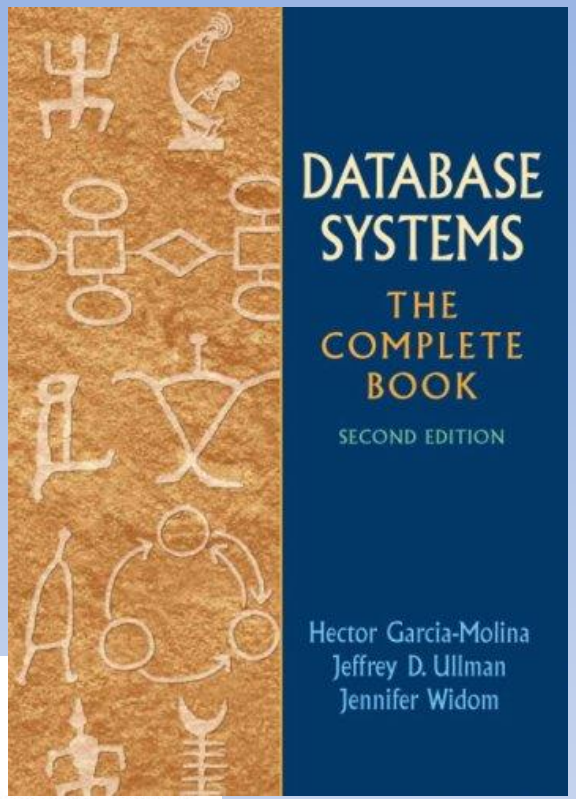
Chapter 5



Chapter 5

Algebraic and Logical Query Languages

5.3 A Logic for Relations

We now swi
databases.
guages, one

As an alternative to abstract query languages based on algebra, one can use a form of logic t
logic") consists
certain combin

5.4 Relational Algebra and Datalog

Each of the relational-algebra operators of Section 2.4 can be mimicked by one or several Datalog rules. In this section we shall consider each operator in turn. We shall then consider how to combine Datalog rules to mimic complex algebraic

Example 5.19: Consider the rule

$$\texttt{LongMovie(t,y)} \leftarrow \texttt{Movies(t,y,l,\_,\_,\_)} \text{ AND } \texttt{l} \geq 100$$

Logic as a Query Language

- If-then logical rules have been used in many systems.
  - Automated Reasoning Systems

- Perhaps Logic can be used for Queries?

A Logic for Relations: Datalog

- Database Logic
- Consists of If-Then Rules
- These rules express the idea:
  - IF: certain combinations of tuples exist in certain relations
  - THEN some other tuple must be in some other relation
    - Answering our query!

Predicates & Relations

- Datalog represents a relations as a Logic Predicate.
- R1 is a predicate in Datalog

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

Predicates & Relations

- Datalog Predicate takes a fixed number of arguments
- These arguments corresponds to the fixed schema for the corresponding relation
- Predicate R1 takes the arguments: K, A, B, C

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

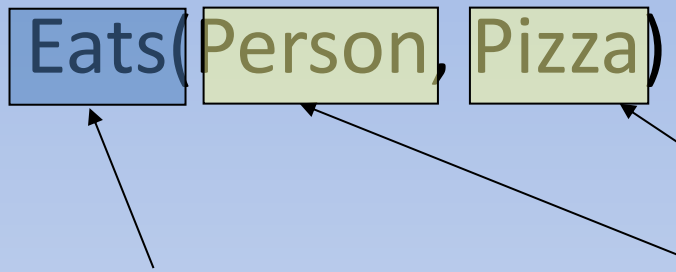
Atoms & Predicates

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

- ATOM: Is a Predicate together with its arguments.
  - R1(4, 2, 0 ,6)
  - R1(5, 2, 0, 5)
  - R1(0, 0, 0, 0)
- An atom is TRUE if its argument tuple is in the corresponding relation, and FALSE otherwise.
  - R1(4, 2, 0 ,6) is TRUE since (4, 2, 0, 6) IN R1.
  - R1(0, 0, 0, 0) is FALSE since (0, 0, 0, 0) NOT IN R1.
- Underscores are allowed to match any value.
  - R1(2, _, _, _) is TRUE because tuple (2, 1, 3, 7) exists.
- Database Semantics:
  - Predicates are false unless their arguments are in DB.

Example: Atom



The predicate = name of a relation

Arguments are variables (or constants).

We've created a small sample database to use for this assignment. It contains four relations:

```
Person(name, age, gender)          // name is a key
Frequents(name, pizzeria)          // [name,pizzeria] is a key
Eats(name, pizza)                  // [name,pizza] is a key
Serves(pizzeria, pizza, price)     // [pizzeria,pizza] is a key
```

Predicates & Queries

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

- Query Idea:
  - Define predicates based on logical expressions involving relations in database.
  - Query Result will be set of tuple arguments for the predicate that make it true.
- For Example:
  - R1(K, A, B, C) is true for each tuple in relation R1's current instance.
    - K, A, B, C are variables that can take on any value.
  - SELECT * FROM R1;

Predicates & Queries

R1

K	A	B	C
4	2	0	6
5	2	0	5
1	1	3	8
2	1	3	7
3	2	3	3

- Query Idea:
  - Define predicates based on logical expressions involving relations in database.
  - Query Result will be set of tuple arguments for the predicate that make it true.
- For Example:
  - R1(K, A, B, C) is true for each tuple in relation R1's current instance.
    - K, A, B, C are variables that can take on any value.
  - SELECT * FROM R1;
  - Underscore ('_') is an anonymous variable.

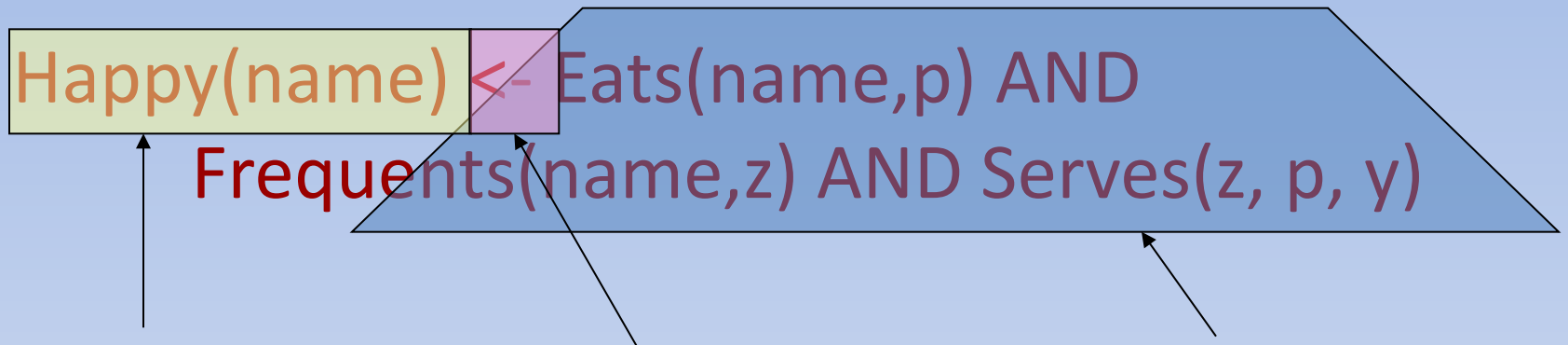
Arithmetic Atoms

- Datalog include Arithmetic Atoms in addition to Relational Atom
  - X<Y
  - X*Y < 10

Datalog Rules & Queries

- Datalog rules operate like operators of Relational Algebra

Anatomy of a Rule



Head = consequent,
a single subgoal

Read this
symbol “if”

Body = antecedent =
AND of *subgoals*.

```
Person(name, age, gender)          // name is a key
Frequents(name, pizzeria)          // [name,pizzeria] is a key
Eats(name, pizza)                  // [name,pizza] is a key
Serves(pizzeria, pizza, price)     // [pizzeria,pizza] is a key
```

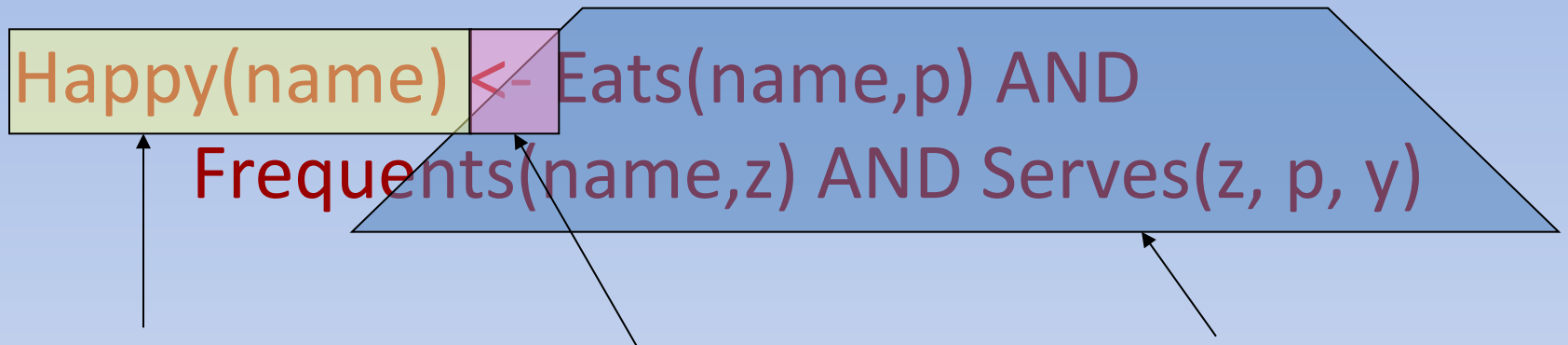
Subgoals Are Atoms

- An *atom* is a *predicate* (or relation name) with variables or constants as arguments.
- The head is an atom; the body is the AND of one or more atoms.
- Convention: Predicates begin with a capital, variables begin with lower-case.

Interpreting Rules

- A variable appearing in the head is *distinguished* ; otherwise it is *nondistinguished*.
- Rule meaning: The head is true for given values of the distinguished variables if there exist values of the nondistinguished variables that make all subgoals of the body true.

Anatomy of a Rule



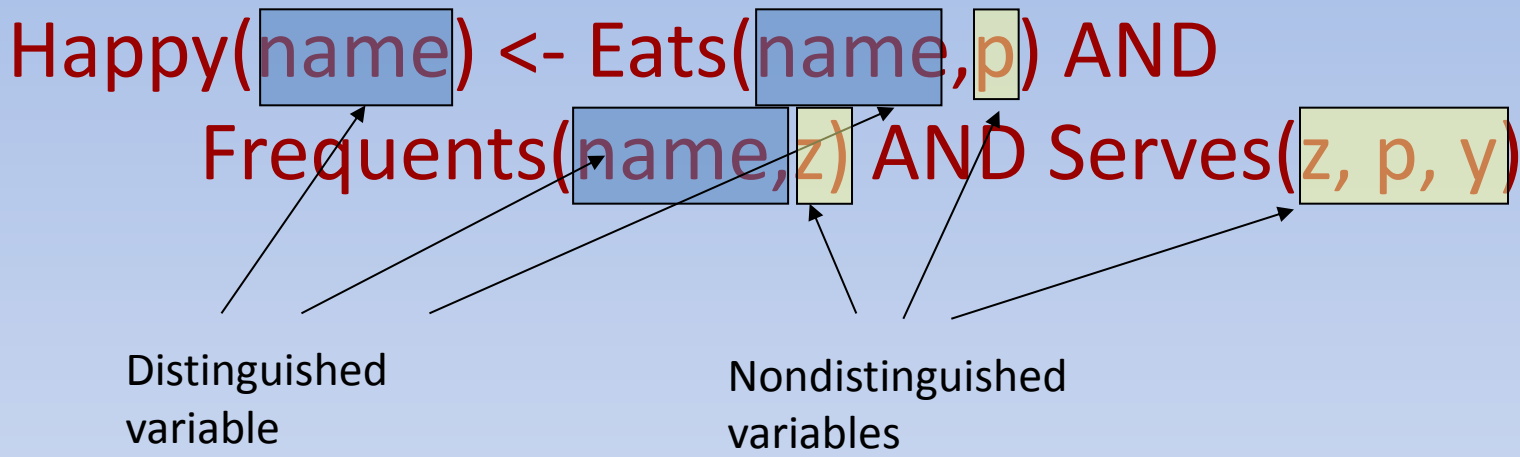
Head = consequent,
a single subgoal

Read this
symbol “if”

Body = antecedent =
AND of *subgoals*.

```
Person(name, age, gender)          // name is a key
Frequents(name, pizzeria)          // [name,pizzeria] is a key
Eats(name, pizza)                  // [name,pizza] is a key
Serves(pizzeria, pizza, price)     // [pizzeria,pizza] is a key
```

Example: Interpretation



Interpretation: Pizza lover *name* is happy if there exist a Pizzeria z, and a pizza p, such that *name* frequents the pizzeria serving the pizza, getting to eat the pizza

Applying a Rule

- Approach 1: consider all combinations of values of the variables.
- If all subgoals are true, then evaluate the head.
- The resulting head is a tuple in the result.

Example: Rule Evaluation

Happy(name) <- Eats(name,p) AND
Frequents(name,z) AND Serves(z, p, y)

FOR (each name, p, z)

IF (Frequents(name,z), Eats(name,p), and Serves(z,p,y) are all true)

add Happy(name) to the result

- Note: set semantics so add only once.

A Glitch (Fixed Later)

- Relations are finite sets.
- We want rule evaluations to be finite and lead to finite results.
- "Unsafe" rules like P(x)<-Q(y) have infinite results, even if *Q* is finite.
- Even P(x)<-Q(x) requires examining an infinity of *x*-values.

Applying a Rule – (2)

- Approach 2: For each subgoal, consider all tuples that make the subgoal true.
- If a selection of tuples define a single value for each variable, then add the head to the result.
- Leads to finite search for P(x)<-Q(x)
  - but P(x)<-Q(y) is problematic.

Example: Rule Evaluation – (2)

Happy(name) <- Eats(name,p) AND
Frequents(name,z) AND Serves(z, p, y)

FOR (each f in Frequents, e in Eats, and
s in Serves)

IF (f[1]=e[1] and f[2]=s[1] and
e[2]=s[2])

add Happy(f[1]) to the result

Arithmetic Subgoals

- In addition to relations as predicates, a predicate for a subgoal of the body can be an arithmetic comparison.
- We write arithmetic subgoals in the usual way, e.g., $x < y$.

Negated Subgoals

- NOT in front of a subgoal negates its meaning.
- Example: Think of Arc(a,b) as arcs in a graph.
  - S(x,y) says the graph is not transitive from *x* to *y* ; i.e., there is a path of length 2 from *x* to *y*, but no arc from *x* to *y*.

S(x,y) <- Arc(x,z) AND Arc(z,y)
AND NOT Arc(x,y)

Safe Rules

- A rule is *safe* if:
  1. Each distinguished variable,
  2. Each variable in an arithmetic subgoal, and
  3. Each variable in a negated subgoal,

  also appears in a nonnegated, relational subgoal.
  - GROUNDED
- Safe rules prevent infinite results.

Example: Unsafe Rules

- Each of the following is unsafe and not allowed:
  1. S(x) <- R(y)
  2. S(x) <- R(y) AND NOT R(x)
  3. S(x) <- R(y) AND x < y
- In each case, an infinity of *x*'s can satisfy the rule, even if *R* is a finite relation.

An Advantage of Safe Rules

- We can use “approach 2” to evaluation, where we select tuples from only the nonnegated, relational subgoals.
- The head, negated relational subgoals, and arithmetic subgoals thus have all their variables defined and can be evaluated.

Datalog Programs

- *Datalog program* = collection of rules.
- In a program, predicates can be either
  1. EDB = *Extensional Database* = stored table.
  2. IDB = *Intensional Database* = relation defined by rules.
- No EDB in heads.

Evaluating Datalog Programs

- As long as there is no recursion, we can pick an order to evaluate the IDB predicates, so that all the predicates in the body of its rules have already been evaluated.
- If an IDB predicate has more than one rule, each rule contributes tuples to its relation.

Example: Datalog Program

- Using EDB Serves(pizzeria, pizza) and Eats(name, pizza), find the pizzas Joe doesn't eat.

JoeEats(j) <- Eats('Joe', j)

Answer(pizza) <- Serves(z,pizza,y)

AND NOT JoeEats(pizza)

Example: Evaluation

- Step 1: Examine all Eats tuples with first component 'Joe'.
  - Add the second component to JoeEats.
- Step 2: Examine all Serves tuples (z,pizza,y).
  - If *pizza* is not in JoeEats, add *pizza* to Answer.

Expressive Power of Datalog

- Without recursion, Datalog can express all and only the queries of core relational algebra.
  - The same as SQL select-from-where, without aggregation and grouping.
- But with recursion, Datalog can express more than these languages.
- Yet still not Turing-complete.

Recursive Example

- EDB: Par(c,p) = *p* is a parent of *c*.
- Generalized cousins: people with common ancestors one or more generations back:

Sib(x,y) <- Par(x,p) AND Par(y,p) AND x<>y

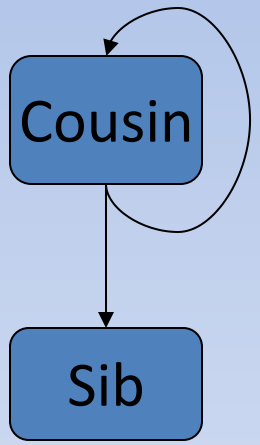
Cousin(x,y) <- Sib(x,y)

Cousin(x,y) <- Par(x,xp) AND Par(y,yp)
AND Cousin(xp,yp)

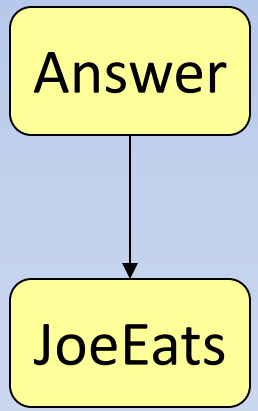
Definition of Recursion

- Form a *dependency graph* whose nodes = IDB predicates.
- Arc *X* ->*Y* if and only if there is a rule with *X* in the head and *Y* in the body.
- Cycle = recursion; no cycle = no recursion.

Example: Dependency Graphs



Recursive

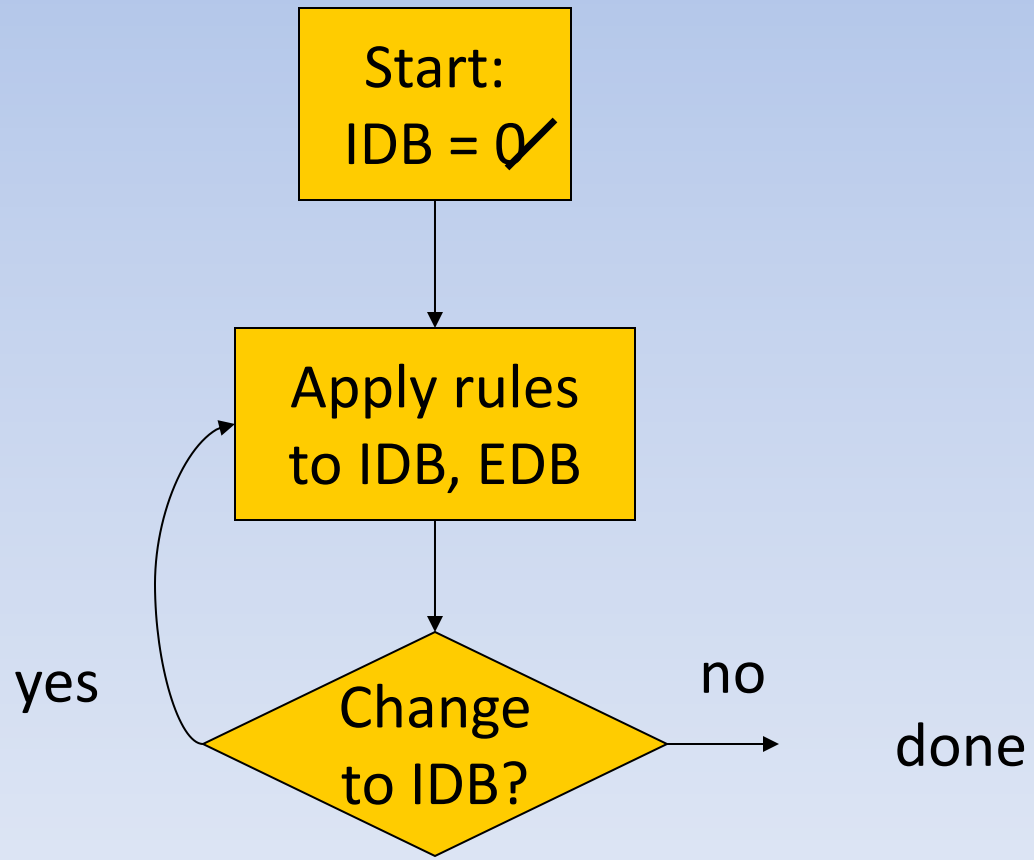


Nonrecursive

Evaluating Recursive Rules

- The following works when there is no negation:
  1. Start by assuming all IDB relations are empty.
  2. Repeatedly evaluate the rules using the EDB and the previous IDB, to get a new IDB.
  3. End when no change to IDB.

The “Naïve” Evaluation Algorithm



Seminaive Evaluation

- Since the EDB never changes, on each round we only get new IDB tuples if we use at least one IDB tuple that was obtained on the previous round.
- Saves work; lets us avoid rediscovering *most* known facts.
  - A fact could still be derived in a second way.

Example: Evaluation of Cousin

- We'll proceed in rounds to infer Sib facts (red) and Cousin facts (green).
- Remember the rules:

Sib(x,y) <- Par(x,p) AND Par(y,p) AND x<>y

Cousin(x,y) <- Sib(x,y)

Cousin(x,y) <- Par(x,xp) AND Par(y,yp)
AND Cousin(xp,yp)

Par Data: Parent Above Child

Sib(x,y) <- Par(x,p) AND Par(y,p) AND x<>y

Cousin(x,y) <- Par(x,xp) AND Par(y,yp)
AND Cousin(xp,yp)

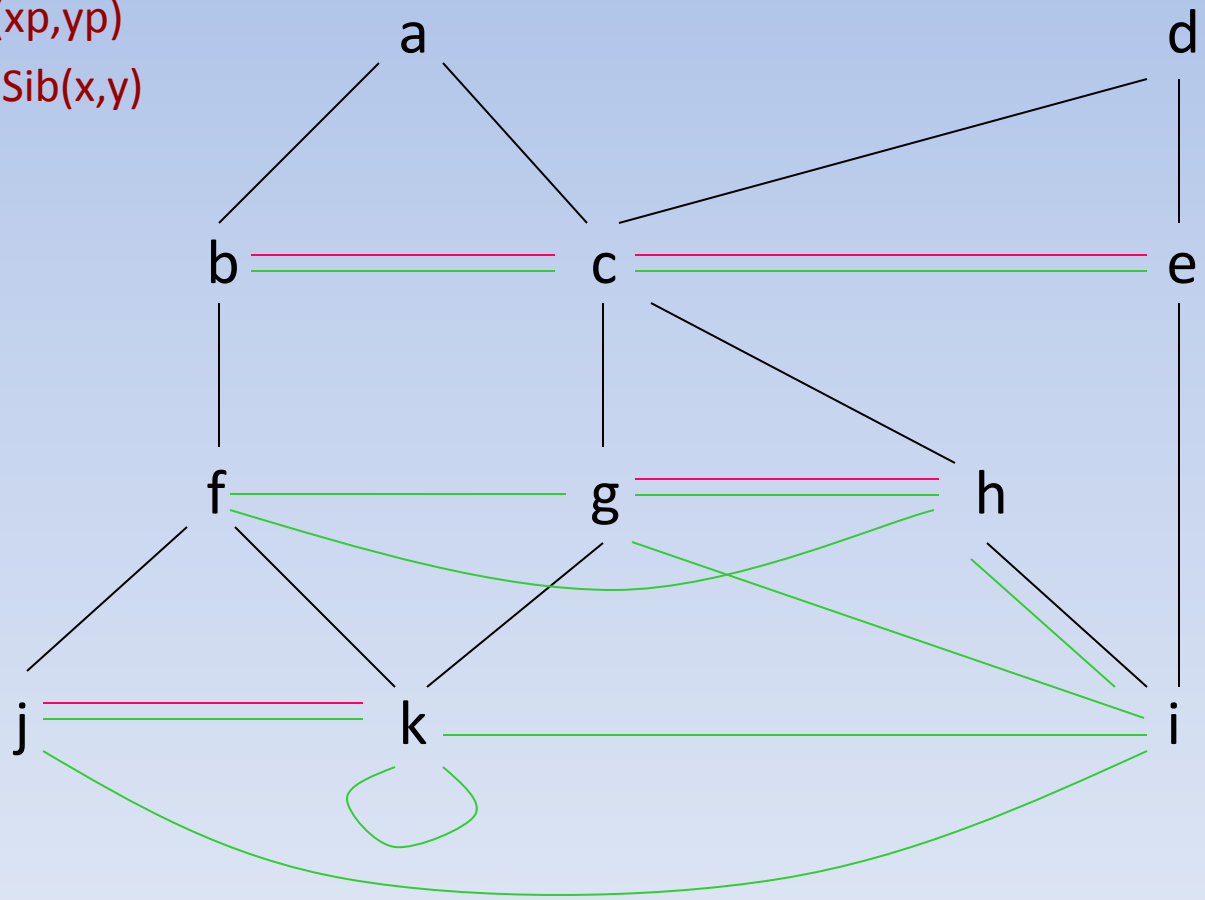
Cousin(x,y) <- Sib(x,y)

Round 1

Round 2

Round 3

Round 4



SQL-99 Recursion

- Datalog recursion has inspired the addition of recursion to the SQL-99 standard.
- Tricky, because SQL allows negation grouping-and-aggregation, which interact with recursion in strange ways.

Example:
Exercise – 2.4.1

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- Laptop(model, speed, ram, hd, screen, price)
- Printer(model, color, type, price)

a) What PC models have a speed of at least 3.00?

a) What PC models have a speed of at least 3.00?

- R1 := $\sigma_{speed \geq 3.00}$ (PC)
- R2 := π_{model}(R1)

model
1005
1006
1013

2.4.1a

- Answer(model) ← PC(model,speed,_,_,_) AND speed ≥ 3.00

Example:
Exercise – 2.4.1

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- Laptop(model, speed, ram, hd, screen, price)
- Printer(model, color, type, price)

b) Which manufacturers make laptops with a hard disk of at least 100gb

b) Which manufacturers make laptops with a hard disk of at least 100gb

- R1 := $\sigma_{hd \geq 100}$ (Laptop)
- R2 := Product (R1)
- R3 := π_{maker} (R2)

maker
E
A
B
F
G

2.4.1b

- Answer(maker) ← Laptop(model,_,_,hd,_,_) AND Product(maker,model,_) AND hd ≥ 100

Example:
Exercise – 2.4.1

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- Laptop(model, speed, ram, hd, screen, price)
- Printer(model, color, type, price)

c) Find all model number and price of all products (of any type) made by manufacturer B

c) Find all model number and price of all products (of any type) made by manufacturer B

- R1 := $\sigma_{maker=B}$ (Product $\bowtie$ PC)
- R2 := $\sigma_{maker=B}$ (Product $\bowtie$ Laptop)
- R3 := $\sigma_{maker=B}$ (Product $\bowtie$ Printer)
- R4 := $\pi_{model,price}$ (R1)
- R5 := $\pi_{model,price}$ (R2)
- R6: = $\pi_{model,price}$ (R3)
- R7 := R4 $\cup$ R5 $\cup$ R6

model	price
1004	649
1005	630
1006	1049
2007	1429

2.4.1c

Answer(model,price) ← PC(model,_,_,_,price) AND Product(maker,model,_) AND maker=’B’

Answer(model,price) ← Laptop(model,_,_,_,_,price) AND Product(maker,model,_) AND maker=’B’

Answer(model,price) ← Printer(model,_,_,price) AND Product(maker,model,_) AND maker=’B’

Example
Exercise 2.4.1

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- Laptop(model, speed, ram, hd, screen, price)
- Printer(model, color, type, price)

Ex: 2.4.1.d) Find the model numbers of all color laser printers

Ex: 2.4.1.e) Find those manufacturers (maker) that sell laptops, but not PC's.

2.4.1d

- Answer(model) ← Printer(model,color,type,_) AND color=’true’ AND type=’laser’

2.4.1e

PCMaker(maker) ← Product(maker,_,type) AND type=’pc’

LaptopMaker(maker) ← Product(maker,_,type) AND type=’laptop’

Answer(maker) ← LaptopMaker(maker) AND NOT PCMaker(maker)