5

Second iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

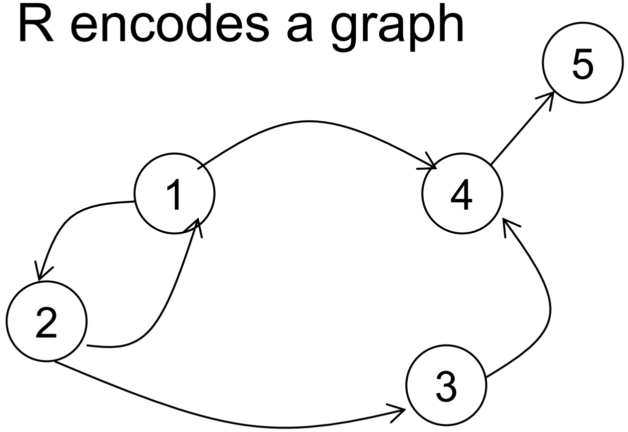
Third iteration:
T =

1	2	Both rule
2	1	
2	3	First rule
1	4	
3	4	
4	5	
1	1	Second rule
2	2	
1	3	
2	4	
1	5	
3	5	
2	5	

New fact

EXAMPLE

R encodes a graph



T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:
T is empty.

First iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5

Second iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

Third iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5
2	5

Fourth
iteratio
T =
(same)

No new facts. DONE

DATALOG SEMANTICS

Fixpoint semantics

Start:

IDB_0 = empty relations

t = 0

Repeat:

IDB_{t+1} = Compute Rules(EDB, IDB_t)

t = t+1

Until $IDB_t = IDB_{t-1}$

Remark: since rules are monotone:

$\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots$

It follows that a datalog program w/o functions (+, *, ...) always terminates. (Why? In what time?)

DATALOG SEMANTICS

Minimal model semantics:

Return the IDB that

1) For every rule,
   $\forall$ vars [(Body(EDB,IDB) $\Rightarrow$ Head(IDB)]
2) Is the smallest IDB satisfying (1)

Theorem: there exists a smallest IDB satisfying (1)

DATALOG SEMANTICS

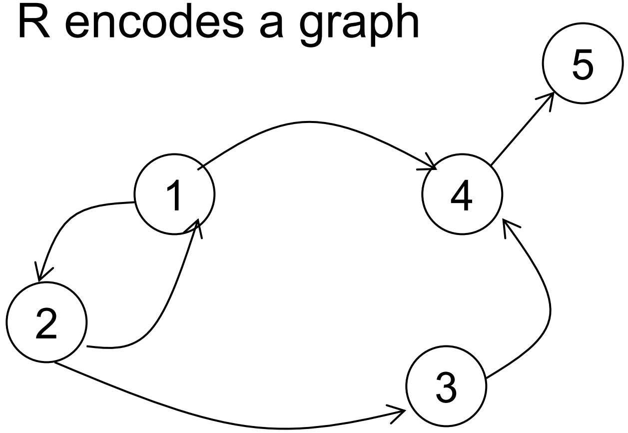
The fixpoint semantics tells us how to compute a datalog query

The minimal model semantics is more declarative: only says what we get

The two semantics are equivalent meaning: you get the same thing

THREE EQUIVALENT PROGRAMS

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

Right linear

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

Left linear

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
```

Non-linear

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), T(z,y)
```

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)
```

ParentChild(p,c)

SAFE DATALOG RULES

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

```
U1(x,y) :- ParentChild("Alice",x), y != "Bob"
```

Holds for
every y other than "Bob"
U1 = infinite!

```
U2(x)   :- ParentChild("Alice",x), !ParentChild(x,y)
```

SAFE DATALOG RULES

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

U1(x,y) :- ParentChild("Alice",x), y != "Bob"

Holds for
every y other than "Bob"
U1 = infinite!

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)

ParentChild(p,c)

SAFE DATALOG RULES

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

```
U1(x,y) :- ParentChild("Alice",x), y != "Bob"
```

Holds for
every y other than "Bob"
U1 = infinite!

```
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)

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

DATALOG: RELATIONAL DATABASE

- **Datalog can express things RA cannot**
  - Recursive Queries
- **Can Datalog express all queries in RA?**

DATALOG: RELATIONAL DATABASE

- **Datalog can express things RA cannot**
  - Recursive Queries
- **Can Datalog express all queries in RA?**

RELATIONAL ALGEBRA OPERATORS

Union $\cup$, difference **-**

Selection σ

Projection π

Cartesian product X, join ⋈

OPERATORS IN DATALOG

- Suppose we want Q1(…) to contain all the values from F1(…) and F2(…)

OPERATORS IN DATALOG

- **Suppose we want Q1(…) to contain all the values from F1(…) and F2(…)**
  - Q1(…) :- F1(…)
  - Q1(…) :- F2(…)
- **What about for difference?**

OPERATORS IN DATALOG

- **Suppose we want Q1(…) to contain all the values from F1(…) and F2(…)**
  - Q1(…) :- F1(…)
  - Q1(…) :- F2(…)
- **What about for difference?**
  - Q1(…) :- F1(…), !F2(…)

OPERATORS IN DATALOG

- **Suppose we want Q1(…) to contain all the values from F1(…) and F2(…)**
  - Q1(…) :- F1(…)
  - Q1(…) :- F2(…)
- **What about for difference?**
  - Q1(…) :- F1(…), !F2(…)
  - The variables (…) in F1 and F2 must be the same, or else we have an *unsafe* rule

OPERATORS IN DATALOG

- **Projection, from the variables R_1,R_2,...R_k select some subset of the variables**

OPERATORS IN DATALOG

- **Projection, from the variables R_1,R_2,...R_k select some subset of the variables**
  - Q1(subset) :- Original(all_attributes)
- **Selection: only return certain records from our knowledge base**

OPERATORS IN DATALOG

- **Projection, from the variables R_1,R_2,...R_k select some subset of the variables**
  - Q1(subset) :- Original(all_attributes)
- **Selection: only return certain records from our knowledge base**
  - Q1(…) :- Original(…), selection_criteria

OPERATORS IN DATALOG

- Cross product: find all the pairs between $R(a_1,a_2\ldots)$ and $S(b_1,b_2\ldots)$

OPERATORS IN DATALOG

- **Cross product: find all the pairs between $R(a_1,a_2\ldots)$ and $S(b_1,b_2\ldots)$**
  - $Q1(a_1,b_1,a_2,b_2\ldots)$:- $R(a_1,a_2\ldots)$, $S(b_1,b_2\ldots)$
- **Joins?**
  - Natural

OPERATORS IN DATALOG

- **Cross product: find all the pairs between $R(a_1,a_2\ldots)$ and $S(b_1,b_2\ldots)$**
  - $Q1(a_1,b_1,a_2,b_2\ldots)$:- $R(a_1,a_2\ldots)$, $S(b_1,b_2\ldots)$
- **Joins?**
  - Natural: Q1(a,b,c) :- R(a,b), S(b,c)
  - Theta

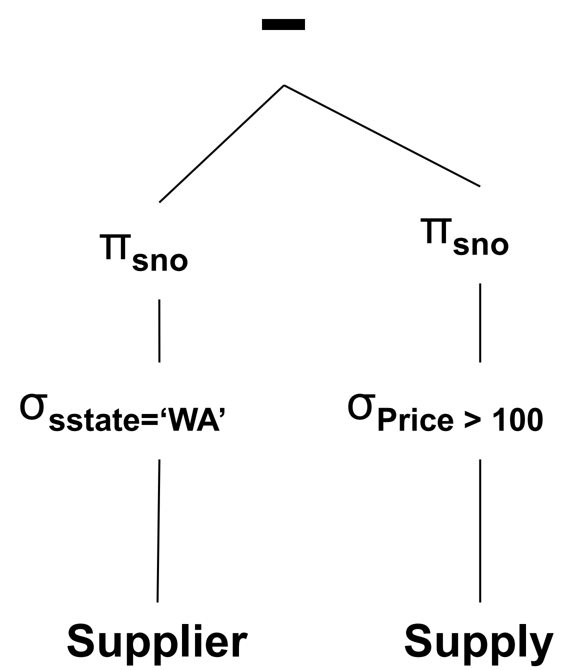
OPERATORS IN DATALOG

- **Cross product: find all the pairs between $R(a_1,a_2\ldots)$ and $S(b_1,b_2\ldots)$**
  - $Q1(a_1,b_1,a_2,b_2\ldots)$:- $R(a_1,a_2\ldots)$, $S(b_1,b_2\ldots)$
- **Joins?**
  - Natural: Q1(a,b,c) :- R(a,b), S(b,c)
  - Theta: Cross product with selection
  - Equijoin: subset of Theta join

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

EXAMPLE

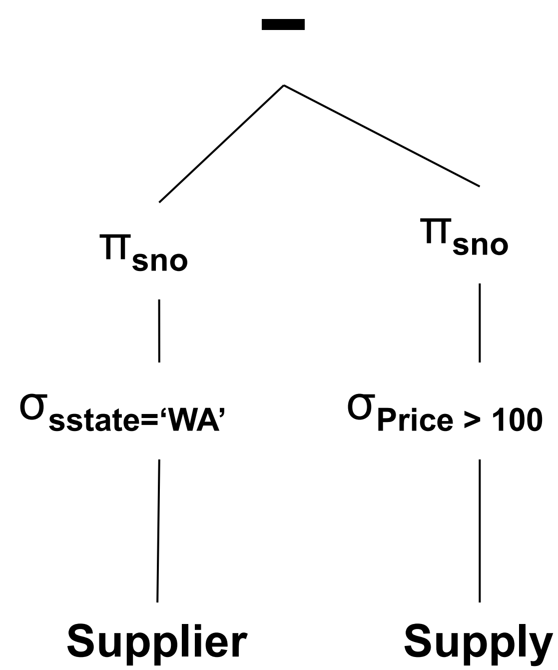
```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = ‘WA’)
    EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```



EXAMPLE

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

```
Datalog:
```

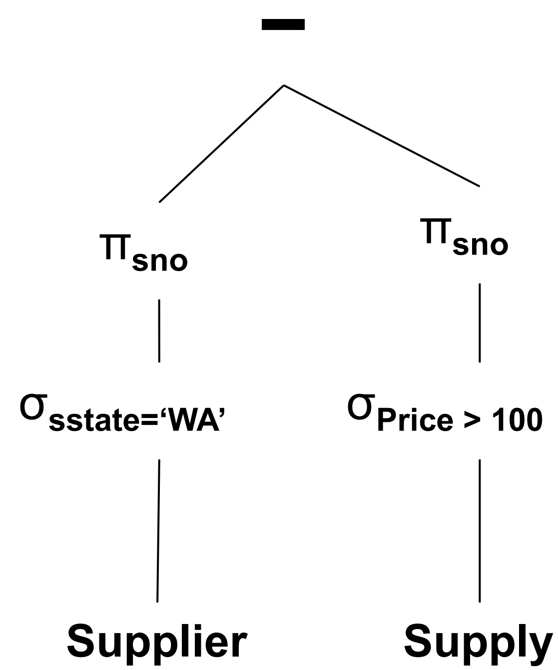


EXAMPLE

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

```
Datalog:

Q1(no,name,city,state) :-
    Supplier(sno,sname,
    scity,sstate),
    sstate='WA'
Q2(no,pno,price) :-
    Supply(s,pn,pr),
    pr > 100
Q3(sno) :- Q1(sno,n,c,s)
Q4(sno) :- Q2(sno,pn,pr)
Result(sno) :- Q1(sno),
    !Q2(sno)
```



MORE EXAMPLES W/O RECURSION

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

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

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

MORE EXAMPLES W/O RECURSION

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)
```

MORE EXAMPLES W/O RECURSION

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)
```

MORE EXAMPLES W/O RECURSION

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)
```

ParentChild(p,c)

MORE EXAMPLES W/ RECURSION

Two people are in the *same generation* if they are siblings, or if they have parents in the same generation

Find all persons in the same generation with Alice

ParentChild(p,c)

MORE EXAMPLES W/ RECURSION

Find all persons in the same generation with Alice

Let's compute SG(x,y) = "x,y are in the same generation"

```
SG(x,y) :- ParentChild(p,x), ParentChild(p,y)
SG(x,y) :- ParentChild(p,x), ParentChild(q,y), SG(p,q)
Answer(x) :- SG("Alice", x)
```

DATALOG SUMMARY

EDB (base relations) and IDB (derived relations)

Datalog program = set of rules

Datalog is recursive

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