CSE544 Data Management

Lectures 15

Datalog

Agenda

- Finish the discussion of datalog
- Brief review of what this class was about

Monotone Queries

- A set function $F(R_1, R_2, \ldots)$ is monotone if

$$R_1 \subseteq R_1', R_2 \subseteq R_2', \ldots \Rightarrow F(R_1, R_2, \ldots) \subseteq F(R_1', \ldots)$$

Monotone Queries

- A set function $F(R_1, R_2, \dots)$ is monotone if

$$R_1 \subseteq R_1', R_2 \subseteq R_2', \dots \Rightarrow F(R_1, R_2, \dots) \subseteq F(R_1', \dots)$$

- Set difference is not monotone:

$$R_1 - R_2 \not\subseteq R_1 - R_2'$$

Monotone Queries

- A set function $F(R_1, R_2, \ldots)$ is monotone if

$$R_1 \subseteq R_1', R_2 \subseteq R_2', \ldots \Rightarrow F(R_1, R_2, \ldots) \subseteq F(R_1', \ldots)$$

- Set difference is not monotone:

$$R_1 - R_2 \nsubseteq R_1 - R_2'$$

- Aggregates are not montone:

$$\{1+2\} \nsubseteq \{1+2+3\}$$

Non-Monotone Features

- Negation
- Aggregates/group-by

Three Useful Queries w/ Negation

Transitive closure of the complement

$$NR(x,y) :- V(x), V(y), \neg R(x,y)$$
$$T(x,y) :- NR(x,y)$$
$$T(x,y) :- NR(x,z), T(z,y)$$

Three Useful Queries w/ Negation

Transitive closure of the complement

$$NR(x,y) :- V(x), V(y), \neg R(x,y)$$
$$T(x,y) :- NR(x,y)$$
$$T(x,y) :- NR(x,z), T(z,y)$$

Complement of the transitive closure

$$T(x,y) :- R(x,y)$$
$$T(x,y) :- R(x,z), T(z,y)$$
$$Answ(x,y) :- V(x), V(y), \neg T(x,y)$$

Three Useful Queries w/ Negation

Transitive closure of the complement

$$\mathrm{NR}(x,y) :- V(x), V(y), \neg R(x,y)$$
$$T(x,y) :- \mathrm{NR}(x,y)$$
$$T(x,y) :- \mathrm{NR}(x,z), T(z,y)$$

Complement of the transitive closure

$$T(x,y) :- R(x,y)$$
$$T(x,y) :- R(x,z), T(z,y)$$
$$\mathrm{Answ}(x,y) :- V(x), V(y), \neg T(x,y)$$

The Win-Move game (won't discuss in class)

$$W(x) :- R(x,y), \neg W(y)$$

Recursion+Negation Don't Mix

- The next example is super-simple, but recall a simple fact:

- A relation A of arity 0 is a Boolean variable:
  - $\mathrm{A} = \emptyset$ or $\mathrm{A} = \{(\,)\}$,
  - I.e. A is either FALSE or TRUE

Recursion+Negation Don't Mix

$$B() :- \neg A()$$
$$A() :- \neg B()$$

What are
the models?

Recursion+Negation Don't Mix

$$B() :- \neg A()$$
$$A() :- \neg B()$$

What are the models?

A=False, B=True

Recursion+Negation Don't Mix

$$B(\,):-\ \neg A(\,)$$
$$A(\,):-\ \neg B(\,)$$

What are the models?

A=False, B=True

A=True, B=False

Recursion+Negation Don’t Mix

$$B() :- \neg A()$$
$$A() :- \neg B()$$

What are the models?

A=False, B=True

A=True, B=False

A=True, B=True

Recursion+Negation Don't Mix

$$B() :- \neg A()$$
$$A() :- \neg B()$$

What are the models?

A=False, B=True

A=True, B=False

A=True, B=True

No minimal model

Recursion+Negation Don't Mix

$$B():-\ \neg A()$$
$$A():-\ \neg B()$$

What are the models?

A=False, B=True

A=True, B=False

A=True, B=True

No minimal model

What are the fixpoints?

Recursion+Negation Don't Mix

$$B():-\ \neg A()$$
$$A():-\ \neg B()$$

What are the models?

A=False, B=True

A=True, B=False

A=True, B=True

No minimal model

What are the fixpoints?

(False,True), (True, False)

No least fixpoint

Recursion+Negation Don't Mix

$$B():-\ \neg A()$$
$$A():-\ \neg B()$$

What are the models?

A=False, B=True

A=True, B=False

A=True, B=True

No minimal model

What are the fixpoints?

(False,True), (True, False)

No least fixpoint

What does the naïve algorithm compute?

Recursion+Negation Don't Mix

$$B(\,):-\ \neg A(\,)$$
$$A(\,):-\ \neg B(\,)$$

What are the models?

A=False, B=True

A=True, B=False

A=True, B=True

No minimal model

What are the fixpoints?

(False,True), (True, False)

No least fixpoint

What does the naïve algorithm compute?

$(\mathrm{A}_0, \mathrm{B}_0) = (0,0);$

Recursion+Negation Don't Mix

$$B():-\ \neg A()$$
$$A():-\ \neg B()$$

What are the models?

A=False, B=True

A=True, B=False

A=True, B=True

No minimal model

What are the fixpoints?

(False,True), (True, False)

No least fixpoint

What does the naïve algorithm compute?

$(A_0, B_0) = (0,0);\ (A_1, B_1) = (1,1);$

Recursion+Negation Don't Mix

$$B():-\ \neg A()$$
$$A():-\ \neg B()$$

What are the models?

A=False, B=True

A=True, B=False

A=True, B=True

No minimal model

What are the fixpoints?

(False,True), (True, False)

No least fixpoint

What does the naïve algorithm compute?

$$(A_0, B_0) = (0,0);\ (A_1, B_1) = (1,1);\ (A_2, B_2) = (0,0); \cdots$$

Recursion+Negation Don’t Mix

$$B():-\ \neg A()$$
$$A():-\ \neg B()$$

What are the models?

A=False, B=True

A=True, B=False

A=True, B=True

No minimal model

What are the fixpoints?

(False,True), (True, False)

No least fixpoint

What does the naïve algorithm compute?

$$(A_0, B_0) = (0{,}0);\ (A_1, B_1) = (1{,}1);\ (A_2, B_2) = (0{,}0); \cdots$$

Does not converge

Approaches to Negation

- Semi-positive datalog
- Stratified datalog
- Sophisticated model-theoretic definitions: stable models, well founded models. Will not discuss.

Semi-positive Datalog

- EDB atoms may be positive or negated
- IDB atoms are positive
- ICO is monotone.
- Semantics: least fixpoint of ICO

Semi-positive Datalog

- E.g. transitive closure of complement

$$NR(x, y) :- V(x), V(y), \neg R(x, y)$$
$$T(x, y) :- NR(x, y)$$
$$T(x, y) :- NR(x, z), T(z, y)$$

Stratified Datalog

Intuition:

- Assign IDBs to strata 1, 2, 3, …
- IDBs computed in stratum s, may use non-monotone occurrences of IDBs at strata < s

Stratified Datalog

Formally: assign a stratum $s(\mathrm{R}) \in \mathbb{N}$ to each IDB predicate R

The program is stratified if there exists a stratification such that:

Stratified Datalog

Formally: assign a stratum $s(\mathrm{R}) \in \mathbb{N}$ to each IDB predicate R

The program is stratified if there exists a stratification such that:

- Positive atoms: $\mathrm{A(X)} :- \cdots \mathrm{B(Y)} \cdots$ $\quad s(\mathrm{A}) \geq s(\mathrm{B})$

Stratified Datalog

Formally: assign a stratum $s(\mathrm{R}) \in \mathbb{N}$ to each IDB predicate R

The program is stratified if there exists a stratification such that:

- Positive atoms: $\mathrm{A(X)} :- \cdots \mathrm{B(Y)} \cdots$ $\quad s(\mathrm{A}) \geq s(\mathrm{B})$
- Negative atoms: $\mathrm{A(X)} :- \cdots \neg \mathrm{B(Y)} \cdots$ $\quad s(\mathrm{A}) > s(\mathrm{B})$

Stratified Datalog

Formally: assign a stratum $s(R) \in \mathbb{N}$ to each IDB predicate R

The program is stratified if there exists a stratification such that:

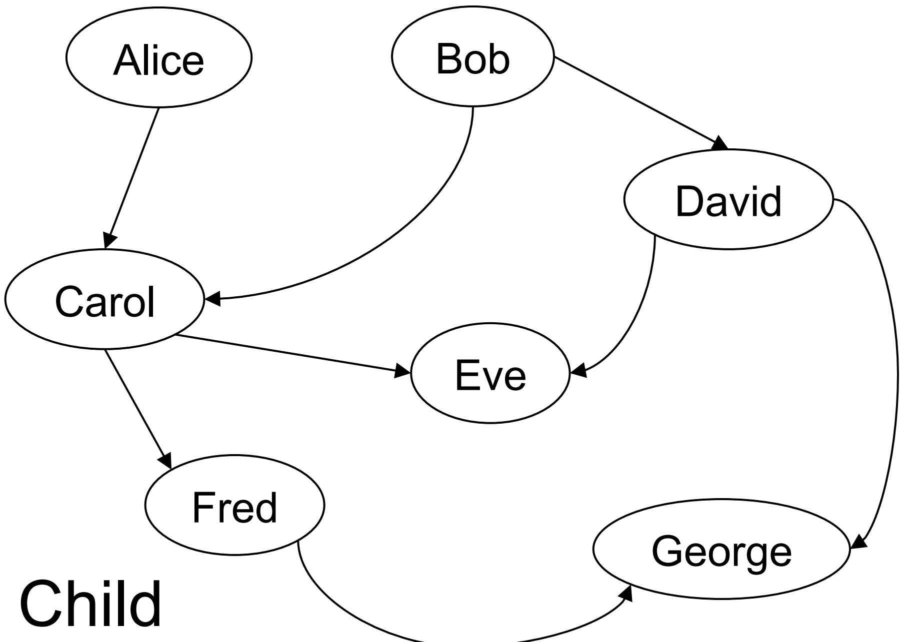
- Positive atoms: $A(X) :- \cdots B(Y) \cdots$ $\quad s(A) \geq s(B)$
- Negative atoms: $A(X) :- \cdots \neg B(Y) \cdots$ $\quad s(A) > s(B)$
- Aggregates: $A(agg(\cdots)) :- body$ $\quad \forall B \in body: \ s(A) > s(B)$

Negation, Aggregates in Souffle

- Negation: !

- Aggregates: complicated syntax, will show by examples

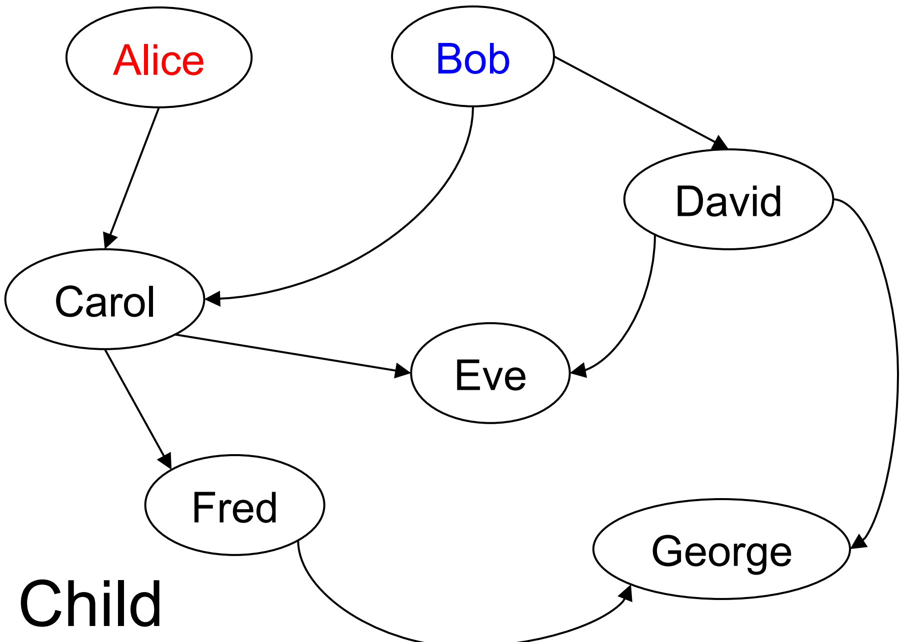
Negation in Souffle



Child

p	c
Alice	Carol
Bob	Carol
Bob	David
Carol	Eve
…	

Negation in Souffle

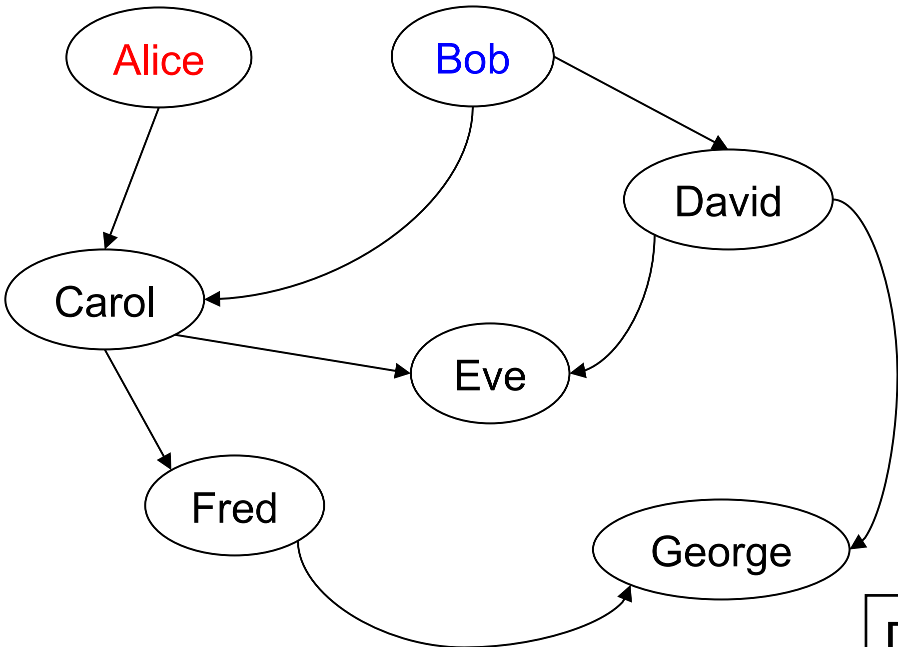


Find all descendants of Bob
that are not descendants of Alice

Child

p	c
Alice	Carol
Bob	Carol
Bob	David
Carol	Eve
…	

Negation in Souffle



Find all descendants of Bob that are not descendants of Alice

```
Dalice(y) :- Child('Alice',y)
Dalice(y) :- Dalice(x),Child(x,y)
Dbob(y) :- Child('Bob',y)
Dbob(y) :- Dbob(x), Child(x,y)


Answ(x) :- Dbob(x), !Dalice(x)
```

Two strata

Aggregates in Souffle

Find the minimum id of all Actors called ‘John’

Aggregates in Souffle

Find the minimum id of all Actors called ‘John’

```
Q(minId) :- minId = min x : { Actor(x, y, _), y = ‘John’ }
```

Aggregates in Souffle

Find the minimum id of all Actors called 'John'

An aggregate expression

```
Q(minId) :- minId = min x : { Actor(x, y, _), y = 'John' }
```

Aggregates in Souffle

Find the minimum id of all Actors called 'John'

An aggregate expression

```
Q(minId) :- minId = min x : { Actor(x, y, _), y = 'John' }
```

Anonymous variable

Aggregates in Souffle

Find the minimum id of all Actors called 'John'

An aggregate expression

The result variable

```
Q(minId) :- minId = min x : { Actor(x, y, _), y = 'John' }
```

Anonymous variable

Aggregates in Souffle

Find the minimum id of all Actors called 'John'

An aggregate expression

The result variable

```
Q(minId) :- minId = min x : { Actor(x, y, _), y = 'John' }
```

Anonymous variable

In SQL

```
SELECT min(id) as minId
FROM Actor as a
WHERE a.fname = 'John'
```

Aggregates in Souffle

- `count`
- `min`
- `max`
- `sum`

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

Counting

Count the number of actors called 'John'

```
Q(c) :- c = count : { Actor(_, y, _), y = 'John' }
```

No variable

Meaning (in SQL, assuming no NULLs)

```
SELECT count(*) as c
FROM Actor as a
WHERE a.name = 'John'
```

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

Group-By

```
Q(y,c) :- Movie(_,_,y), c = count : { Movie(_,_,y) }
```

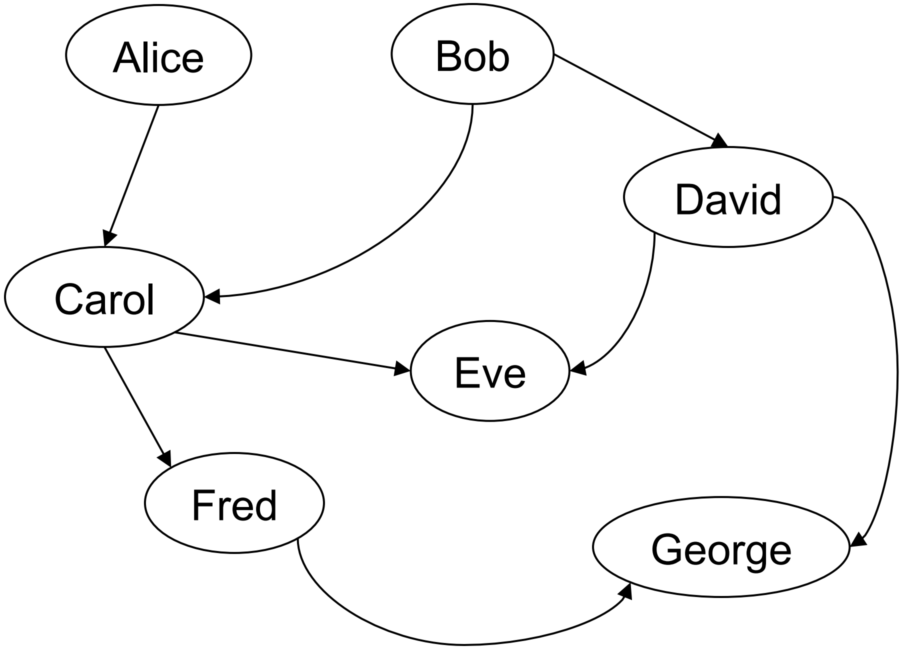
Group-by variable occurs in the head

Meaning (in SQL)

```
SELECT m.year, count(*)
FROM Movie as m
GROUP BY m.year
```

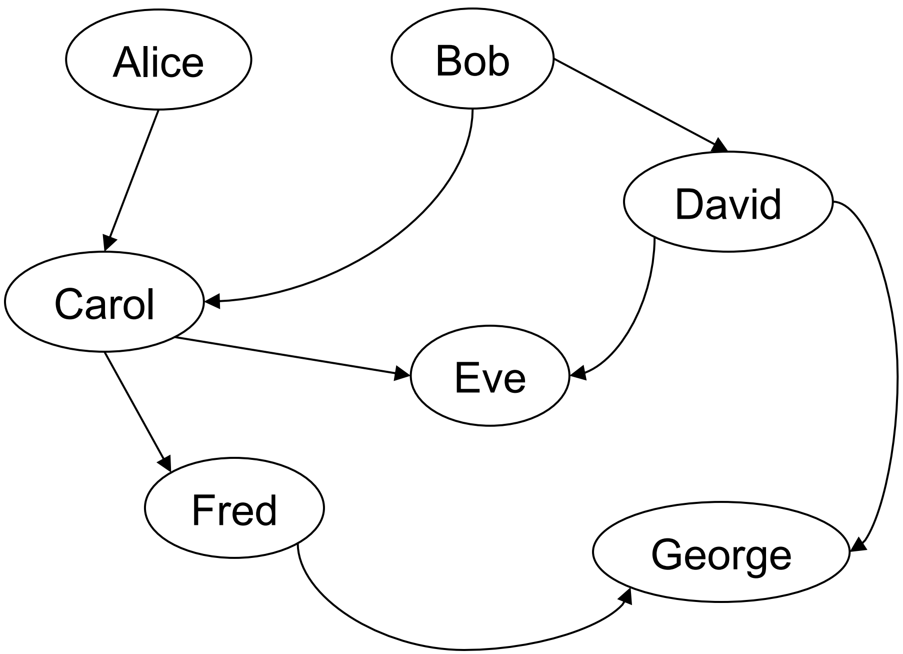
Group-By

For each person, count his/her descendants



Group-By

For each person, count his/her descendants

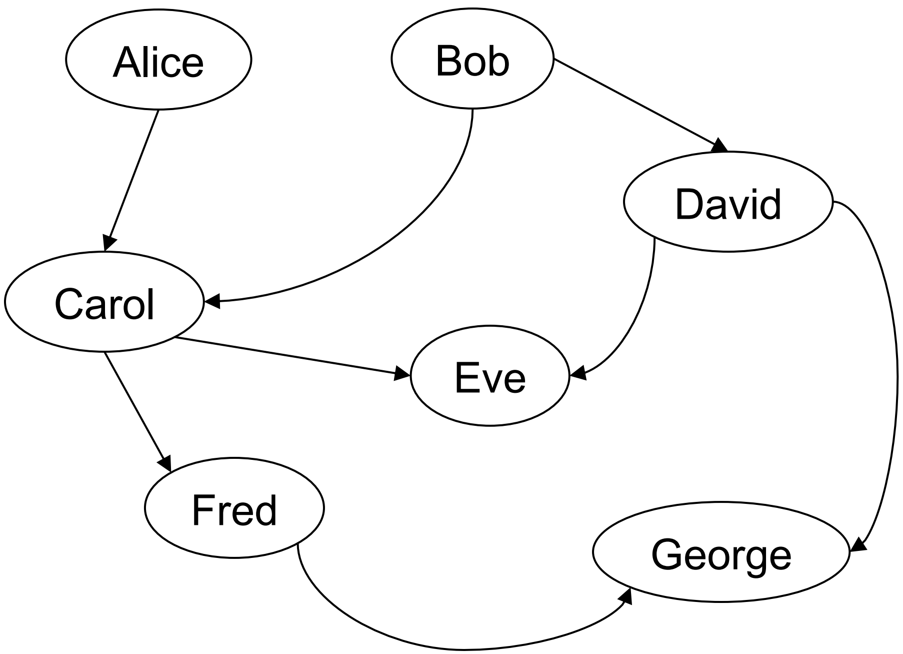


Answer

p	cnt
Alice	4
Bob	5
Carol	3
David	2
Fred	1

Group-By

For each person, count his/her descendants



Answer

p	cnt
Alice	4
Bob	5
Carol	3
David	2
Fred	1

Note: Eve and George do not appear in the answer (why?)

Group-By

```
// for each person, compute his/her descendants
D(x,y) :- Child(x,y).
D(x,z) :- D(x,y), Child(y,z).
```

Group-By

```
// for each person, compute his/her descendants
D(x,y) :- Child(x,y).
D(x,z) :- D(x,y), Child(y,z).

// For each person, count the number of descendants
T(p,c) :- D(p,_), c = count : { D(p,y) }.
```

Stratified

Group-By

```
// for each person, compute his/her descendants
D(x,y) :- Child(x,y).
D(x,z) :- D(x,y), Child(y,z).

// For each person, count the number of descendants
T(p,c) :- D(p,_), c = count : { D(p,y) }.
```

Stratified

How many descendants does Alice have?

Group-By

How many descendants does Alice have?

```
// for each person, compute his/her descendants
D(x,y) :- Child(x,y).
D(x,z) :- D(x,y), Child(y,z).

// For each person, count the number of descendants
T(p,c) :- D(p,_), c = count : { D(p,y) }.

// Find the number of descendants of Alice
Q(d) :- T(p,d), p = “Alice”.
```

Stratified

How many descendants does Alice have?

Stratified Datalog

- If we don't use aggregates or negation, then the Datalog program is already stratified

- If we do use aggregates or negation, it is usually quite natural to write the program in a stratified way

Safe/Unsafe Datalog Rules

- All rules in datalog must be safe
- We have seen only safe rules so far, what is an unsafe rule?
- Examples next, then the definition of safety

Unsafe Datalog Rules

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

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

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

```
U3(minId, y) :- minId = min x : { Actor(x, y, _) }
```

Unsafe Datalog Rules

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

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

y takes infinitely many values

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

```
U3(minId, y) :- minId = min x : { Actor(x, y, _) }
```

Unsafe Datalog Rules

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

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

y takes infinitely many values

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

x has no children?

Or there exists y who is not child of x?

```
U3(minId, y) :- minId = min x : { Actor(x, y, _) }
```

Unsafe Datalog Rules

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

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



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



```
U3(minId, y) :- minId = min x : { Actor(x, y, _) }
```

Unsafe Datalog Rules

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

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

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

A datalog rule is *safe* if every variable appears in some positive, non-aggregated relational atom

```
U3(minId, y) :- minId = min x : { Actor(x, y, _) }
```

Making Rules Safe

Return pairs (x,y) where x is a child of Alice, and y is anybody

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

Unsafe

Making Rules Safe

Return pairs (x,y) where x is a child of Alice, and y is anybody

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

Unsafe

Safe

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

Making Rules Safe

Find Alice's children who don't have children.

Unsafe

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

Making Rules Safe

Find Alice's children who don't have children.

Unsafe

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

Safe

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

Making Rules Safe

Find the smallest Actor ID and his/her first name

Unsafe

```
U3(minId, y) :- minId = min x : { Actor(x, y, _) }
```

Making Rules Safe

Find the smallest Actor ID and his/her first name

Unsafe

```
U3(minId, y) :- minId = min x : { Actor(x, y, _) }
```

Safe

```
U3(minId, y) :- minId = min x : { Actor(x, _, _) }, Actor(minID, y, _)
```

Recap of the Quarter

- Relational Model:
  - SQL
  - Data Models
- Query Engine:
  - Execution
  - Optimization (3 dimensions)
- Datalog

Some Things We Didn't Cover

- Transactions
- Provenance
- Tree decomposition, worst-case optimal algorithms
- LSM trees
- Push v.s. pull model

What you should do next

- Finish HW3
- Finish the project, meet on Friday
- Finish the project, present Wednesday
- Finish the project, submit final report
- Submit Review 4
- Finish HW4