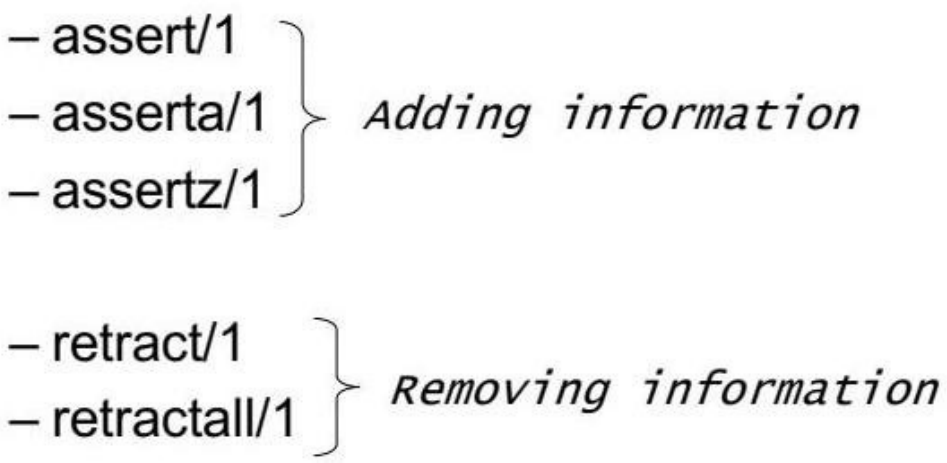
Author : Melissa Boutlendj

Lecture 11 : Manipulation de bases de données et collecte de solutions

Manipulation de la base de données

Prolog dispose de cinq commandes de base pour la manipulation de la base de données, qui permettent d'ajouter et de supprimer des informations pendant l'exécution du programme.



Le diagramme ci-dessus illustre la gestion des informations à travers deux opérations principales : l'ajout et la suppression.

Commandes pour ajouter des informations

- assert/1 : Ajoute une clause à la base de données.
- asserta/1 : Ajoute une clause au début de la base de données.
- assertz/1 : Ajoute une clause à la fin de la base de données.

Commandes pour supprimer des informations

- retract/1 : Supprime une clause de la base de données.
- retractall/1 : Supprime toutes les clauses correspondantes de la base de données.

Exemples d'utilisation d'assert/1

Initialement, la base de données est vide.

```
?- listing.
yes
```

Nous pouvons ajouter des faits à la base de données en utilisant `assert/1`.

```
?- assert(heureux(mia)).
yes
```

```
?- assert(happy(mia)).
yes
?- listing.
happy(mia).
?- assert(happy(vincent)),
    assert(happy(marsellus)),
    assert(happy(butch)),
    assert(happy(vincent)).
```

Après avoir ajouté plusieurs faits, la base de données contient maintenant ces informations.

Prédicats dynamiques et statiques

> Les prédicats ordinaires sont parfois appelés prédicats statiques. Les prédicats dont le sens change pendant l'exécution sont appelés prédicats dynamiques.

Il est parfois nécessaire de déclarer les prédicats dynamiques dans certains interpréteurs Prolog.

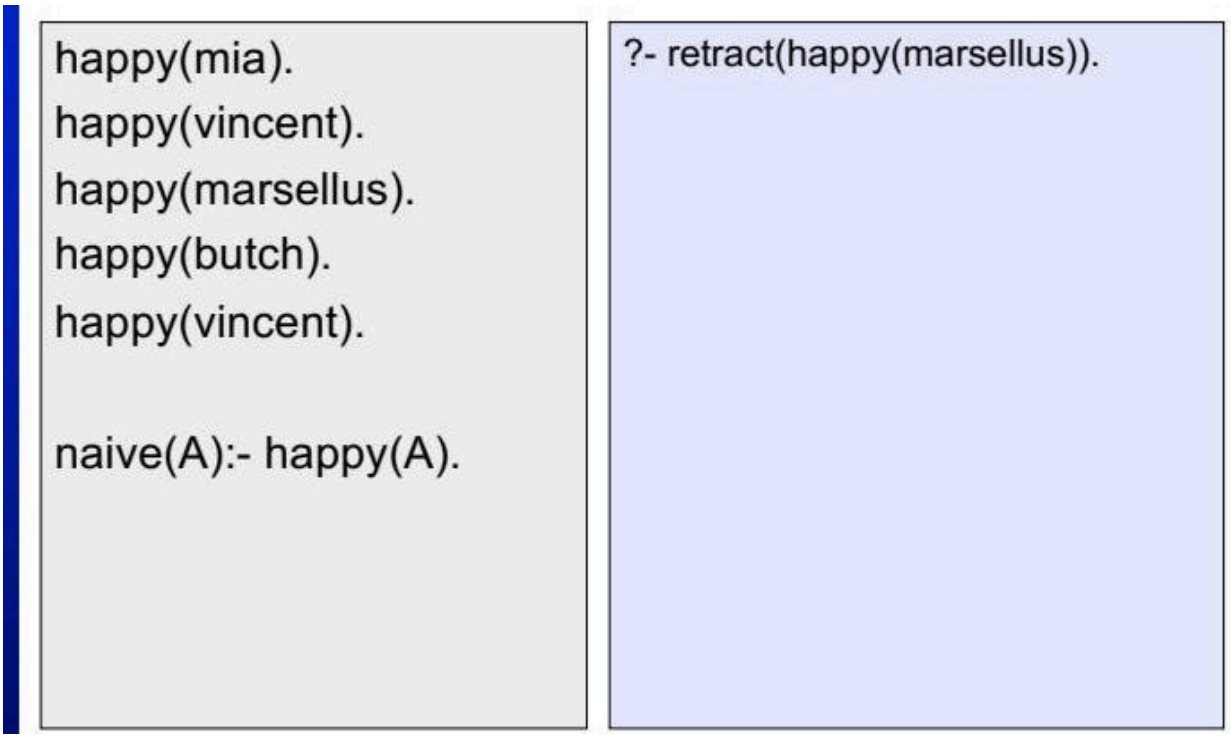
Assertion de règles

On peut également ajouter des règles à la base de données.

```
?- assert(naif(X) :- heureux(X)).
yes
```

Suppression d'informations avec retract/1

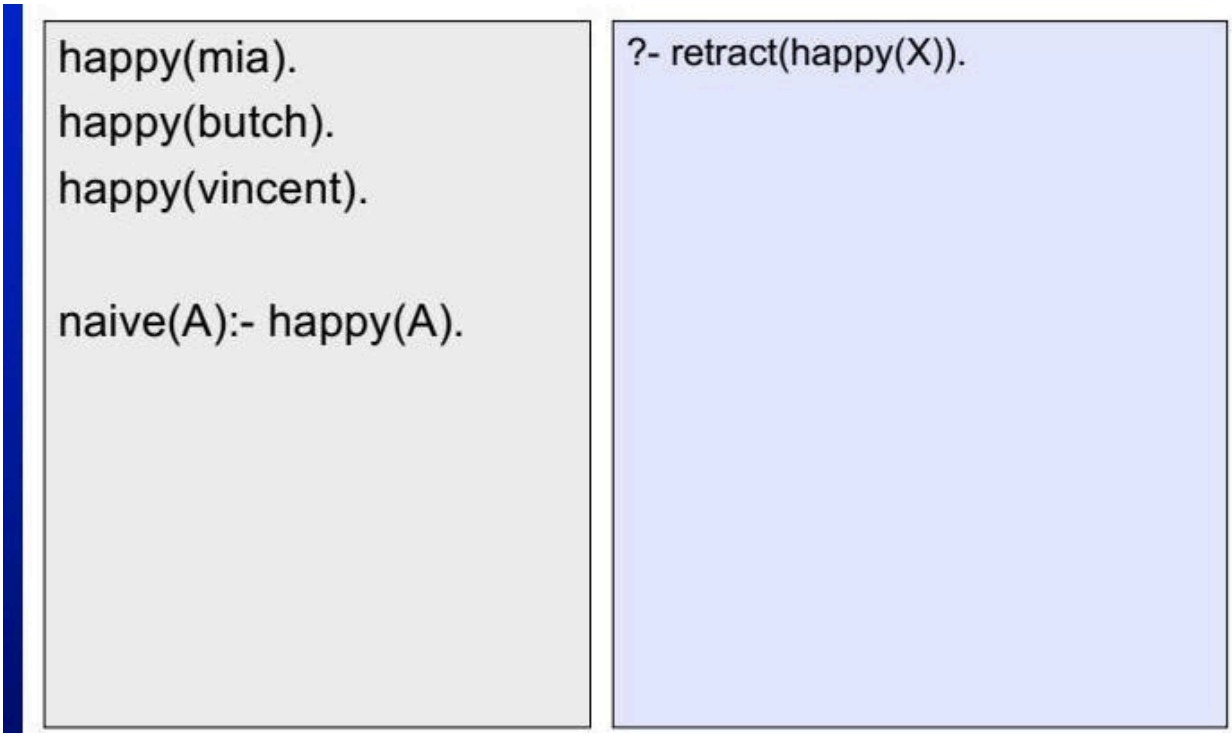
La commande `retract/1` permet de supprimer des informations de la base de données.



Exemple :

```
?- retract(heureux(marsellus)).
yes
```

On peut également utiliser une variable pour supprimer des faits.




```
?- retract(heureux(X)).
X = mia ;
X = butch ;
X = vincent ;
no
```

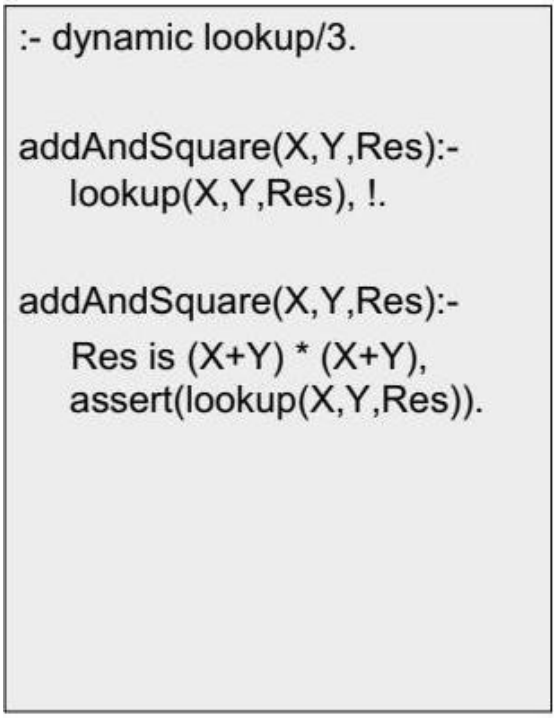
```
?- retract(happy(X)).
X=mia;
X=butch;
X=vincent;
no
?-
```

Utilisation de asserta/1 et assertz/1

- asserta/1 : place le matériel asserté au début de la base de données.
- assertz/1 : place le matériel asserté à la fin de la base de données.

Mémorisat ion

La mémorisation (ou "caching") est une technique utile pour stocker les résultats de calculs, au cas où la même requête doit être recalculée.

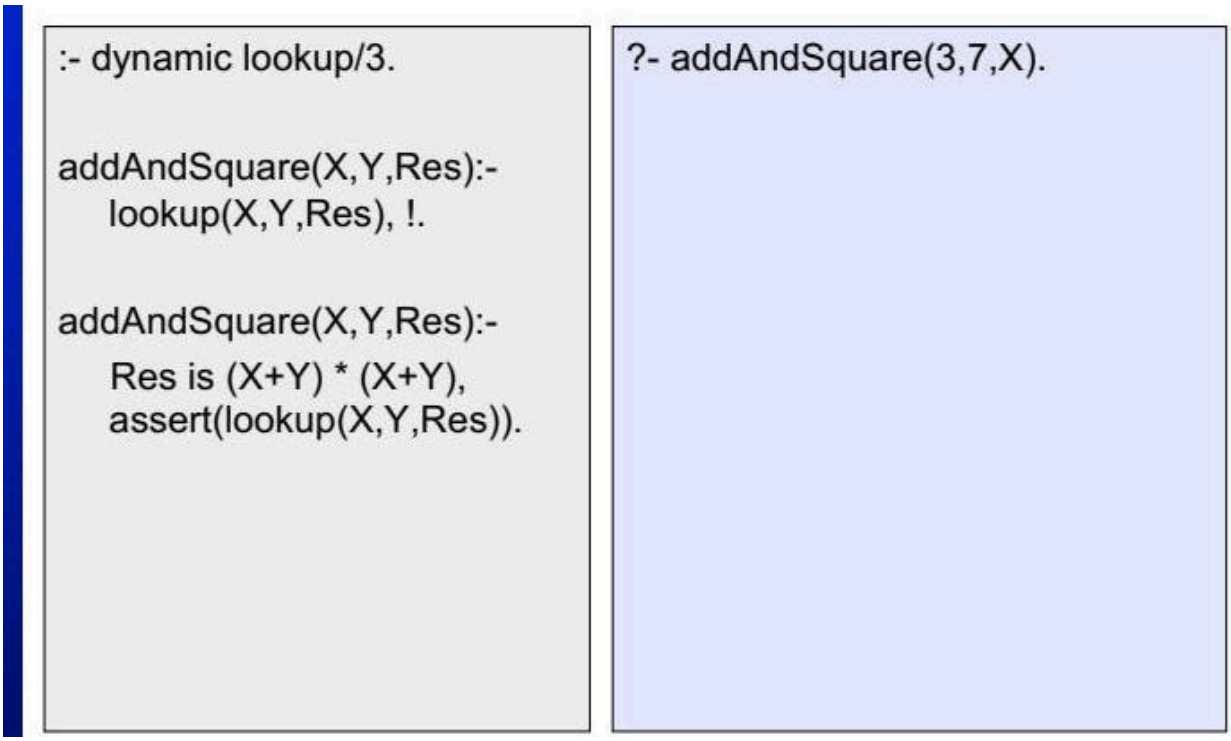


Exemple de code Prolog utilisant la mémorisation :

```
:- dynamic lookup/3.

addAndSquare(X, Y, Res) :-
    lookup(X, Y, Res), !.
addAndSquare(X, Y, Res) :-
    Res is (X+Y) * (X+Y),
    assert(lookup(X, Y, Res)).
```

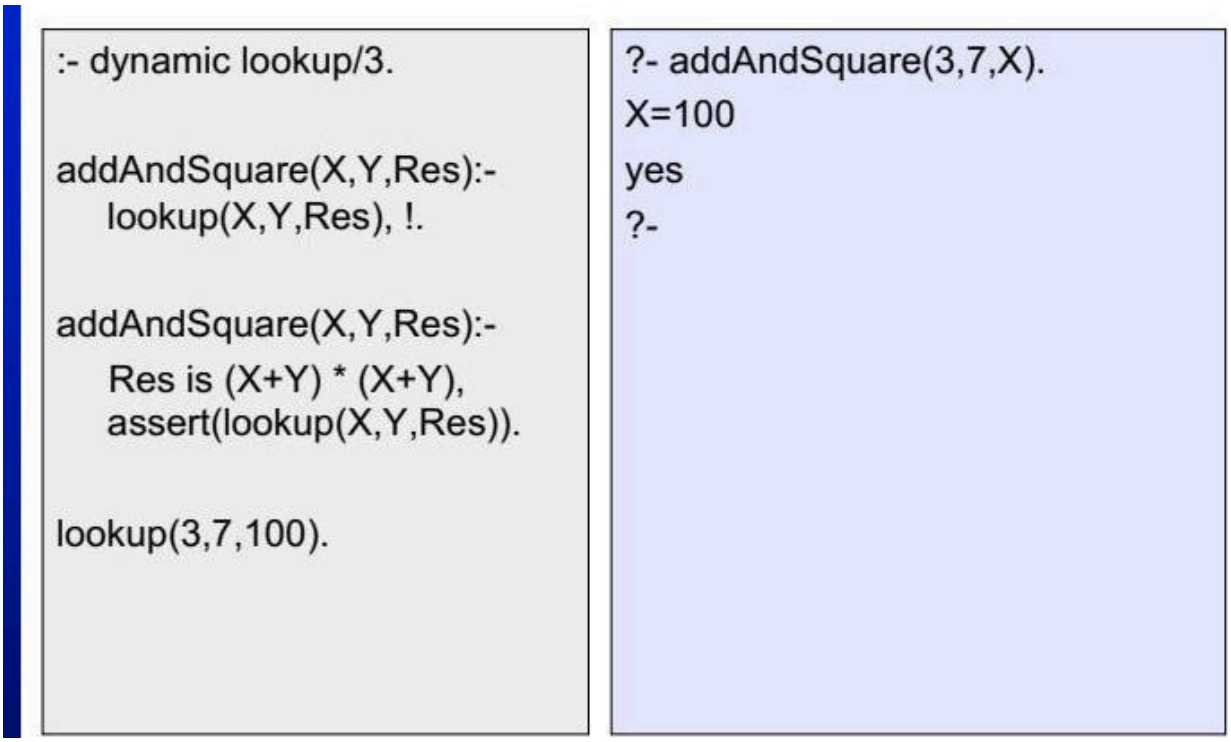
Dans cet exemple, lookup/3 est un prédicat dynamique utilisé pour stocker les résultats de la fonction addAndSquare. Si le résultat est déjà stocké, il est renvoyé directement ; sinon, il est calculé, stocké et renvoyé.



Exemple d'utilisation :

```
?- addAndSquare(3, 7, X).
X = 100
yes

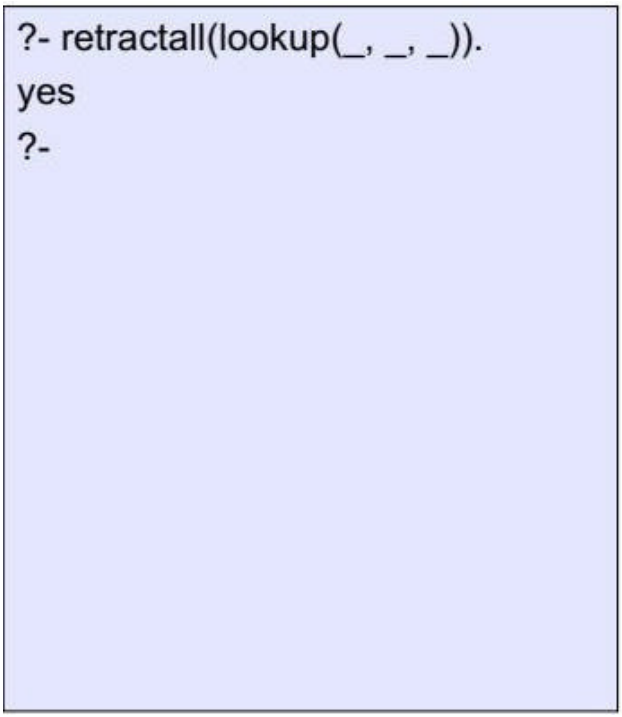
?- lookup(3, 7, 100).
yes
```



Après avoir calculé `addAndSquare(3, 7, X)`, le résultat est stocké dans `lookup/3`.

Utilisation de retractall/1

La commande `retractall/1` permet de supprimer toutes les clauses correspondantes de la base de données.




```
?- retractall(lookup(_, _, _)).
yes
```

Cette commande supprime tous les faits `lookup/3` de la base de données.

Coupes rouges et coupes vertes

Il existe deux types de coupes en Prolog :

- Coupe rouge : Modifie le comportement du programme en empêchant la recherche d'autres solutions.
- Coupe verte : Améliore l'efficacité du programme sans modifier son comportement.

Avertissement sur la manipulation de la base de données

La manipulation de la base de données peut rendre le code difficile à comprendre et non déclaratif. Elle doit être utilisée avec prudence.

Collecte de solutions

Prolog dispose de trois prédicats intégrés pour collecter toutes les solutions à une requête dans une seule liste : findall/3, bagof/3 et setof/3. Il existe des différences importantes entre eux.

findall /3

> La requête findall(O, G, L) produit une liste L de tous les objets O qui satisfont le but G. Elle réussit toujours et unifie L avec une liste vide si G ne peut pas être satisfait.

Exemple :

```
child(martha, charlotte).
child(charlotte, caroline).
child(caroline, laura).
child(laura, rose).

descend(X, Y) :- child(X, Y).
descend(X, Y) :- child(X, Z), descend(Z, Y).

?- findall(X, descend(martha, X), L).
L = [charlotte, caroline, laura, rose]
yes
```

Dans cet exemple, `findall/3` collecte tous les descendants de Martha dans une liste `L` .

Autres exemples :

```
?- findall(f:X, descend(martha, X), L).
L = [f:charlotte, f:caroline, f:laura, f:rose]
yes

?- findall(X, descend(rose, X), L).
L = []
yes

?- findall(d, descend(martha, X), L).
L = [d, d, d, d]
yes
```

L'exemple suivant montre que `findall/3` peut être assez brut car il ne gère pas les variables libres :

```
?- findall(Chi, descend(Mot, Chi), L).
L = [charlotte, caroline, laura, rose, caroline, laura, rose, laura, rose, rose]
yes
```

bagof/3

> La requête bagof(O, G, L) produit une liste L de tous les objets O qui satisfont le but G. Elle ne réussit que si le but G réussit et lie les variables libres dans G.

 Exemple d'utilisation de bagof/3

Exemple :

```
?- bagof(Chi, descend(Mot, Chi), L).
Mot = caroline
L = [laura, rose] ;

Mot = charlotte
L = [caroline, laura, rose] ;

Mot = laura
L = [rose] ;

Mot = martha
L = [charlotte, caroline, laura, rose] ;

no
```

Pour éviter de lier les variables libres, on peut utiliser ^ .

```
?- bagof(Chi, Mot^descend(Mot, Chi), L).
L = [charlotte, caroline, laura, rose, caroline, laura, rose, laura, rose, rose]
```

setof/3

> La requête setof(O, G, L) produit une liste triée L de tous les objets O qui satisfont le but G. Elle ne réussit que si le but G réussit, lie les variables libres dans G, supprime les doublons de L et trie les réponses dans L.

```
?- bagof(Chi,Mot^descend(Mot,Chi),L).
L=[charlotte, caroline, laura, rose,
    caroline, laura, rose, laura, rose,
    rose]
yes

?-
```

Exemple :

```
?- setof(Chi, Mot^descend(Mot, Chi), L).
L = [caroline, charlotte, laura, rose]
yes
```<script type="text/javascript">
window.MathJax = {
      tex: {
        inlineMath: [['$', '$'], ['\\(', '\\)']],
        displayMath: [['$$', '$$'], ['\\[', '\\]']]
      }
 };
</script>
<script type="text/javascript" async
src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-chtml.js">
</script>

Références :

SWI-Prolog. (s. d.). https://www.swi-prolog.org/
Learn ProLog Now ! (s. d.). https://lpn.swi-prolog.org/lpnpage.php?pageid=top
PROLOG – Programmation Logique
© Patrick Blackburn, Johan Bos & Kristina Striegnitz
Traduction : Yannick Estève
Cours dispensé à CERI – Avignon Université