Databases 1

Logical Query Language: Datalog

Datalog: Logic As a Query Language

- Abstract Query Languages:
  - Relational Algebra (procedural $\rightarrow$ optimization)
  - Logical QL: Datalog, Rel.Calculus (declarative)
- Datalog = ‘Data’- Database, ‘log’- logic,Prolog
- If-then logical rules have been used in many systems.
- Nonrecursive rules are equivalent to the core relational algebra.
- Recursive rules extend relational algebra and appear in SQL-99.

Intuitive introduction

- Example 1: Ancestors

  ParentOf(parent,child)

  - Find all of Mary’s ancestors

- Example 2: Company hierarchy

  Employee(ID,salary)

  Manager(mID,eID)

  Project(name,mgrID)

  - Find total salary cost of project ‘X’

- Example 3: Airline flights

  Flight(orig,dest,airline,cost)

  - Find cheapest way to fly from ‘A’ to ‘B’

A Logical Rule

- Our first example of a rule uses the relations
  Frequents(drinker, bar),
  Likes(drinker, beer),
  Sells(bar, beer, price).
- The rule is a query asking for “happy” drinkers --- those that frequent a bar that serves a beer that they like.

Anatomy of a Rule

Happy(d) ← Frequents(d,bar) AND Likes(d,beer) AND Sells(bar,beer,p)

Head = consequent, a single subgoal

Body = antecedent = AND of *subgoals*.

Read this symbol “if”

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.

Example: Atom

(1) Sells(bar, beer, p)

The predicate = name of a relation

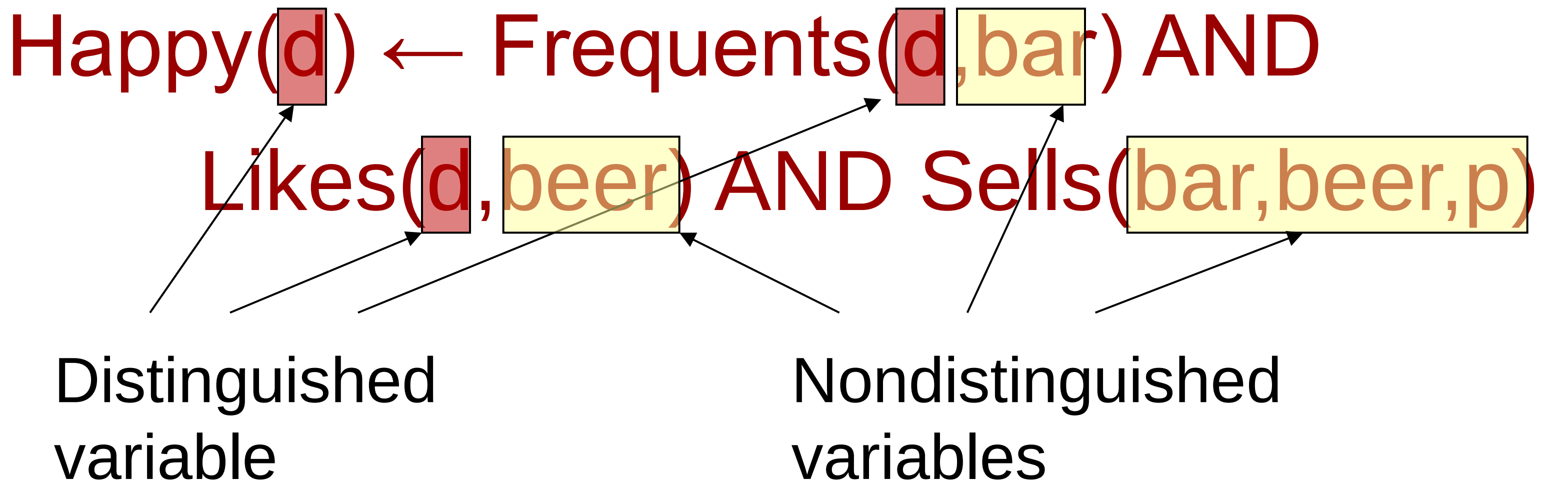
Arguments are variables (or constants).

(2) Fruits(x, x, y, 5, ‘apple’)

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.

Example: Interpretation


Happy(d) ← Frequents(d,bar) AND Likes(d,beer) AND Sells(bar,beer,p)

Interpretation: drinker *d* is happy if there exist a bar, a beer, and a price *p* such that *d* frequents the bar, likes the beer, and the bar sells the beer at price *p*.

Applying a Rule --- (1)

- 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(d) ← Frequents(d,bar) AND
        Likes(d,beer) AND Sells(bar,beer,p)
FOR (each d, bar, beer, p)
  IF (Frequents(d,bar), Likes(d,beer), and
  Sells(bar,beer,p) are all true)
        add Happy(d) to the result
```

- Note: set semantics so add only once.
- Set semantics vice versa bag semantics

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.

Happy(d) ← Frequents(d,bar) AND
        Likes(d,beer) AND Sells(bar,beer,p)

```
FOR (each f in Frequents, i in Likes,
                                    and s in Sells)
IF (f[1]=i[1] and f[2]=s[1] and i[2]=s[2])
     add Happy(f[1]) to the result
```

A Glitch (Fixed Later)

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

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., $x < y$.

Example: Arithmetic

- A beer is "cheap" if there are at least two bars that sell it for under $2.

```
Cheap(beer) ← Sells(bar1,beer,p1) AND
        Sells(bar2,beer,p2) AND p1 < 2.00
        AND p2 < 2.00 AND bar1 <> bar2
```

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.
- Safe rules prevent infinite results.

Example: Unsafe Rules

- Each of the following is unsafe and not allowed:
  1. $S(x) \leftarrow R(y)$
  2. $S(x) \leftarrow R(y)$ AND $x < y$
  3. $S(x) \leftarrow R(y)$ AND NOT $R(x)$
- 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.
- Never both! 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 Sells(bar, beer, price) and Beers(name, manf), find the manufacturers of beers Joe doesn't sell.

JoeSells(b) ← Sells('Joe''s Bar', b, p)

Answer(m) ← Beers(b,m)

                    AND NOT JoeSells(b)

Example: Evaluation

- Step 1: Examine all Sells tuples with first component ’Joe’’s Bar’.
  - Add the second component to JoeSells.
- Step 2: Examine all Beers tuples (b,m).
  - If *b* is not in JoeSells, add *m* 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.

Expressive Power of Datalog

- Relational algebra: $\Pi_{\mathbf{List}}(\sigma_{\mathbf{Cond}}(\mathbf{R} \bowtie \mathbf{S} \bowtie \ldots))$

  SQL : SELECT list

  FROM join of tables

  WHERE cond

  A Datalog rule
- Set operators:

Relációs algebra és Datalog ---1

$R(x_1,\dots,x_n)$, $S(x_1,\dots,x_n)$

predikátumokhoz tartozó reláció $R(A_1,\dots,A_n)$, $S(A_1,\dots,A_n)$

- $R \cap S$ metszetnek megfelelő szabály:

  $Q(x_1,\dots,x_n) \leftarrow R(x_1,\dots,x_n)$ AND $S(x_1,\dots,x_n)$

- Who likes apple and pear?

  Q(N) ← Likes(N,’apple’) AND Likes(N,’pear’)

Relációs algebra és Datalog ---1

- $R \cap S$ Datalog Rule :

  $Q(x_1,\ldots,x_n) \leftarrow R(x_1,\ldots,x_n)$ AND $S(x_1,\ldots,x_n)$

- $R-S$ Datalog Rule :

  $Q(x_1,\ldots,x_n) \leftarrow R(x_1,\ldots,x_n)$ AND NOT $S(x_1,\ldots,x_n)$

- $R \cup S$ Datalog Program :

  $Q(x_1,\ldots,x_n) \leftarrow R(x_1,\ldots,x_n)$
  $Q(x_1,\ldots,x_n) \leftarrow S(x_1,\ldots,x_n)$

Relációs algebra és Datalog ---2

Kiválasztás:

- $\sigma_{xi\ \theta\ xj}$ (R) kifejezésnek megfelelő szabály :
  $Válasz(x_1,\ldots,x_n) \leftarrow R(x_1,\ldots,x_n)$ AND $x_i \theta x_j$
- $\sigma_{xi\ \theta\ c}$(E1) kifejezésnek megfelelő szabály:
  $Válasz(x_1,\ldots,x_n) \leftarrow R(x_1,\ldots,x_n)$ AND $x_i \theta c$

Vetítés:

- $\Pi_{Ai1,\ldots,Aik}$(R) kifejezésnek megfelelő szabály:
  $Válasz(x_{i_1},\ldots,x_{i_k}) \leftarrow R(x_1,\ldots,x_n)$

Megjegyzés: név nélküli anonymus változók, amelyek csak egyszer szerepelnek és mindegy a nevük azt aláhúzás helyettesítheti. Például:

HosszúFilm(c,é) ← Film(c,é,h,_,_,_) AND h ≥ 100

Relációs algebra és Datalog ---3

Természetes összekapcsolás: Tegyük fel, hogy $R(A_1,\ldots,A_n, C_1, \ldots, C_k)$ és $S(B_1,\ldots,B_m, C_1, \ldots, C_k)$

- $R \bowtie S$ kifejezésnek megfelelő szabály:

$\text{Válasz}(x_1,\ldots,x_n,y_1,\ldots,y_m, z_1,\ldots, z_k) \leftarrow$

$\leftarrow R(x_1,\ldots,x_n, z_1, \ldots, z_k) \text{ AND } S(y_1,\ldots,y_m, z_1, \ldots, z_k)$

- Who likes apple and pear?

Q(N) ← Likes(N,’apple’) AND Likes(N,’pear’)

Databases 1

Recursion

Intuitive introduction

- Example 1: Ancestors

  ParentOf(parent,child)

  - Find all of Mary's ancestors

- Example 2: Company hierarchy

  Employee(ID,salary)

  Manager(mID,eID)

  Project(name,mgrID)

  - Find total salary cost of project 'X'

- Example 3: Airline flights

  Flight(orig,dest,airline,cost)

  - Find cheapest way to fly from 'A' to 'B'

Another 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

$$\begin{tikzcd}
\text{Cousin} \arrow[loop right] \arrow[d] \\
\text{Sib}
\end{tikzcd}$$

Recursive

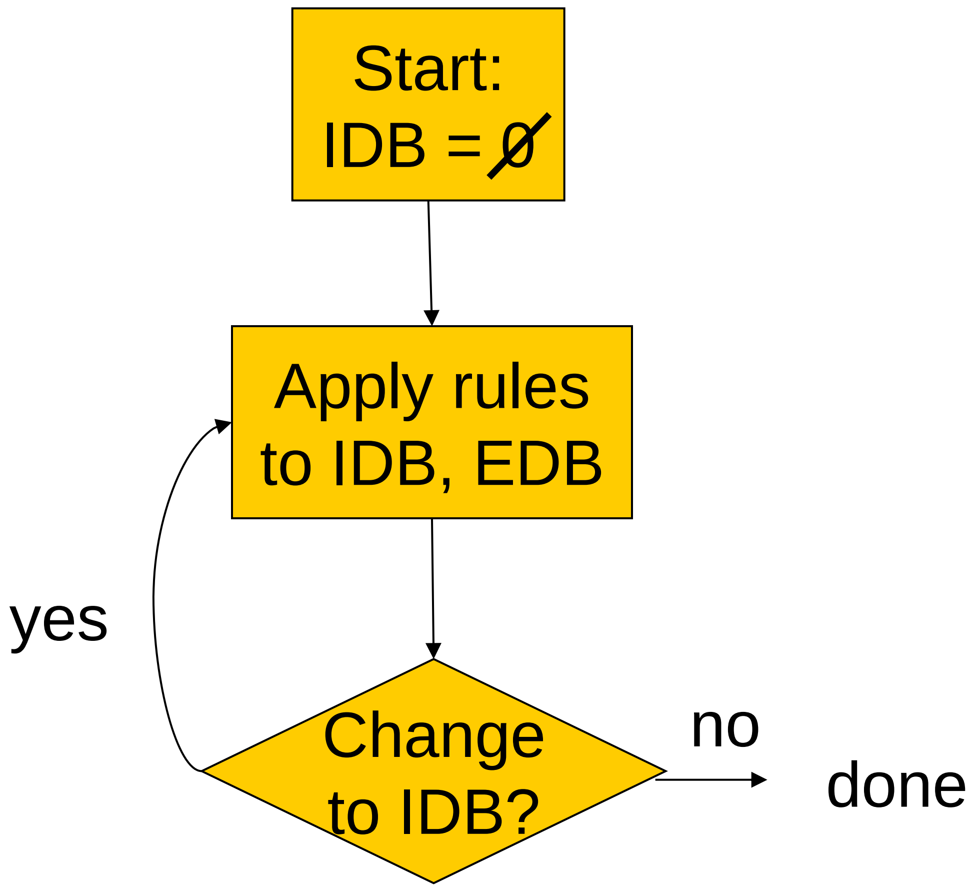
$$\begin{tikzcd}
\text{Answer} \arrow[d] \\
\text{JoeSells}
\end{tikzcd}$$

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



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.
- Define only the “monotone” recursions.

Form of SQL Recursive Queries

```
WITH
   <stuff that looks like Datalog rules>
<a SQL query about EDB, IDB>

“Datalog rule” =
   [RECURSIVE] <name>(<arguments>)
   AS <query>
```

Example: SQL Recursion ---(1)

- Find Sally’s cousins, using SQL like the recursive Datalog example.
- Par(child,parent) is the EDB.

```
WITH
Sib(x,y) AS
    SELECT p1.child, p2.child
    FROM Par p1, Par p2
    WHERE p1.parent = p2.parent AND
    p1.child <> p2.child;
```

Like Sib(x,y) ← Par(x,p) AND Par(y,p) AND x <> y

Example: SQL Recursion --- (2)

Required – Cousin is recursive

```
RECURSIVE Cousin(x,y) AS
  (SELECT * FROM Sib)
      UNION
  (SELECT p1.child, p2.child
   FROM Par p1, Par p2, Cousin
   WHERE p1.parent = Cousin.x AND
   p2.parent = Cousin.y);
```

Reflects Cousin(x,y) ← Sib(x,y)

Reflects Cousin(x,y) ← Par(x,xp) AND Par(y,yp) AND Cousin(xp,yp)

Example: SQL Recursion --- (3)

- With those definitions, we can add the query, which is about the virtual view Cousin(x,y):

```
SELECT y
FROM Cousin
WHERE x = ’Sally’;
```

Book example (Chapter 10.2 Recursion)

- Flights(airline, fromcity, tocity, departs, arrives)
- For what pairs of cities(x, y) is it possible to get from city x to city y by taking one or more flights?
- Reaches(x, y) <- Flights (_, x, y, _, _, _)
  Reaches (x, y) <- Reaches (x, z) AND Flights (_, z, y, _, _)
- WITH RECURSIVE Reaches AS

```
        (SELECT fromcity, tocity FROM Flights
    UNION
        (SELECT Reaches. fromcity, Flights.tocity
         FROM Reaches, Flights
         WHERE Reaches.tocity = Flights.fromcity)
  SELECT * FROM Reaches;
```