# Basi di Dati
## Datalog

Michele Beretta

michele.beretta@unibg.it

## Introduzione

- Programmazione logica, come Prolog – basato su *regole*
- Linguaggio dichiarativo
- Sintassi minimale, ricorsione – più espressivo di algebra e calcolo

Fare attenzione al cambio di terminologia:

- Relazione $\rightarrow$ Predicato
- Attributo $\rightarrow$ Argomento
- Tupla $\rightarrow$ Fatto
- Query $\rightarrow$ Regola
- Interrogazione $\rightarrow$ Goal

Fare attenzione alla negazione, dev'essere **safe**:

- Una regola è safe se ogni variabile compare almeno una volta positivamente
- Fare attenzione a ricorsione e negazione

## Esercizi I

```
genitore(Padre, Figlio).
```

1. Ricavare tutti i discendenti di ogni padre
2. Ricavare tutti i discendenti di Bob
3. Ricavare tutti i discendenti di Bob ma non di Alice

## Esercizi II

```
amico(Persona, Persona).
nemico(Persona, Persona).
```

1. Trovare tutti gli amici e gli amici degli amici di Bob
2. Trovare tutti gli amici di Bob che hanno solo Bob come amico
3. Trovare tutte le persone che hanno come amici i nemici dei loro nemici

## Esercizio I.1

Ricavare tutti i discendenti di ogni padre.

```
disc(X, Y) :- genitore(X, Y).
disc(X, Y) :- disc(X, T), genitore(T, Y).
```

## Esercizio I.2

Ricavare tutti i discendenti di Bob.

```
disc_bob(X) :- genitore("Bob", X).
disc_bob(X) :- disc_bob(T), genitore(T, X).
```

## Esercizio I.3

Ricavare tutti i discendenti di Bob ma non di Alice.

```
disc_alice(X) :- genitore("Alice", X).
disc_alice(X) :- disc_alice(T), genitore(T, X).

disc_bob_non_alice(X) :- genitore("Bob", X), not disc_alice(X).
disc_bob_non_alice(X) :- disc_bob_non_alice(T), genitore(T, X), not disc_alice(X).
```

La seguente invece è *sbagliata*:

```
disc_bob_non_alice(X, Y) :- genitore("Bob", Y).
disc_bob_non_alice(X, Z) :- disc_bob_non_alice(X, Y), not genitore("Alice", Z).
```

In quanto la seconda regola non è safe, $Z$ compare solo in negazioni.

## Esercizio II.1

```
r(x) :- amico("Bob", x).
r(x) :- amico("Bob", z), amico(z, x).
```

## Esercizio II.2

Trovare tutti gli amici di Bob che hanno solo Bob come amico.

```
amico_bob(x) :- amico("Bob", x).
esclusi(x) :- amico_bob(x), amico(x, y), y != "Bob".
r(x) :- amico_bob(x), not esclusi(x).
```

## Esercizio II.3

Nella soluzione includiamo anche tutte le persone che non hanno né amici né nemici.

**Interpretazione**: se vediamo `amico("Bob", "Mario")`, supponiamo che Bob ritenga che Mario sia suo amico, ma che il contrario non sia necessariamente vero (i.e., se Mario ritiene Bob suo amico). Similmente per `nemico`.

**Parte 1**: troviamo tutte le persone che *non hanno amici*:

```
amici_sx(x) :- amico(x, y).
tutti_a(x) :- amico(x, y).
tutti_a(x) :- amico(y, x).
amici_dx(x) :-  tutti_a(x), not amici_sx(x).
```

**Parte 2**: troviamo tutte le persone che *non hanno nemici*:

```
nemici_sx(x) :- nemico(x, y).
tutti_b(x) :- nemico(x, y).
tutti_b(x) :- nemico(y, x).
nemici_dx(x) :-  tutti_b(x), not nemici_sx(x).
```

Risultato:

```
r(x) :- nemico(x, y), nemico(y, z), amico(x, z).
r(x) :- amici_dx(x), nemici_dx(x).
```

## Esercizi III

```
PassoProduzione(ProdottoComposto, ProdottoComponente, Qta)
```

Costruire in Datalog una vista `ProdottoBasi(Prodotto, ProdottoBase)` che descrive per ogni prodotto quali sono i prodotti di base (non derivati) che contribuiscono alla sua realizzazione.

```
ProdottoBasi(X, B) :- PassoProduzione(X, B, _), not PassoProduzione(B, _, _).
ProdottoBasi(X, B) :- PassoProduzione(X, Y, _), ProdottoBasi(Y, B).
```