Aula 10: Manipulação de Base de dados e coleta de soluções

- Teoria
  - Discutir manipulação da base de dados em Prolog
  - Discutir predicados pré-construídos que colecionam todas as soluções de um problema em uma única lista

Manipulação da base de dados

- Prolog possui cinco comandos básicos para manipulação da base de dados:
  - assert/1
  - asserta/1
  - assertz/1
  - retract/1
  - retractall/1

Manipulação da base de dados

- Prolog possui cinco comandos básicos para a manipulação da base de dados:

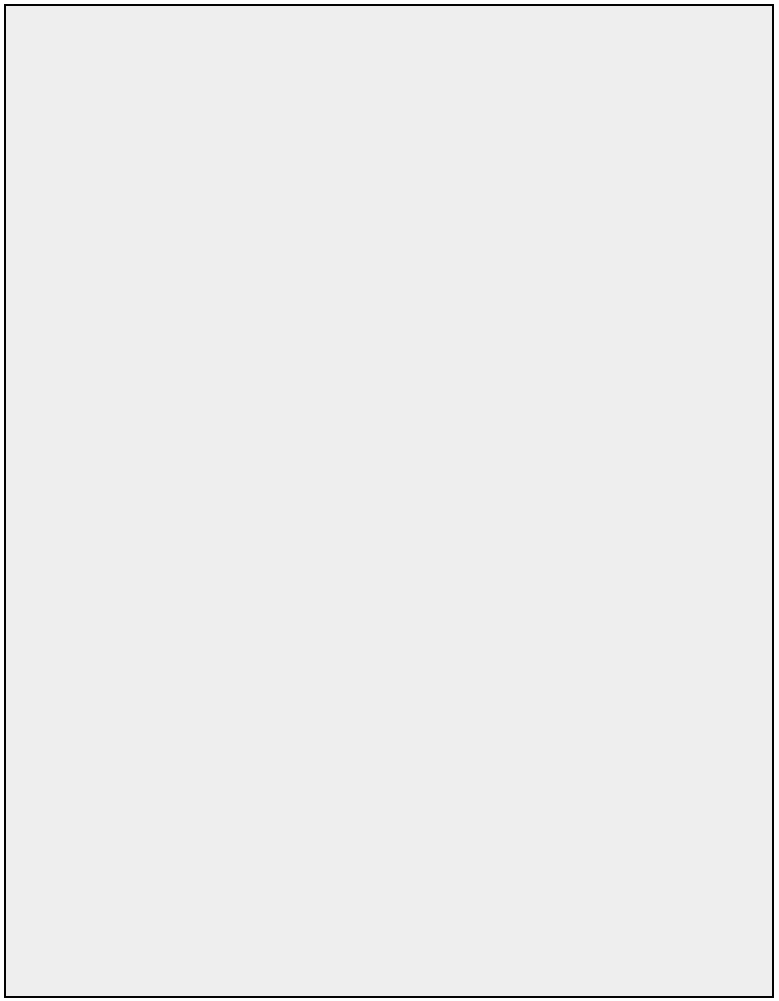
  - assert/1
  - asserta/1
  - assertz/1

  } *Adicionar informação*

  - retract/1
  - retractall/1

  } *Remover informação*

Inicie com uma base de dados vazia



Inicie com uma base de dados vazia

?- listing.

true

Using assert/1

?- assert(feliz(maria)).

true

Using assert/1

```
feliz(maria).
```

```
?- assert(feliz(maria)).
true
?-
```

Using assert/1

```
feliz(maria).
```

```
?- assert(feliz(maria)).
true
?- listing.
feliz(maria).
?-
```

Using assert/1

```
feliz(maria).
```

```
?- assert(feliz(maria)).
true
?- listing.
feliz(maria).
?- assert(feliz(vicente)),
     assert(feliz(marcelo)),
     assert(feliz(bruno)),
     assert(feliz(vicente)).
```

Using assert/1

```
feliz(maria).
feliz(vicente).
feliz(marcelo).
feliz(bruno).
feliz(vicente).
```

```
?- assert(feliz(maria)).
true
?- listing.
feliz(maria).
?- assert(feliz(vicente)),
     assert(feliz(marcelo)),
     assert(feliz(bruno)),
     assert(feliz(vicente)).
true
?-
```

Alterando o significado de predicados

- As manipulações na base de dados alteraram o significado do predicado **feliz/1**
- Mais genericamente:
  - Os comandos de manipulação da base de dados nos dão a habilidade de alterar o significado dos predicados em tempo de execução.

Predicados estáticos e dinâmicos

- Predicados cujo o significado se altera durante o tempo de execução são chamados de predicados **dinâmicos**
  - feliz/1 é um predicado dinâmico
  - Alguns interpretadores Prolog necessitam de uma declaração de predicados dinâmicos.
- Predicados comuns são algumas vezes chamados de predicados **estáticos**.

Inserindo regras

```
feliz(maria).
feliz(vicente).
feliz(marcelo).
feliz(bruno).
feliz(vicente).
```

```
?- assert( (ingenuo(X):- feliz(X)).
```

Inserindo regras

```
feliz(maria).
feliz(vicente).
feliz(marcelo).
feliz(bruno).
feliz(vicente).

ingenuo(A):- feliz(A).
```

```
?- assert( (ingenuo(X):- feliz(X)).
true
?-
```

Removendo informação

- Agora sabemos como adicionar informação à base de dados do Prolog
  - Fazemos isto como o predicado **assert**/1
- Como nós removemos informação?

Removendo informação

- Agora sabemos como adicionar informação à base de dados do Prolog
  - Fazemos isto como o predicado **assert**/1
- Como nós removemos informação?
  - Fazemos isto como o predicado **retract**/1 que removerá uma cláusula
  - Podemos remover muitas cláusulas simultaneamente com o predicado **retractall**/1

Usando retract/1

```
feliz(maria).
feliz(vicente).
feliz(marcelo).
feliz(bruno).
feliz(vicente).

ingenuo(A):- feliz(A).
```

```
?- retract(feliz(marcelo)).
```

Usando retract/1

```
feliz(maria).
feliz(vicente).
feliz(bruno).
feliz(vicente).

ingenuo(A):- feliz(A).
```

```
?- retract(feliz(marcelo)).
true
?-
```

Usando retract/1

```
feliz(maria).
feliz(vicente).
feliz(bruno).
feliz(vicente).

ingenuo(A):- feliz(A).
```

```
?- retract(feliz(marcelo)).
true
?- retract(feliz(vicente)).
```

Usando retract/1

```
feliz(maria).
feliz(bruno).
feliz(vicente).

ingenuo(A):- feliz(A).
```

```
?- retract(feliz(marcelo)).
true
?- retract(feliz(vicente)).
true
```

Usando retract/1

```
feliz(maria).
feliz(bruno).
feliz(vicente).

ingenuo(A):- feliz(A).
```

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

Usando retract/1

```
feliz(bruno).
feliz(vicente).

ingenuo(A):- feliz(A).
```

```
?- retract(feliz(X)).
X=maria;
```

Usando retract/1

```
feliz(vicente).

ingenuo(A):- feliz(A).
```

```
?- retract(feliz(X)).
X=maria;
X=bruno;
```

Usando retract/1

```
ingenuo(A):- feliz(A).
```

```
?- retract(feliz(X)).
X=maria;
X=bruno;
X=vicente;
```

Usando retract/1

```
ingenuo(A):- feliz(A).
```

```
?- retract(feliz(X)).
X=maria;
X=bruno;
X=vicente;
false
?-
```

Usando asserta/1 e assertz/1

- Se quisermos mais controle sobre onde posicionar o material inserido, podemos usar os variantes de assert/1:
  - **asserta**/1
    posiciona o material a ser inserido no **início** da base de dados
  - **assertz**/1
    posiciona o material a ser inserido no **fim** da base de dados

Memorisação

- A manipulação da base de dados é uma técnica útil
- Ela é especialmente útil para armazenar resultados de cálculos, para o caso de precisarmos recalcular a mesma consulta.
- Isto é normalmente chamado de **memorisação**

Exemplo de memorisação

```
:- dynamic consulta/3.

somaAoQuadrado(X,Y,Res):-
    consulta(X,Y,Res), !.

somaAoQuadrado(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(consulta(X,Y,Res)).
```

Exemplo de memorisação

```
:- dynamic consulta/3.

somaAoQuadrado(X,Y,Res):-
    consulta(X,Y,Res), !.

somaAoQuadrado(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(consulta(X,Y,Res)).
```

```
?- somaAoQuadrado(3,7,X).
```

Exemplo de memorisação

```
:- dynamic consulta/3.

somaAoQuadrado(X,Y,Res):-
    consulta(X,Y,Res), !.

somaAoQuadrado(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(consulta(X,Y,Res)).

consulta(3,7,100).
```

```
?- somaAoQuadrado(3,7,X).
X=100
true
?-
```

Exemplo de memorisação

```
:- dynamic consulta/3.

somaAoQuadrado(X,Y,Res):-
    consulta(X,Y,Res), !.

somaAoQuadrado(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(consulta(X,Y,Res)).

consulta(3,7,100).
```

```
?- somaAoQuadrado(3,7,X).
X=100
true
?- somaAoQuadrado(3,4,X).
```

Exemplo de memorisação

```
:- dynamic consulta/3.

somaAoQuadrado(X,Y,Res):-
    consulta(X,Y,Res), !.

somaAoQuadrado(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(consulta(X,Y,Res)).

consulta(3,7,100).
consulta(3,4,49).
```

```
?- somaAoQuadrado(3,7,X).
X=100
true
?- somaAoQuadrado(3,4,X).
X=49
true
```

Usando retractall/1

```
:- dynamic consulta/3.

somaAoQuadrado(X,Y,Res):-
    consulta(X,Y,Res), !.

somaAoQuadrado(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(consulta(X,Y,Res)).

consulta(3,7,100).
consulta(3,4,49).
```

```
?- retractall(consulta(_, _, _)).
```

Usando retractall/1

```
:- dynamic consulta/3.

somaAoQuadrado(X,Y,Res):-
    consulta(X,Y,Res), !.

somaAoQuadrado(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(consulta(X,Y,Res)).
```

```
?- retractall(consulta(_, _, _)).
true
?-
```

Cortes verdes e vermelhos

Corte vermelho

```
:- dynamic consulta/3.

somaAoQuadrado(X,Y,Res):-
    consulta(X,Y,Res), !.

somaAoQuadrado(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(consulta(X,Y,Res)).
```

Cortes verdes e vermelhos

Corte vermelho

```
:- dynamic consulta/3.

somaAoQuadrado(X,Y,Res):-
    consulta(X,Y,Res), !.

somaAoQuadrado(X,Y,Res):-
    Res is (X+Y) * (X+Y),
    assert(consulta(X,Y,Res)).
```

Cortes verdes

```
:- dynamic consulta/3.

somaAoQuadrado(X,Y,Res):-
    consulta(X,Y,Res), !.

somaAoQuadrado(X,Y,Res):-
    \+ consulta(X,Y,Res), !,
    Res is (X+Y) * (X+Y),
    assert(consulta(X,Y,Res)).
```

Uma palavra de aviso…

- Uma palavra de aviso sobre a manipulação da base de dados:
  - Normalmente é uma técnica útil
  - Mas pode levar a código feio e difícil de compreender
  - Não é declarativo e assim, não é lógico
  - Logo, deveria ser usado com cautela
- Os interpretadores Prolog também diferem no modo como **assert/1** e **retract/1** são implementados em relação ao retrocesso
  - Ou a operação de asserção ou a de remoção é cancelada no retrocesso, ou nenhuma delas é cancelada.

Considere esta base de dados

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- descende(maria,Anc).
```

Considere esta base de dados

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- descende(maria,Anc).
Anc=carla;
```

Considere esta base de dados

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- descende(maria,Anc).
Anc=carla;
Anc=carolina;
```

Considere esta base de dados

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- descende(maria,Anc).
Anc=carla;
Anc=carolina;
Anc=laura;
```

Considere esta base de dados

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- descende(maria,Anc).
Anc=carla;
Anc=carolina;
Anc=laura;
Anc=rosa;
```

Considere esta base de dados

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- descende(maria,Anc).
Anc=carla;
Anc=carolina;
Anc=laura;
Anc=rosa;
false
```

Coletando soluções

- Podem existir muitas soluções para uma única consulta ao Prolog
  - Entretanto, Prolog gera as soluções uma por uma
- Algumas vezes gostaríamos de ter *todas* as soluções de uma só vez
- Desnecessário dizer, seria útil te-las em um formato limpo e utilizável

Coletando soluções

- Prolog possui três predicados pré-construídos para fazer isto: **findall**/3, **bagof**/3 e **setof**/3
- Em essência, todos estes predicados coletam todas as soluções para uma consulta e as coloca em uma única lista
- Mas existem diferenças importantes entre eles.

findall/3

- A consulta

```
?- findall(O,M,L).
```

produz uma lista **L** de todos os objetos **O** que satisfazem a meta **M**

  - Sempre sucede
  - Unifica **L** com a lista vazia se **M** não pode ser satisfeita

Um exemplo de findall/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- findall(A,descende(maria,A),L).
```

Um exemplo de findall/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- findall(A,descende(maria,A),L).
L=[carla,carolina,laura,rosa]
true
```

Outros exemplos de findall/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- findall(f:A,descende(maria,A),L).
```

Outros exemplos de findall/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- findall(f:A,descende(maria,A),L).
L=[f:carla,f:carolina,f:laura,f:rosa]
true
```

Outros exemplos de findall/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- findall(A,descende(rosa,A),L).
```

Outros exemplos de findall/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- findall(A,descende(rosa,A),L).
L=[ ]
true
```

Outros exemplos de findall/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- findall(d,descende(maria,A),L).
```

Outros exemplos de findall/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- findall(d,descende(maria,A),L).
L=[d,d,d,d]
true
```

findall/3 às vezes é um pouco tosco

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- findall(Anc,descende(Fil,Anc),L).
```

findall/3 às vezes é um pouco tosco

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):- filha(X,Y).
descende(X,Y):- filha(X,Z),
                descende(Z,Y).
```

```
?- findall(Anc,descende(Fil,Anc),L).
L=[carla,carolina,laura, rosa,
   carolina,laura,rosa,laura,rosa,rosa]
true
```

bagof/3

- A consulta

```
?- bagof(O,M,L).
```

produz uma lista **L** de todos os objetos **O** que satisfazem a meta **M**

  - Somente sucede se a meta **M** sucede
  - Liga as variáveis livres em **M**

Usando bagof/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):-
    filha(X,Y).
descende(X,Y):-
    filha(X,Z),
    descende(Z,Y).
```

```
?- bagof(Anc,descende(Fil,Anc),L).
```

Usando bagof/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):-
    filha(X,Y).
descende(X,Y):-
    filha(X,Z),
    descende(Z,Y).
```

```
?- bagof(Anc,descende(Fil,Anc),L).
Fil=carla
L=[carolina,laura,rosa];
```

Usando bagof/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):-
    filha(X,Y).
descende(X,Y):-
    filha(X,Z),
    descende(Z,Y).
```

```
?- bagof(Anc,descende(Fil,Anc),L).
Fil=carla
L=[carolina,laura,rosa];
Fil=carolina
L=[laura, rosa];
```

Usando bagof/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):-
    filha(X,Y).
descende(X,Y):-
    filha(X,Z),
    descende(Z,Y).
```

```
?- bagof(Anc,descende(Fil,Anc),L).
Fil=carla
L=[carolina,laura,rosa];
Fil=carolina
L=[laura, rosa];
Fil=laura
L=[rosa];
```

Usando bagof/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):-
    filha(X,Y).
descende(X,Y):-
    filha(X,Z),
    descende(Z,Y).
```

```
?- bagof(Anc,descende(Fil,Anc),L).
Fil=carla
L=[carolina,laura,rosa];
Fil=carolina
L=[laura, rosa];
Fil=laura
L=[rosa];
Fil=maria
L=[carla,carolina,laura,rosa]
```

Usando bagof/3 com ^

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):-
    filha(X,Y).
descende(X,Y):-
    filha(X,Z),
    descende(Z,Y).
```

```
?- bagof(Anc,Fil^descende(Fil,Anc),L).
L=[carla, carolina, laura, rosa,
    carolina,laura,rosa,laura, rosa, rosa]
```

setof/3

- A consulta

```
?- setof(O,M,L).
```

produz uma lista ordenada **L** de todos os objetos **O** que satisfazem a meta **M**

  - Somente sucede se a meta **M** sucede
  - Liga as variáveis livres em **M**
  - Remove duplicações de **L**
  - Ordena as respostas em **L**

Usando setof/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):-
    filha(X,Y).
descende(X,Y):-
    filha(X,Z),
    descende(Z,Y).
```

```
?- bagof(Anc,Fil^descende(Fil,Anc),L).
L=[carla, carolina, laura, rosa, carolina,
    laura, rosa, laura, rosa, rosa]
true

?-
```

Usando setof/3

```
filha(maria,carla).
filha(carla,carolina).
filha(carolina,laura).
filha(laura,rosa).

descende(X,Y):-
    filha(X,Y).
descende(X,Y):-
    filha(X,Z),
    descende(Z,Y).
```

```
?- bagof(Anc,Fil^descende(Fil,Anc),L).
L=[carla, carolina, laura, rosa, carolina,
    laura, rosa, laura, rosa, rosa]
true

?- setof(Anc,Fil^descende(Fil,Anc),L)..
L=[carla, carolina, laura, rosa]
true

?-
```

Próxima aula

- Trabalharemos com arquivos
  - Discutiremos como as definições de predicados podem ser colocadas em diferentes arquivos
  - Componentes modulares em Prolog
  - Escrevendo e lendo de arquivos