# IKI30820 – Logic Programming
# Database Manipulation and Collecting Solutions
# Slide 08

Ari Saptawijaya (slides)  Adila A. Krisnadhi (LaTeXadaptation)

Fakultas Ilmu Komputer
Universitas Indonesia

2009/2010 • Semester Gasal

# Outline

# Database Manipulation

- In the relational model of databases, a database is a specification of a set of relations.
- A Prolog program can be viewed as such a database: the specification of relations is partly explicit (in the form of facts) and partly implicit (in the form of rules).
- Some built-in predicates make it possible to update this database during the execution of the program.
  - This is done by adding new clauses or by deleting existing clauses.
  - The predicates are: **`assert`**, **`asserta`**, **`assertz`**, **`retract`**, **`retractall`**.

# Built-in Predicates

- **`retract`**`(+Term)`
  When `Term` is an atom or a term it is unified with the first unifying fact or clause in the database. The fact or clause is removed from the database.
- **`retractall`**`(+Head)`
  All facts or clauses in the database for which the head unifies with `Head` are removed.
- **`assert`**`(+Term)`
  Assert a fact or clause in the database. `Term` is asserted as the last fact or clause of the corresponding predicate.
- **`asserta`**`(+Term)`
  Equivalent to **`assert`**`/1`, but `Term` is asserted as first clause or fact of the predicate.
- **`assertz`**`(+Term)`
  Equivalent to **`assert`**`/1`.
- `dynamic +`**`Name`**`/+Arity, ...`
  Informs the interpreter that the definition of the predicate(s) may change during execution (using **`assert`**`/1` and/or **`retract`**`/1`).

# Examples I

```
?- happy(ari).

?- dynamic happy/1.

?- happy(ari).

?- assert(happy(ari)).

?- happy(ari).

?- happy(denny).

?- asserta(happy(denny)).

?- happy(X).
```

# Examples II

```
?- retract(happy(X)).

?- happy(ari).

?- happy(denny).

% Use 'assert' to make a table of product mod 7
maketable :-
  L = [0,1,2,3,4,5,6],
  member(X, L),
  member(Y, L),
  A is X*Y mod 7,
  assert( productMOD7(X,Y,A) ),
  fail.
```

# Outline

# Collecting solutions

```
age(peter,7).
age(ann,5).
age(pat,8).
age(tom,5).

?- age(Child,5).
```

All the objects that satisfy some goal can be collected into a list.

# bagof/3

`bagof(+`**`Var`**`, +Goal, -Bag)`

Unify `Bag` with the alternatives of **`Var`**. If `Goal` has free variables besides the one sharing with **`Var`**, `bagof` will backtrack over the alternatives of these free variables, unifying `Bag` with the corresponding alternatives of **`Var`**.

- The construct `+`**`Var`**`^Goal` tells `bagof` not to bind **`Var`** in `Goal`.
- `bagof/3` fails if `Goal` has no solutions.

# Example

```
?- bagof(Child, age(Child, 5), List).

?- bagof(childrenOf5(X), age(X, 5), List).

?- bagof(Child, age(Child, Age), List).

?- bagof(Child, Age^age(Child,Age), List).
```

# **findall**/3

**findall**(+**Var**, +Goal, -Bag)

Creates a list of the instantiations which **Var** gets successively on backtracking over Goal and unifies the result with Bag.

- Succeeds with an empty list if Goal has no solutions.
- **findall**/3 is equivalent to bagof/3 with all free variables bound with the existence operator (^), except that bagof/3 fails when Goal has no solutions.

# Example

```
?- findall(Child, age(Child, Age), List).

?- findall(Baby,age(Baby,1),L).

?- bagof(Baby,age(Baby,1),L).
```

# setof/3

`setof(+`**`Var`**`, +Goal, -Set)`

Equivalent to `bagof/3`, but sorts the result to get a sorted list of alternatives without duplicates.
Examples:

```
?- setof(Child, Age^age(Child,Age), List).

?- setof(Age/Child, age(Child,Age), List).

?- setof(Age,Child^age(Child,Age),L).

?- bagof(Age,Child^age(Child,Age),L).
```