CSE 344

APRIL 9TH – DATALOG

ADMINISTRATIVE MINUTIAE

- **Midterm exam**
  - Piazza poll
- **OQ 2/3 Due Friday**
- **HW2 Due Wednesday**
- **HW3 Out Wednesday**

RELATIONAL ALGEBRA

Set-at-a-time algebra, which manipulates relations

In SQL we say ***what*** we want

In RA we can express ***how*** to get it

Every DBMS implementations converts a SQL query to RA in order to execute it

An RA expression is called a ***query plan***

BASICS

- Relations and attributes
- Functions that are applied to relations
  - Return relations
  - Can be composed together
  - Often displayed using a tree rather than linearly
  - Use Greek symbols: σ, π, δ, etc

JOIN SUMMARY

Theta-join: $R \bowtie_\theta S = \sigma_\theta (R \times S)$

- Join of R and S with a join condition θ
- Cross-product followed by selection θ
- No projection

Equijoin: $R \bowtie_\theta S = \sigma_\theta (R \times S)$

- Join condition θ consists only of equalities
- No projection

Natural join: $R \bowtie S = \pi_A (\sigma_\theta (R \times S))$

- Equality on **all** fields with same name in R and in S
- Projection π_A drops all redundant attributes

MORE JOINS

Outer join

- Include tuples with no matches in the output
- Use NULL values for missing attributes
- Does not eliminate duplicate columns

Variants

- Left outer join
- Right outer join
- Full outer join

SOME EXAMPLES

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

Name of supplier of parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10}(\text{Part}))$

Name of supplier of red parts or parts with size greater than 10

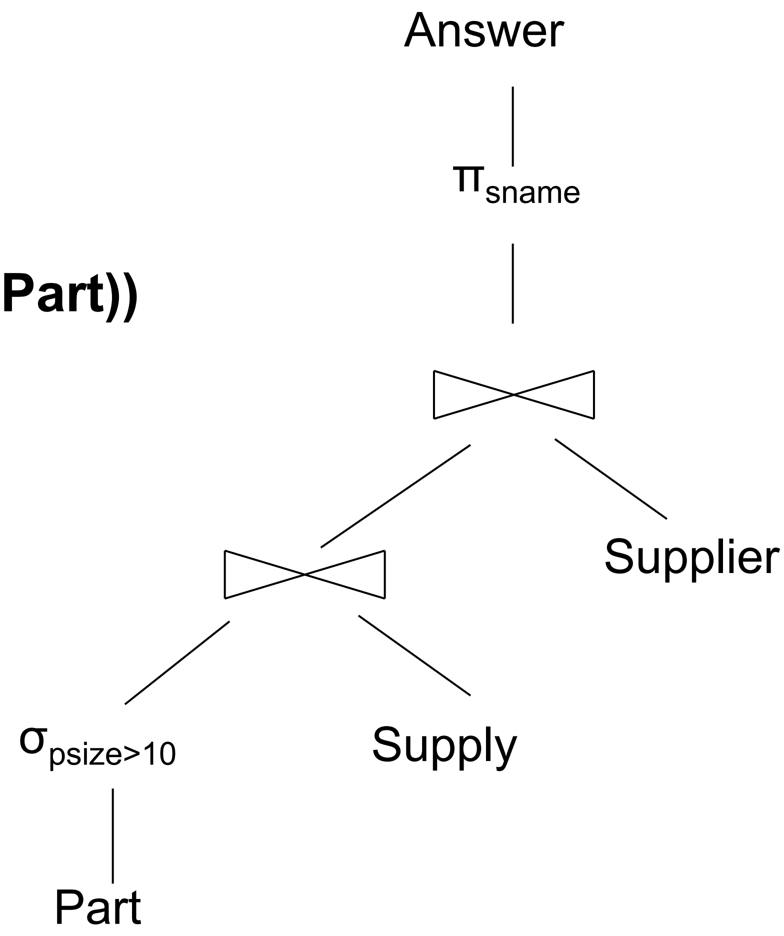
$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10}(\text{Part}) \cup \sigma_{pcolor='red'}(\text{Part})))$

$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10 \vee pcolor='red'}(\text{Part})))$

Can be represented as trees as well

REPRESENTING RA QUERIES AS TREES

$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10} (\text{Part}))$



RELATIONAL ALGEBRA OPERATORS

Union ∪ , ~~intersection ∩,~~ difference -

Selection σ

Projection π

Cartesian product X, join ⋈

(Rename ρ)

Duplicate elimination δ

Grouping and aggregation ɣ

Sorting τ

RA

Extended RA

All operators take in 1 or more relations as inputs and return another relation

EXTENDED RA: OPERATORS ON BAGS

Duplicate elimination δ

Grouping γ

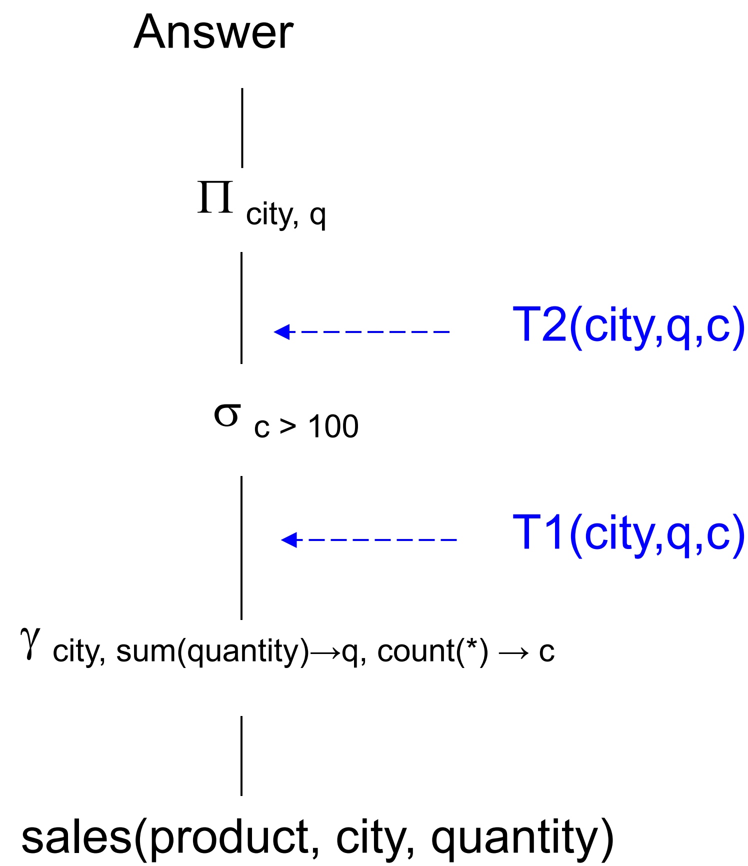
- Takes in relation and a list of grouping operations (e.g., aggregates). Returns a new relation.

Sorting τ

- Takes in a relation, a list of attributes to sort on, and an order. Returns a new relation.

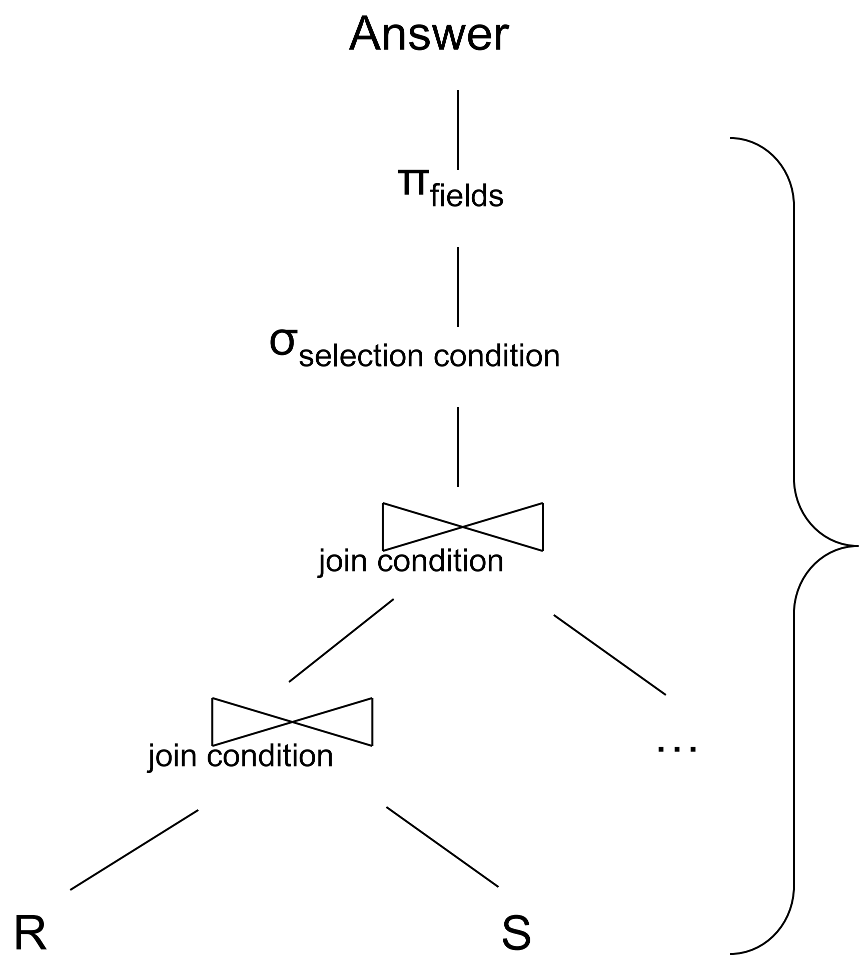
USING EXTENDED RA OPERATORS

```sql
SELECT city, sum(quantity)
FROM sales
GROUP BY city
HAVING count(*) > 100
```



T1, T2 = temporary tables

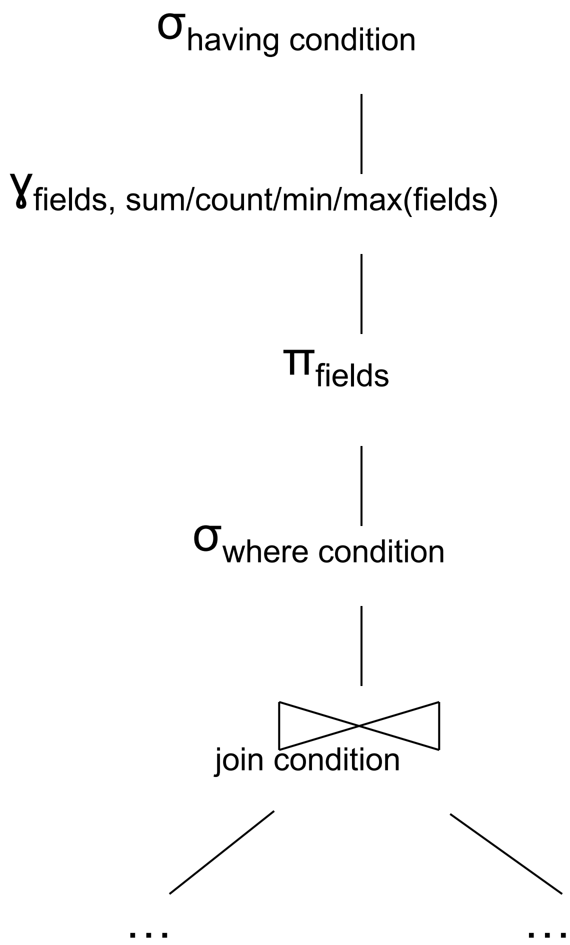
TYPICAL PLAN FOR A QUERY (1/2)




```
SELECT fields
FROM R, S, …
WHERE condition
```

SELECT-PROJECT-JOIN Query

TYPICAL PLAN FOR A QUERY (1/2)




```
SELECT fields
FROM R, S, …
WHERE condition
GROUP BY fields
HAVING condition
```

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

HOW ABOUT SUBQUERIES?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and not exists
  (SELECT *
   FROM Supply P
   WHERE P.sno = Q.sno
         and P.price > 100)
```

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

HOW ABOUT SUBQUERIES?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and not exists
  (SELECT *
   FROM Supply P
   WHERE P.sno = Q.sno
         and P.price > 100)
```

Correlation !

HOW ABOUT SUBQUERIES?

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

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and not exists
  (SELECT *
   FROM Supply P
   WHERE P.sno = Q.sno
         and P.price > 100)
```

De-Correlation

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
  and Q.sno not in
  (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

HOW ABOUT SUBQUERIES?

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

Un-nesting

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

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
  and Q.sno not in
  (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

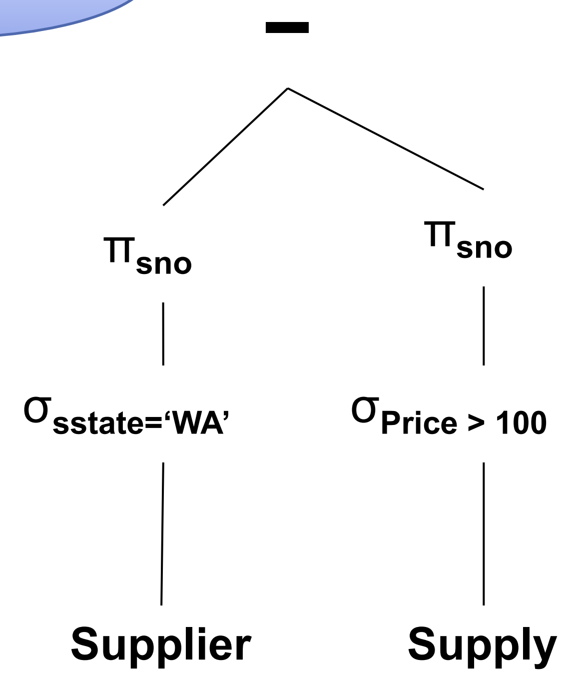
EXCEPT = set difference

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

HOW ABOUT SUBQUERIES?

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

Finally…



SUMMARY OF RA AND SQL

SQL = a declarative language where we say ***what*** data we want to retrieve

RA = an algebra where we say ***how*** we want to retrieve the data

Theorem: SQL and RA can express exactly the same class of queries

RDBMS translate SQL → RA, then optimize RA

RELATIONAL ALGEBRA TAKEAWAYS

- Be able to get a query write the relational algebra expression equivalent to it
- Given a relational algebra expression, write the equivalent query
- Understand what each are trying to get semantically

SUMMARY OF RA AND SQL

SQL (and RA) cannot express ALL queries that we could write in, say, Java

Example:

- Parent(p,c): find all descendants of 'Alice'
- No RA query can compute this!
- This is called a *recursive query*

Datalog **is an extension that can compute recursive queries**

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'.
```

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

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').

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').
```

Find Actors who acted in Movies made in 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)
```

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)
```

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

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').
```

- head: Q2(f, l)
- body: Actor(z,f,l), Casts(z,x), Movie(x,y,'1940')
- atom: Actor(z,f,l)
- atom: Casts(z,x)
- atom (aka subgoal): 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

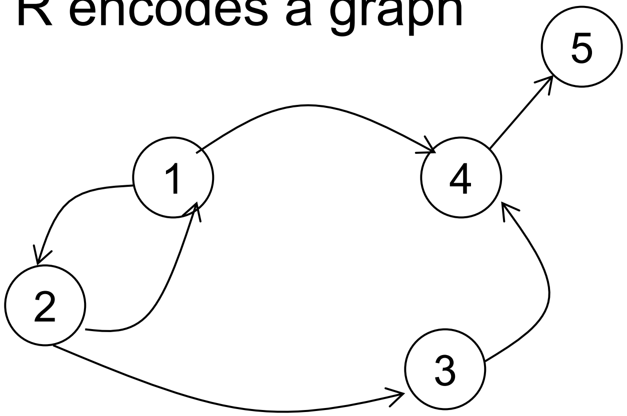
Importantly, rules may be recursive!

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

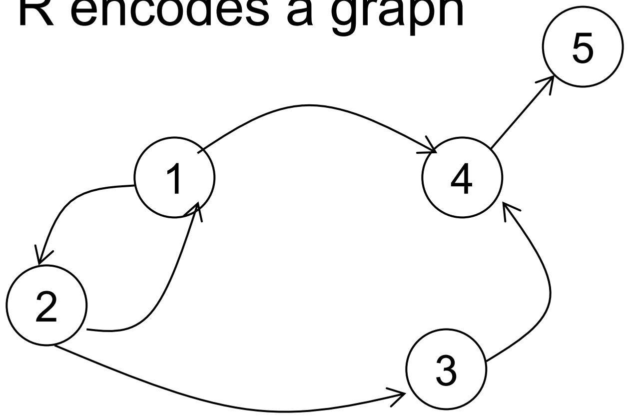


R=

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

EXAMPLE

R encodes a graph



R=

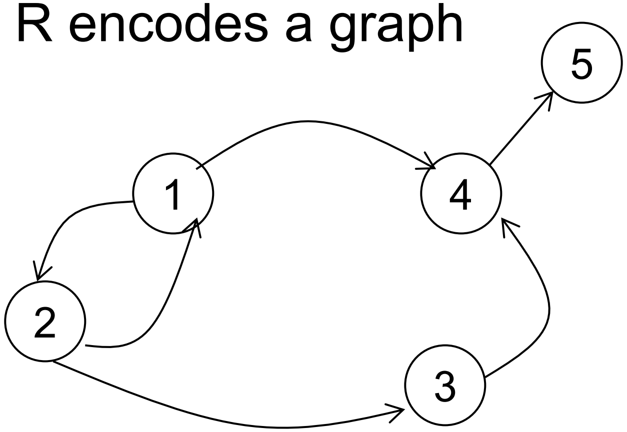
1	2
2	1
2	3
1	4
3	4
4	5

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

What does it compute?

EXAMPLE

R encodes a graph



R=

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

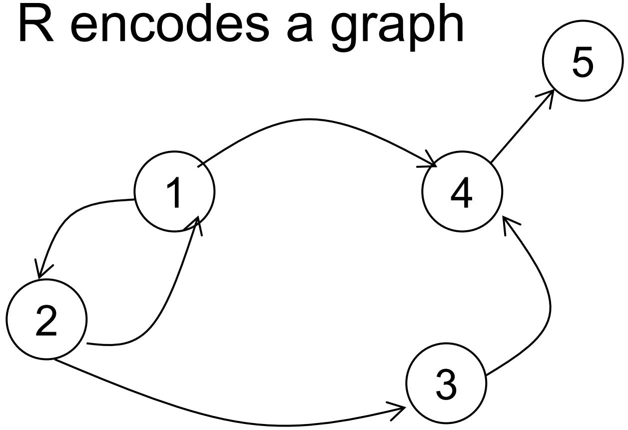
Initially:
T is empty.

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.

First iteration:
T =

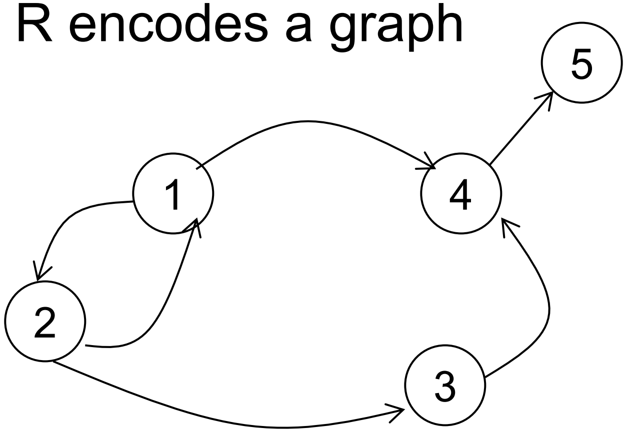
1	2
2	1
2	3
1	4
3	4
4	5

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

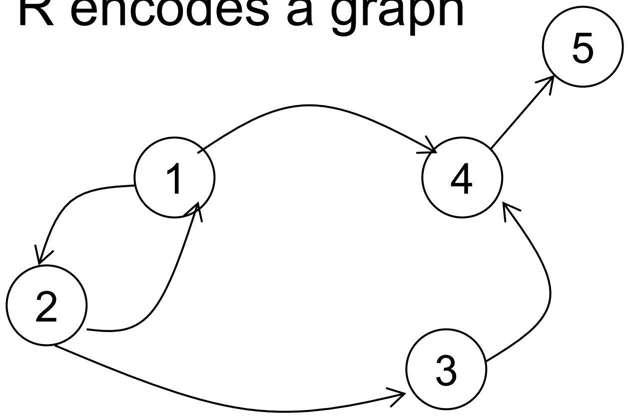
First rule generates this (rows 1 2 through 4 5)

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

New facts

EXAMPLE

R encodes a graph



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

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 =

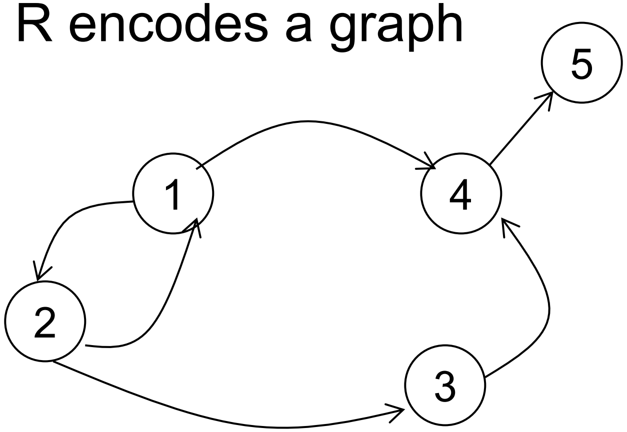
1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

Third iteration:
T =

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

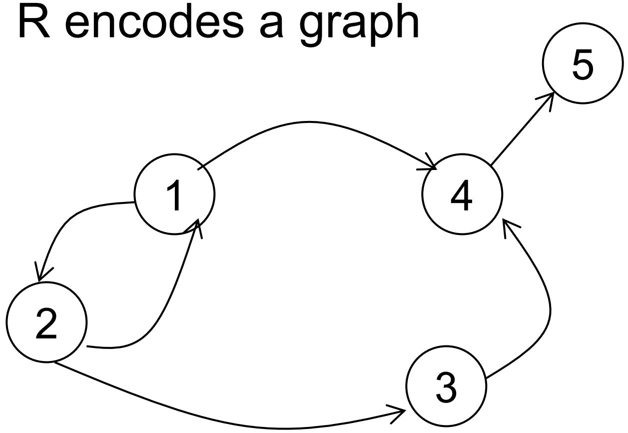
The fixpoint semantics tells us how to compute a datalog query

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)
```

ParentChild(p,c)

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)

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)

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