Slides to be used with Ullman/Widom *A First Course in Database Systems*, Prentice-Hall, 1997

## Logical Query Languages

Motivation:

1. Logical rules extend more naturally to recursive queries than does relational algebra.
   - ❖ Used in SQL3 recursion.
2. Logical rules form the basis for many information-integration systems and applications.

## Datalog Example

```
Likes(drinker, beer)
Sells(bar, beer, price)
Frequents(drinker, bar)

Happy(d) <-
    Frequents(d,bar) AND
    Likes(d,beer) AND
    Sells(bar,beer,p)
```

- Above is a *rule*.
- Left side = *head*.
- Right side = *body* = `AND` of *subgoals*.
- Head and subgoals are *atoms*.
  - Atom = *predicate* and arguments.
  - Predicate = relation name or arithmetic predicate, e.g. $<$.
  - Arguments are variables or constants.
- Subgoals (not head) may optionally be negated by `NOT`.

## Meaning of Rules

Head is true of its arguments if there exist values for *local* variables (those in body, not in head) that make all of the subgoals true.

- If no negation or arithmetic comparisons, just natural join the subgoals and project onto the head variables.

## Example

Above rule equivalent to `Happy(d)` = $\pi_{drinker}$(`Frequents` $\bowtie$ `Likes` $\bowtie$ `Sells`)

## Evaluation of Rules

Two, dual, approaches:

1. *Variable-based*: Consider all possible assignments of values to variables. If all subgoals are true, add the head to the result relation.

2. *Tuple-based*: Consider all assignments of tuples to subgoals that make each subgoal true. If the variables are assigned consistent values, add the head to the result.

## Example: Variable-Based Assignment

```
S(x,y) <- R(x,z) AND R(z,y)
    AND NOT R(x,y)
```

$R =$

| A | B |
|---|---|
| 1 | 2 |
| 2 | 3 |

- Only assignments that make first subgoal true:

  1. $x \rightarrow 1$, $z \rightarrow 2$.

  2. $x \rightarrow 2$, $z \rightarrow 3$.

- In case (1), $y \rightarrow 3$ makes second subgoal true. Since $(1, 3)$ is *not* in $R$, the third subgoal is also true.

  - ❖ Thus, add $(x, y) = (1, 3)$ to relation $S$.

- In case (2), no value of $y$ makes the second subgoal true. Thus, $S$ =

| A | B |
|---|---|
| 1 | 3 |

## Example: Tuple-Based Assignment

Trick: start with the positive (not negated), relational (not arithmetic) subgoals only.

```
S(x,y) <- R(x,z) AND R(z,y)
    AND NOT R(x,y)
```

$R =$

| A | B |
|---|---|
| 1 | 2 |
| 2 | 3 |

- Four assignments of tuples to subgoals:

| $R(x,z)$ | $R(z,y)$ |
|---|---|
| $(1,2)$ | $(1,2)$ |
| $(1,2)$ | $(2,3)$ |
| $(2,3)$ | $(1,2)$ |
| $(2,3)$ | $(2,3)$ |

- Only the second gives a consistent value to $z$.
- That assignment also makes `NOT R(x,y)` true.
- Thus, $(1,3)$ is the only tuple for the head.

## Safety

A rule can make no sense if variables appear in funny ways.

## Examples

- `S(x) <- R(y)`
- `S(x) <- NOT R(x)`
- `S(x) <- R(y) AND x < y`

In each of these cases, the result is infinite, even if the relation $R$ is finite.

- To make sense as a database operation, we need to require three things of a variable $x$. If $x$ appears in either

  1. The head,
  2. A negated subgoal, or
  3. An arithmetic comparison,

  then $x$ must also appear in a nonnegated, "ordinary" (relational) subgoal of the body.

- We insist that rules be safe, henceforth.

## Datalog Programs

- A collection of rules is a *Datalog program.*
- Predicates/relations divide into two classes:
  - EDB = *extensional database* = relation stored in DB.
  - IDB = *intensional database* = relation defined by one or more rules.
- A predicate must be IDB or EDB, not both.
  - Thus, an IDB predicate can appear in the body or head of a rule; EDB only in the body.

## Example

Convert the following SQL (Find the manufacturers of the beers Joe sells):

```
Beers(name, manf)
Sells(bar, beer, price)

SELECT manf
FROM Beers
WHERE name IN(
    SELECT beer
    FROM Sells
    WHERE bar = 'Joe''s Bar'
);
```

to a Datalog program.

```
JoeSells(b) <-
    Sells('Joe''s Bar', b, p)
Answer(m) <-
    JoeSells(b) AND Beers(b,m)
```

- Note: `Beers`, `Sells` = EDB; `JoeSells`, `Answer` = IDB.

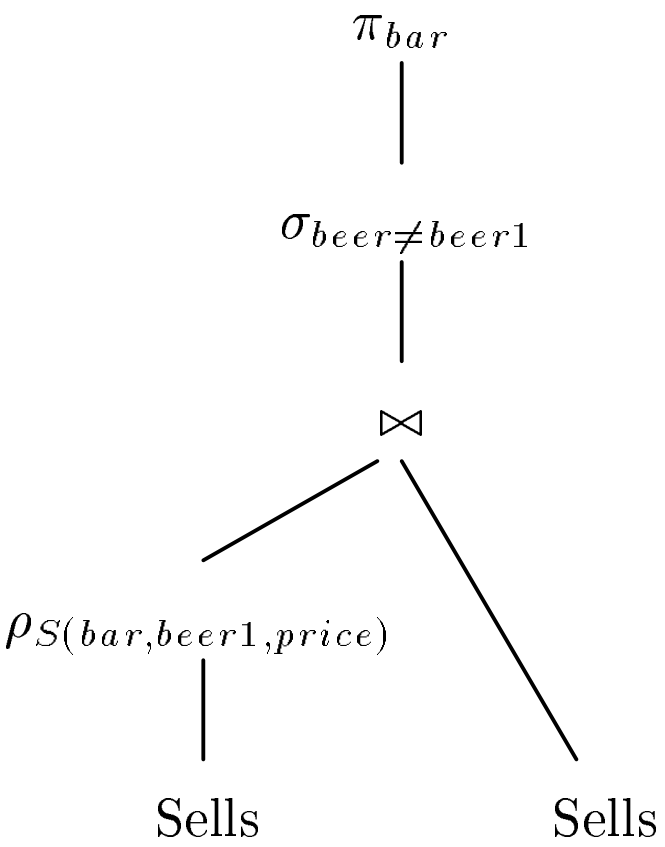
## Expressive Power of Datalog

- Nonrecursive Datalog = relational algebra.
- Datalog simulates SQL select-from-where without aggregation and grouping.
- Recursive Datalog expresses queries that cannot be expressed in SQL.
- But none of these languages have full expressive power (*Turing completeness*).

## Relational Algebra to Datalog

- Text has constructions for each of the operators of R.A.
  - ❖ Only hard part: selections with OR's and NOT's.
- Simulate a R.A. expression in Datalog by creating an IDB predicate for each interior node and using the constuction for the operator at that node.

**Example**: Find the bars that sell two different beers at the same price.




```
R1(bar,beer1,beer,price) <-
    Sells(bar,beer1,price) AND
    Sells(bar,beer,price);
R2(bar,beer1,beer,price) <-
    R1(bar,beer1,beer,price) AND
    beer1 <> beer;
Answer(bar) <-
    R2(bar,beer1,beer,price);
```

## Datalog to Relational Algebra

- General rule is complex; the following often works for single rules:
  - ❖ Problems not handled: constant arguments and variables appearing twice in the same atom.
  - ❖ Can you provide the necessary fixes?
  1. Use $\rho$ to create for each relational subgoal a relation whose schema is the variables of that subgoal.
  2. Handle negated subgoals by finding an expression for the finite set of all possible values for each of its variables ($\pi$ a suitable column) and take their product. Then subtract.
  3. Natural join the relations from (1), (2).
  4. Get the effect of arithmetic comparisons with $\sigma$.
  5. Project onto head with $\pi$.
- Several rules for same predicate: use $\cup$.

## Example

```
S(x,y) <- R(x,z) AND R(z,y)
    AND NOT R(x,y)
```

S1(x,y,z) := $\rho_{R1(x,z)}$(R) $\bowtie$ $\rho_{R2(z,y)}$(R);
S2(x,y) := $\pi_x$(S1) $\times$ $\pi_y$(S1);
S3(x,y) := S2 $-$ $\rho_{R3(x,y)}$(R);
S(x,y) := $\pi_{x,y}$(S1(x,y,z) $\bowtie$ S3(x,y));

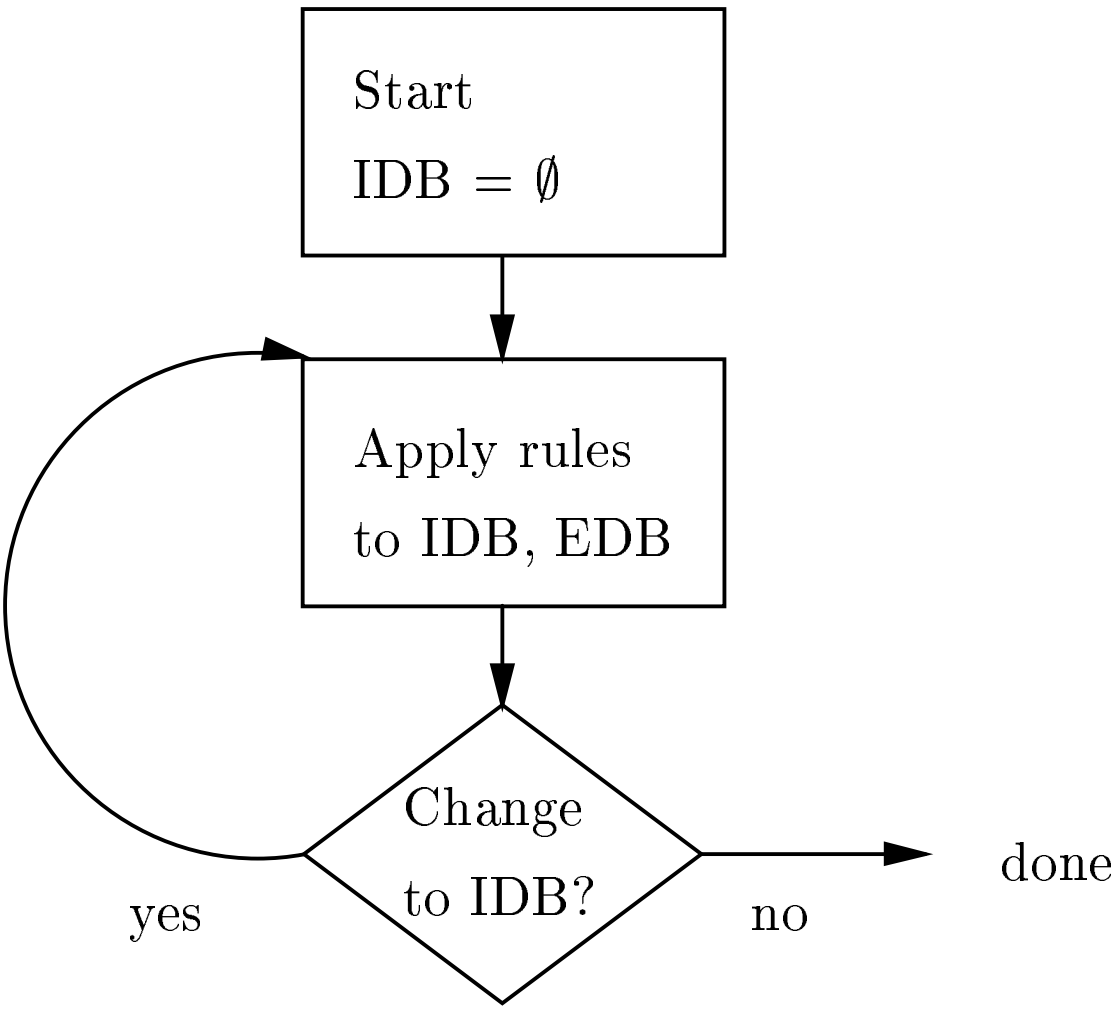
## Recursion

- IDB predicate $P$ *depends* on predicate $Q$ if there is a rule with $P$ in the head and $Q$ in a subgoal.
- Draw a graph: nodes = IDB predicates, arc $P \rightarrow Q$ means $P$ depends on $Q$.
- Cycles iff recursive.

## Recursive Example

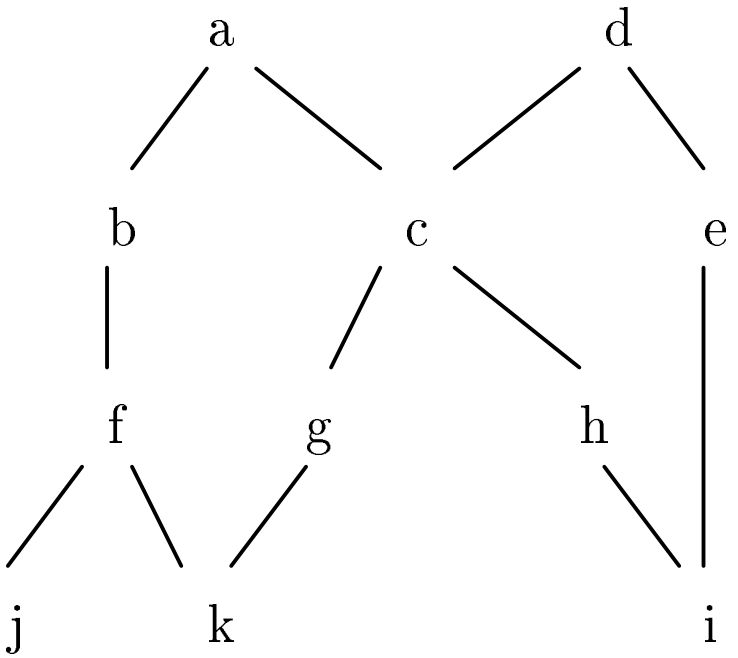
```
Sib(x,y) <- Par(x,p) AND Par(y,p)
    AND x <> y

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

## Iterative Fixed-Point Evaluates Recursive Rules



## Example

EDB `Par` =




- Note, because of symmetry, `Sib` and `Cousin` facts appear in pairs, so we shall mention only $(x, y)$ when both $(x, y)$ and $(y, x)$ are meant.

| | `Sib` | `Cousin` |
|---|---|---|
| Initial | $\emptyset$ | $\emptyset$ |
| Round 1<br>add: | $(b,c),\ (c,e)$<br>$(g,h),\ (j,k)$ | $\emptyset$ |
| Round 2<br>add: | | $(b,c),\ (c,e)$<br>$(g,h),\ (j,k)$ |
| Round 3<br>add: | | $(f,g),\ (f,h)$<br>$(g,i),\ (h,i)$<br>$(i,k)$ |
| Round 4<br>add: | | $(k,k)$<br>$(i,j)$ |