A crash-course on Datalog (from Basi di Dati1)

prof. Letizia Tanca

Relational Formal Languages

- **Algebra**
- **Calculus**
  - **Tuple calculus**
  - **Domain calculus**
- **Datalog**

Datalog is based on Logic Programming

- A rule-based programming paradigm similar to Prolog (1970)
- **Datalog:** "Prolog for databases" (1984)
- Main differences
  - no function symbols
  - a different computation model (declarative)

A Datalog program is a set of rules

Rule head, or LHS and rule body, or RHS:

$$p :\!\!- p_1, p_2, \ldots p_n$$

also written as

$$p \leftarrow p_1, p_2, \ldots p_n$$

A *p* is a *literal*, or *predicate instance*:

- name of the predicate
- list of its arguments:
  - **constants**
  - **variables**
  - **"don't care" symbol: "_ " (forbidden in the lhs)**

Examples

- Rules are *safe*: every variable in a head must appear in the corresponding body
- A safe Datalog rule:

$$\mathbf{S(x) \leftarrow P(a, x), Q(y, a)}$$

- **a** is a constant (from the database domain), **x, y** are variables
- **P(x, y) ← Q(x, z), S(z, x)** is not safe, why?

A sample database

PARENT

Parent	Child
Carlo	Antonio
Carlo	Gianni
Anna	Antonio
Anna	Gianni
Gianni	Andrea
Antonio	Paola

PERSON

Name	Age	Gender
Carlo	65	M
Antonio	40	M
Anna	60	F
Gianni	43	M
Andrea	22	M
Paola	20	F

Examples

Grandparent(X,Z) ← *Parent* **(***X, Y***),** *Parent(Y, Z)*

Father(X,Y):- Person(X,_,'M'),Parent(X,Y).

- *Parent* and *Person* are base relations, or extensional relations, not defined anywhere in the program
- We are defining the *views Grandparent* and *Father*
- Variables are implicitly universally quantified
- E.g. the first rule is a simplified expression for

$\forall X \forall Y \forall Z$ *(Grandparent(X,Z)* ← *Parent (X, Y), Parent(Y, Z)))*

Datalog rules and DBs

- Each tuple corresponds to a **base fact** (or *ground literal*) :

  Parent("Carlo","Antonio").

  Is the first tuple of the PARENT relation

Interpretation and **Unification**

Rule interpretation of:

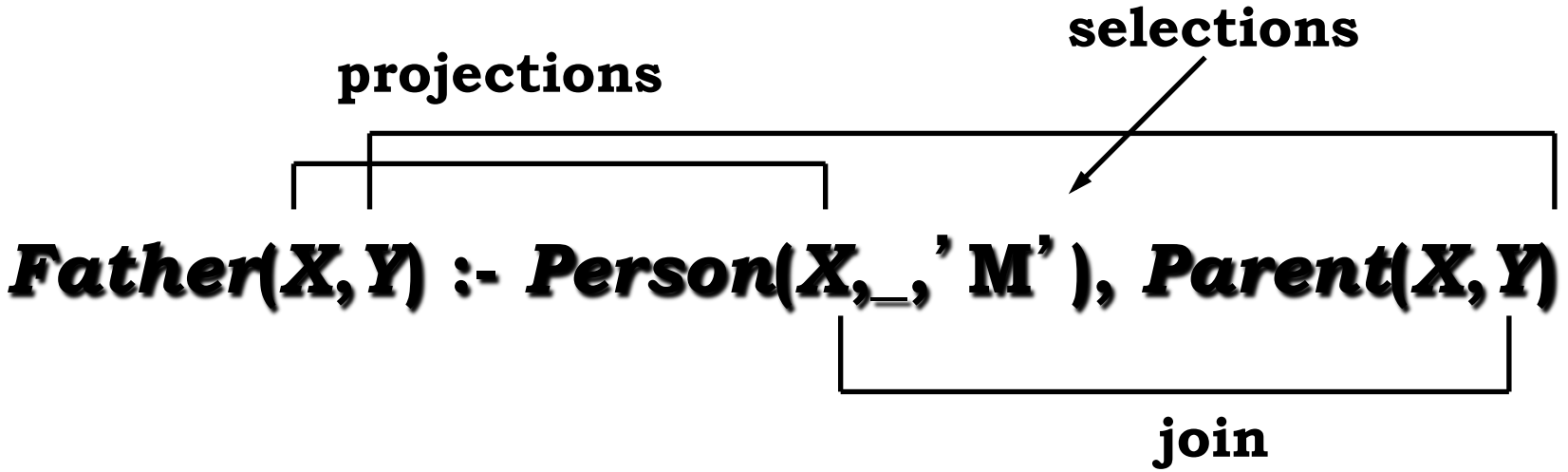
```
Father(X,Y):- Person(X,_,'M'),Parent(X,Y).
```

- *LHS true if RHS is true*
- *RHS is true if, for each literal of RHS, all its variables are unifiable, i.e. replaceable, with constants which make the predicate true*

Person(*X*,_,'M'): possible unifications for *X*:

{"Carlo", "Antonio", "Gianni", "Andrea"}

Corrispondance between Datalog and relational algebra



- In Relational Algebra:

FATHER = $\Pi_{1,5}\ \sigma_{3 = \text{'M'}}$ (PERSON $\triangleright\!\triangleleft_{1 = 1}$ PARENT)

EXTENSIONAL AND INTENSIONAL DATABASE

- Extensional (**EDB**): the DB tables
- Intensional (**IDB**): the set of predicates that appear in the LHS of at least one rule

➢ ***views, or knowledge, inferred from the EDB***

- **by hypothesis, in standard Datalog**

$$EDB \cap IDB = \emptyset$$

Queries in Datalog

Queries are expressed as "***goals***" :

?- parent("Anna",X)

X = "Antonio" or *X="Gianni"*

A goal without variables returns *True* or *False*

?- *parent* ("Anna","Antonio") ⇒ *True*
?- *parent* ("Anna","Andrea") ⇒ *False*

Queries in Datalog (II)

• Queries on EDB + IDB are expressed by ***goals*** as well:

?- Father("Carlo",X)

• Computation: we look for a rule defining Father and for a substitution that unifies with X

We obtain X = *"Antonio"* or X=*"Gianni"*

Also here, a goal without variables returns *True* or *False*

- ?- *Father*("Carlo","Antonio") ⇒ *True*
- ?- *Father*("Carlo","Andrea") ⇒ *False*

Queries in Datalog (III)

To write more complex queries we use the rules:
e.g. “find all Carlo’s brothers”

We define the concept of Brother:

Brother(X,Y) :- Parent (Z,X), Parent(Z,Y), X≠Y.

Then we write the goal:

?- Brother(“Carlo”,X).

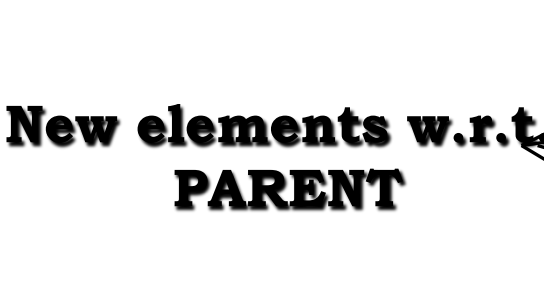
More rules on the same BD

- *Mother*(*X*, *Y*) :- *Person*(*X*,_,'F'), *Parent*(*X*, *Y*).
- *Grandparent*(*X*,*Z*) :- *Parent* (*X*, *Y*), *Parent*(*Y*,*Z*).
- *Uncle* (*X*, *Y*) :- *Person*(*X*,_,'M'), *Brother*(*X*,*Z*), *Parent*(*Z*, *Y*).
- *Brother*(*X*, *Y*) :- *Parent* (*Z*,*X*), *Parent*(*Z*, *Y*), *X*≠*Y*.

NOTE THE FOLLOWING:

- *Ancestor (x, y)* ← *Parent(x, y)*
- *Ancestor (x, z)* ← *Parent(x, y), Ancestor (y, z)*

Recursive queries



ANCESTOR	
Carlo	Antonio
Carlo	Gianni
Anna	Antonio
Anna	Gianni
Gianni	Andrea
Antonio	Paola
Carlo	Andrea
Carlo	Paola
Anna	Andrea
Anna	Paola

Computation of recursive rules

$\mathbf{ANCESTOR^0} \Leftarrow \varnothing$

$\mathbf{ANCESTOR^1} \Leftarrow \mathbf{PARENT}$

$\mathbf{ANCESTOR^2} \Leftarrow \mathbf{[(}\Pi_{1,4}\ \mathbf{(ANCESTOR^1} \triangleright\!\triangleleft_{2=1} \mathbf{PARENT\ )} \cup \mathbf{PARENT]} \cup \mathbf{ANCESTOR^1}$

$\mathbf{ANCESTOR^3} \Leftarrow \mathbf{[}\ \Pi_{1,4}\ \mathbf{(ANCESTOR^2} \triangleright\!\triangleleft_{2=1} \mathbf{PARENT\ )} \cup \mathbf{PARENT]} \cup \mathbf{ANCESTOR^2}$

…… Compute…..

Until $\mathbf{ANCESTOR^n} = \mathbf{ANCESTOR^{n-1}}$ (**fixpoint**)

Uses of Datalog

Constraints:

incorrectdb1(X,Y) :- Mother(X,Z), Mother(Y,Z), X≠Y.
incorrectdb2(X) :- Ancestor(X,X).
incorrectdb3( ) :- Parent(X,Z),¬Person(X,_,_).

Note: by including the variables in the rule heads we can also find the objects that violate the constraint!

Datalog terminology

Relation	→	***Predicate***
Attribute	→	***Argument***
Tuple	→	***Fact***
View	→	***Rule***
Query	→	***Goal + rules***

Important notes:

- *Attributes do not have names!*
- *We did not speak about difference and negation*