# Deductive Databases

Dr Paolo Guagliardo

dbs-lecturer@ed.ac.uk

THE UNIVERSITY of EDINBURGH
informatics

Fall 2018

## Datalog rules (1)

$$\underbrace{H(\bar{x})}_{\text{head}} :- \underbrace{S_1(\bar{x}_1), \ldots, S_n(\bar{x}_n)}_{\text{body}}$$

**Head** predicate over variables and constants

**Body** conjunction of possibly negated atoms (**subgoals**)

- relational atoms (predicates)
- comparisons between variables/constants

- Head variables are implicitly **universally quantified**
- Body variables not in head are **existentially quantified**
- Rule: Body $\rightarrow$ Head (if the body is true, the head is true)

# Datalog rules (2)

Example

$$\text{Lucky}(x) \text{ :- } \text{Customer}(x, y, z), \text{Account}(u, z, x, w), w > 10000$$

In relational calculus we could write:

$$\text{Lucky} = \{\, x \mid \exists y, z, u, w \ \text{Customer}(x, y, z) \\ \wedge \text{Account}(u, z, x, w) \wedge w > 10000 \,\}$$

Safety

Every variable (in head or body)
appears in at least one non-negated relational atom

**Not safe:** $\text{BigNumber}(x) \text{ :- } x > 1000000000$

# Datalog programs

**Program** = set of Datalog rules

Example

$$\text{Parent}(x, y) \text{ :- } \text{Mother}(x, y)$$
$$\text{Parent}(x, y) \text{ :- } \text{Father}(x, y)$$

edb (extensional database) relations **stored** in the database

- can appear only in the body of rules

idb (intensional database) derived relations

- can appear both in the head or body of rules

# From relational algebra to Datalog

Every relational algebra expression can be translated into Datalog

**Projection** $\pi_{\#2}(R)$

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

**Selection** $\sigma_{\#1\,\textbf{op}\,c}(R)$

$E(x, y) \text{ :- } R(x, y), x\ \textbf{op}\ c$

**Product** $R \times S$

$E(x, y, w, z) \text{ :- } R(x, y), S(w, z)$

**Difference** $R - S$

$E(x, y) \text{ :- } R(x, y), \neg S(x, y)$

**Union** $R \cup S$

$E(x, y) \text{ :- } R(x, y)$
$E(x, y) \text{ :- } S(x, y)$

# From relational algebra to Datalog

Let $R$ and $S$ be relations over $A, B$

$$E = \underbrace{\pi_{A,D}\Big(\underbrace{\sigma_{B=C}\big(\underbrace{R \times \rho_{A\to C,\,B\to D}(S)}_{E_1}\big)}_{E_2}\Big)}_{E_3} \cup \underbrace{\rho_{B\to D}(R - S)}_{E_4}$$

$$\begin{aligned}
E_1(x, y, w, z) &\text{ :- } R(x, y), S(w, z)\\
E_2(x, y, w, z) &\text{ :- } E_1(x, y, w, z), y = w\\
E_3(x, y) &\text{ :- } E_2(x, u, v, y)\\
E_4(x, y) &\text{ :- } R(x, y), \neg S(x, y)\\
E(x, y) &\text{ :- } E_3(x, y)\\
E(x, y) &\text{ :- } E_4(x, y)
\end{aligned}$$

# Limitations of relational algebra/calculus

Parent $=$ table of pairs $x, y$ where $x$ is the parent of $y$

$$\begin{aligned}
\text{Parent} &= \{x, y \mid \text{Parent}(x, y)\} \\
\text{Grandparent} &= \{x, y \mid \exists z\ \text{Parent}(x, z) \wedge \text{Parent}(z, y)\} \\
\text{Great-grandparent} &= \{x, y \mid \exists z\ \text{Grandparent}(x, z) \wedge \text{Parent}(z, y)\}
\end{aligned}$$

For a **given** $k$, we can express the query Ancestor$^k$

But

We **cannot express** the Ancestor relation itself
that is: an Ancestor$^k$ query that works for **every** $k$

# Recursion in Datalog

The head relation of a rule can appear in its body

$$\begin{aligned}
&\text{Ancestor}(x, y) :- \text{Parent}(x, y) \\
&\text{Ancestor}(x, y) :- \text{Ancestor}(x, z), \text{Parent}(z, y)
\end{aligned}$$

Intuition

$x$ is an ancestor of $y$ if

$x$ is a parent of $y$

**or**

$x$ is an ancestor of a parent of $y$

# Dependency graph

IDB predicate $P$ **depends on** (IDB) predicate $Q$
if there is a rule with $P$ in the head and $Q$ in a subgoal

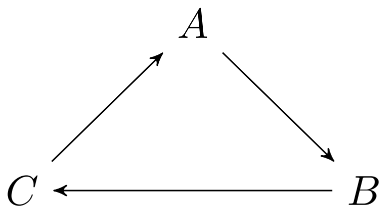
## Dependency graph

nodes IDB predicates

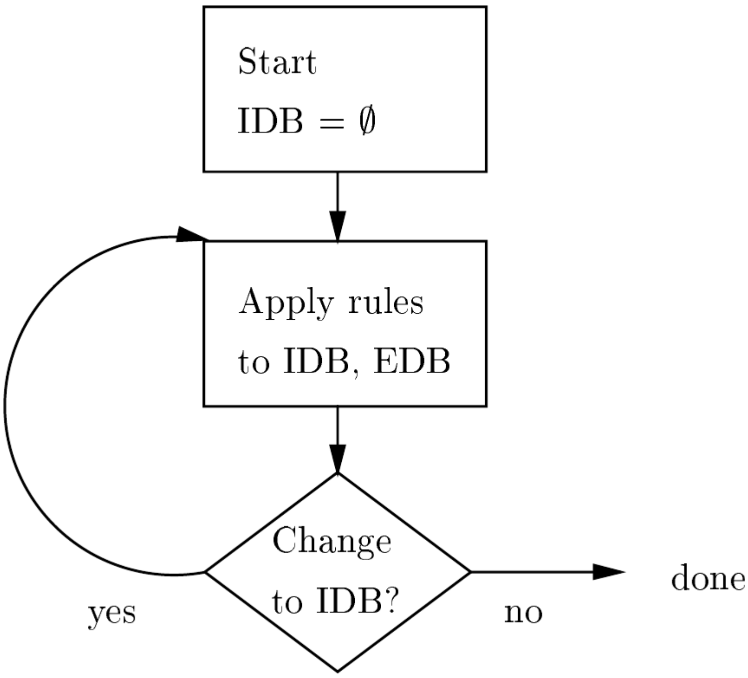
edges $P \to Q$ if $P$ depends on $Q$

A **cycle** in the dependency graph means the program is recursive

$$
\begin{aligned}
C(x) &:- A(y,x) \\
C(x) &:- S(x,y), y > 1 \\
B(x,y) &:- C(x), P(x,y) \\
A(x,y) &:- B(y,x)
\end{aligned}
$$




# Iterative Fixpoint Evaluation



## Evaluation of recursive programs

- The **Parent** relation (EDB) never changes
- The **Ancestor** relation (IDB) is initially empty

  $\text{Ancestor}_0 = \varnothing$

- At step $i + 1$ compute:

  $\text{Ancestor}_{i+1}(x, y) \text{ :- } \text{Parent}(x, y)$
  $\text{Ancestor}_{i+1}(x, y) \text{ :- } \text{Ancestor}_i(x, z), \text{Parent}(z, y)$

- Stop when a **fixpoint** is reached

  $\text{Ancestor}_{i+1} = \text{Ancestor}_i$

## Evaluation of recursive programs

IDB:

| Ancestor | |
|---|---|
| John | Mary |
| John | Jane |
| Jane | Louis |
| Mary | Linda |
| Louis | Mark |
| John | Linda |
| John | Louis |
| Jane | Mark |
| John | Mark |

EDB:

| Parent | |
|---|---|
| John | Mary |
| John | Jane |
| Jane | Louis |
| Mary | Linda |
| Louis | Mark |

$\text{Ancestor}(x, y) \text{ :- } \text{Parent}(x, y)$

$\text{Ancestor}(x, y) \text{ :- } \text{Ancestor}(x, z), \text{Parent}(z, y)$

## Recursion in SQL

Suppose we have a table Parent with attributes name, child

```
WITH RECURSIVE Ancestor(name, descendant) AS (
    SELECT *
    FROM   Parent
    UNION
    SELECT A.name, P.child
    FROM   Ancestor A, Parent P
    WHERE  A.descendant = P.name
)
SELECT * FROM Ancestor ;
```

The definition mimics the structure of the Datalog program

## Nonlinear recursion

The head relation can appear **more than once** in its body

$$\text{Ancestor}(x, y) :- \text{Parent}(x, y)$$
$$\text{Ancestor}(x, y) :- \text{Ancestor}(x, z), \text{Ancestor}(z, y)$$

### Intuition

$x$ is an ancestor of $y$ if

$x$ is a parent of $y$

**or**

$x$ is an **ancestor of an ancestor** of $y$

# Nonlinear recursion in SQL

Not supported

```
WITH RECURSIVE Ancestor(name, descendant) AS (
    SELECT *
    FROM   Parent
    UNION
    SELECT A.name, A2.desc
    FROM   Ancestor A1, Ancestor A2
    WHERE  A1.descendant = A2.name
)
SELECT * FROM Ancestor ;
```

```
ERROR: recursive reference to query "ancestor" must not appear
more than once
```

# Recursive programs with negation

Consider the program $P = \{R(x) :- S(x), \neg R(x)\}$

IDB:

| $R$ |
|---|
| |

EDB:

| $S$ |
|---|
| 1 |
| 2 |

Step 0 IDB relation $R$ is empty

Step 1 $R = \{1, 2\}$

Step 2 $R = \varnothing$

Step 3 $R = \{1, 2\}$

Step 4 $R = \varnothing$

… **No fixpoint!**

Iteration never ends

# Stratification

Partition a program $P$ into a sequence of subprograms $P_1, \ldots, P_n$

- Each subprogram defines one or more IDB relations
- If a relation $S$ is used positively in the definition of $R$ then $S$ must be defined **earlier or simultaneously** with $R$
- If a relation $S$ is used negatively in the definition of $R$ then $S$ must be defined **strictly before** $R$

Stratum 0 EDB relations

Stratum i IDB relations that depend

positively on relations in any stratum $j \leq i$

negatively on relations in any stratum $j < i$

# Stratification

## Stratum graph

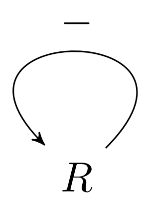
nodes IDB predicates

edges $P \rightarrow Q$ if $P$ depends on $Q$

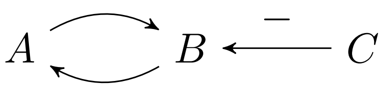
**label the edge with "–" if $Q$ is a negated subgoal**

Stratifiable program: no cycle involving at least one negated edge

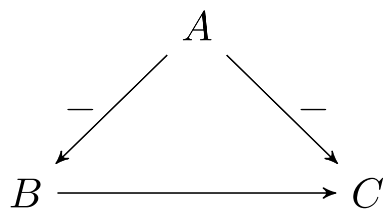
$$R(x) \text{ :– } S(x), \neg R(x)$$



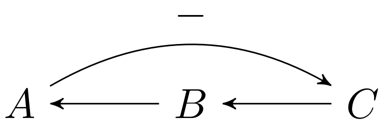
# More examples



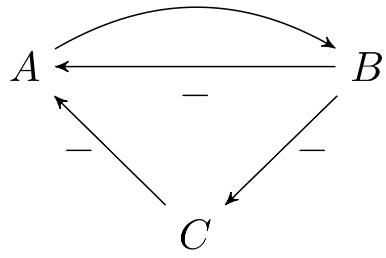

Stratifiable: $A = B < C$




Stratifiable: $C \leq B < A$




**Not stratifiable**:
$A \leq B \leq C < A$

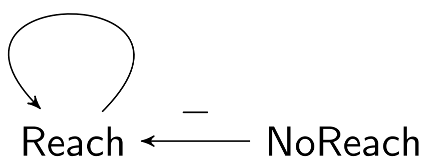



**Not stratifiable**: $A < B \leq A$

# Stratified example

Which target nodes cannot be reached from any source node?

$$\begin{aligned} \text{NoReach}(x) &:- \text{Target}(x), \neg\text{Reach}(x) && \text{(rule1)} \\ \text{Reach}(x) &:- \text{Source}(x) && \text{(rule2)} \\ \text{Reach}(x) &:- \text{Reach}(y), \text{Link}(y, x) && \text{(rule3)} \end{aligned}$$




Stratum 0 Source, Link, Target
Stratum 1 Reach
Stratum 2 NoReach

## Evaluation of stratified programs

$P$ partitioned into a **sequence** $P_1, \ldots, P_n$

Gives us an order in which to apply (each group of) rules

At each iteration $k$, execute each subprogram in sequence

(1) Apply all the rules in $P_1$

(2) Apply all the rules in $P_2$

$\vdots$

($n$) Apply all the rules in $P_n$

## Evaluation of stratified programs

$$
\begin{aligned}
\text{Reach}(x) &:- \text{Source}(x) && (P_1)\\
\text{Reach}(x) &:- \text{Reach}(y), \text{Link}(y, x) && (P_1)\\
\text{NoReach}(x) &:- \text{Target}(x), \neg\text{Reach}(x) && (P_2)
\end{aligned}
$$

EDB

| Source |
|---|
| 1 |
| 2 |

| Link | |
|---|---|
| 1 | 2 |
| 3 | 4 |
| 2 | 4 |

| Target |
|---|
| 2 |
| 3 |
| 4 |

IDB

| Reach |
|---|
| 1 |
| 2 |
| 4 |

| NoReach |
|---|
| 3 |

# Further remarks on SQL recursion

- Requires **stratified negation**
- Only **linear recursion**

**Problems**

Arithmetic operations

introduce new values not present in the database

Multiset semantics

rules must be applied several times

**cycles in the data** must be detected