



Apranda **PROLOG** en N Diapositivas

Programación Lógica

Por:

Domínguez Geniz Amalio Javier

< ajdgeniz@hotmail.com >

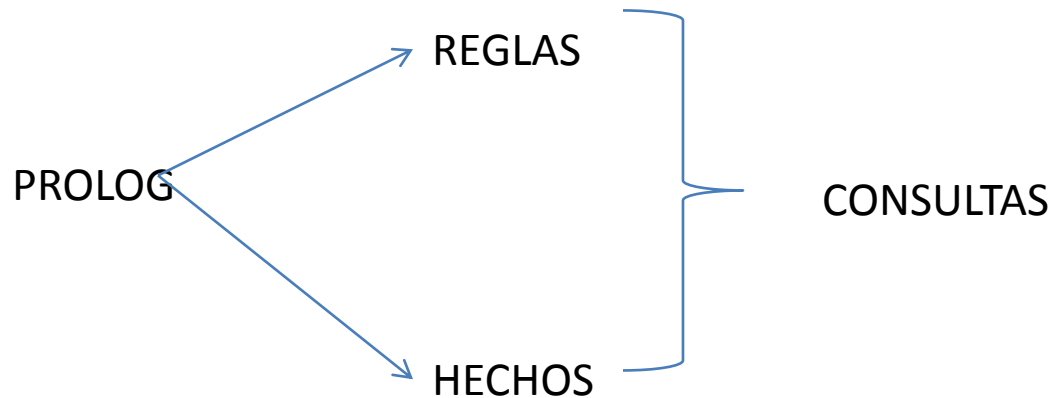
<http://ajdgeniz.wordpress.com>

Tecnológico de Estudios Superiores de Chalco



PROgramming in LOGic

Prolog es un lenguaje declarativo basado en Reglas y Hechos de lógica, cuya información es retribuido en forma de consultas. Originado en Europa a principios de los 70's por Alain Colmerauer . Para realizar los programas, se debe pensar declarativamente.





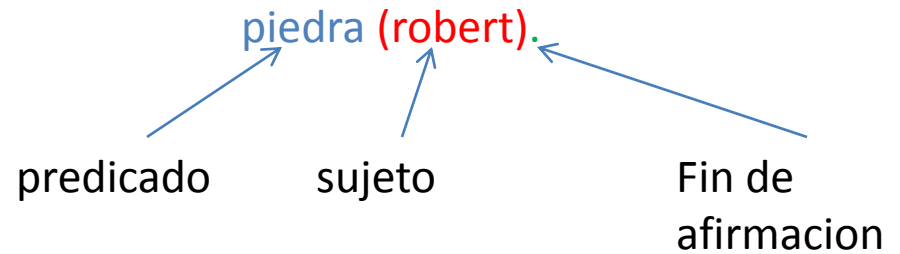
PROgramming in LOGic

HECHOS

- ☐ marco es maestro
- ☐ geniz es un programador

Notación En Prolog

- ✓ maestro (marco).
- ✓ programador(geniz).



Para definir un hecho en Prolog, deberá tomar en cuenta que nuestra oración (hecho) debe llevar el formato `predicado(sujeto)`.



PROgramming in LOGic



SINTAXIS

- ❖ Las variables deben escribirse con Mayúsculas
- ❖ Las constantes se escriben con Minúsculas
- ❖ Las afirmaciones se terminan con . (punto)
- ❖ No se pueden dejar espacios entre los nombres de las constantes, para ello utilice el guion bajo (_)
- ❖ Los comentarios empiezan con %

Operadores

- ❖ Conjunción \rightarrow , (coma)
- ❖ Disyunción \rightarrow ;
- ❖ Regla o Condición \rightarrow :-
- ❖ Fin de la condición \rightarrow .



PROgramming in LOGic



Ejemplo en SWI-Prolog Editor

❖ Ahora basándonos en lo ya aprendido ejecutaremos nuestro primer ejemplo en el editor SWI-Prolog. Para ello teclee lo siguiente:

```
programador(geniz).  
maestro(marco).  
piedra(robert).
```

- Ahora vaya al menu Iniciar y Seleccione “Consultar todo”
- Ahora en el panel de Prolog Teclee las consultas, ejemplo: maestro(marco).

¿marco es maestro?



PROgramming in LOGic

Ejemplo en SWI-Prolog Editor utilizando la conjunción

❖ Ahora utilizaremos variables y la conjunción para realizar consultas. Para ello teclee lo siguiente:

```
sistema_operativo(linux).  
sistema_operativo(windows).  
sistema_operativo(solaris).  
sistema_operativo(mac).
```

```
microsoft(visual_studio,expression_studio).  
sun(netbeans,sun_studio).  
borland(jbuilder,delphi).  
canonical(ubuntu,xubuntu).
```



PROgramming in LOGic

REGLAS

Una regla es una sentencia condicional, por ejemplo:

Base de conocimiento

Regla 1: Si esta contento entonces escucha musica

Regla 2: Si tiene radio entonces escucha musica

Regla 3: Si escucha musica y tiene una guitarra entonces toca la guitarra

Hecho 1: Tiene una guitarra

Hecho 2: Esta contento

Consulta

> Esta tocando la guitarra ?

```
escucha_musica :- esta_contento. % Regla 1
```

```
escucha_musica :- tiene_radio. % Regla 2
```

```
toca_la_guitarra :- escucha_musica, tiene_guitarra. % Regla 3
```

```
tiene_guitarra. % Hecho 1
```

```
esta_contento. % Hecho 2
```



INPUT & OUTPUT

WRITE

La orden WRITE imprime en pantalla la cadena de caracteres en código ASCII por ejemplo:

?- write ("Hola Mundo");

```
3 ?- write("hola Mundo").  
[104, 111, 108, 97, 32, 77, 117, 110, 100, 111]  
true.
```

La manera de hacer que la cadena aparezca de manera normal, es imprimiéndola con una constante, por ejemplo:

?- write(hola_mundo).

```
1 ?- write(hola_mundo).  
hola_mundo  
true.
```

NOTA: Recuerde que las constantes se definen con minúsculas, además que solo deberá utilizar la orden write directamente en el interprete.



INPUT & OUTPUT

write_canonical

La orden `write_canonical` imprime en pantalla un flujo de salida utilizando la notación de un prefijo y los paréntesis, ejemplo:

```
1 ?- write_canonical(+hola+mundo).  
+({(hola), mundo)  
true.
```

writeq, print y display

Sirven de la misma manera que `write`

```
7 ?- print("hola  
| mundo  
| de prolog").  
[104, 111, 108, 97, 10, 109, 117, 110, 100, 111, 10, 100, 101, 32, 112, 114, 111, 108, 111, 103]  
true.
```

tab(N): escribe N espacios en blanco

nl: escribe un salto de línea



INPUT & OUTPUT

READ

La orden read sirve para almacenar el valor a una variable, ejemplo:

?- read(variable):

```
6 ?- write(nombre_ ),  
|   read(NOMBRE),  
|   write(hola_ ),  
|   write(NOMBRE).  
nombre_marco.  
hola_marco  
NOMBRE = marco.
```

NOTA: Recuerde que las variables se definen en letras mayúsculas y también no escriba su nombre con la inicial mayúsculas ya que Prolog lo tomara como otra constante .

INPUT & OUTPUT

READ

Otra forma sencilla de asignar un valor a una variable, colocando directamente el comando, seguido de la variable. Ejemplo:

```
1 ?- read(Piedra) .  
| : robert.  
Piedra = robert.
```

Note que para definir a una variable no es necesario que todas las letras que lo componene sean mayusculas, solo basta con la inicial.



ARITMETICA

PLUS

Plus sirve para sumar los argumentos recibidos, ejemplo:

```
1 ?- plus(1,2,3).  
true.  
  
3 ?- plus(1,2,5).  
false.  
  
4 ?- plus(1,2,SUM).  
SUM = 3.
```

BETWEEN

Encuentra un numero en un rango, ejemplo:

```
3 ?- between(1,7,5).  
true.  
  
4 ?- between(1,7,N).  
N = 1 ;  
N = 2 ;  
N = 3 ;  
N = 4 ;  
N = 5 ;  
N = 6 ;  
N = 7.  
  
5 ?- between(1,7,9).  
false.
```

ARITMETICA

SUCC

Devuelve verdadero si el 2º argumento es $= 1^\circ + 1$ y si el $1^\circ \geq 0$, ejemplo:

```
1 ?- succ(4, 5).  
true.  
  
3 ?- succ(5, 4).  
false.  
  
4 ?- succ(5, 5).  
false.
```

IS

Es un predicado que define una expresión, ejemplo:

```
1 ?- 4 is 2+2.  
true.  
  
3 ?- 10 is 20-15.  
false.  
  
4 ?- 2*5 is 10.  
false.  
  
5 ?- 10 is 2*5.  
true.
```

ARITMETICA

OPERADORES

Operador	Descripción
<	Menor
>	Mayor
=<	Menor que
>=	Mayor que
=\=	Diferente
is	Evalúa si un numero equivale a una expresión
:=	Igual
mod	Modulo

```
1 ?- 5>1.  
true.  
  
3 ?- 5<1.  
false.  
  
4 ?- 5=<1.  
false.  
  
5 ?- 5>=1.  
true.  
  
6 ?- 5=\=10.  
true.  
  
7 ?- 5:=5.  
true.
```



ARITMETICA

FUNCIONES

Función	Descripción
Abs	Valor absoluto
sign	Signo de un numero
Min	Valor minimo
Max	Valor maximo
Random	Numero aleatorio.
round	Redondeo
Floor	Redondeo hacia arriba
ceiling	Redondeo hacia abajo
sqrt	Raiz
powm	Potencia
pi	Valor de pi



ARITMETICA



OTRO EJEMPLO:

`sumar_3_y_duplicar(X, Y) :- Y is (X + 3) * 2.`

```
3 ?- sumar_3_y_duplicar(2, Y).  
Y = 10.
```

`sumar(Y) :- Y is (10 + 3) * 2.`

```
3 ?- sumar(26).  
true.
```


ATOM

Un átomo son aquellos que pertenecen al grupo de las constantes de un vocabulario, se utiliza la orden atom para saber cuales son validas, ejemplo:

```
1 ?- atom(A) .  
false.  
  
3 ?- atom(geniz) .  
true.  
  
4 ?- atom(1) .  
false.  
  
5 ?- atom(marco_1) .  
true.
```

Recuerde que las contantes deben empezar con minúsculas , aunque la orden atom también puede buscar en una cadena.

```
1 ?- atom("A") .  
false.  
  
3 ?- atom('A') .  
true.
```



RECURSION



Ejemplo:

hijo_de(maria, carlos).

hijo_de(carlos, cristina).

hijo_de(cristina, luis).

descendiente(X, Y) :- hijo_de(X, Y).

descendiente(X, Y) :- hijo_de(X, Z), descendiente(Z, Y).

Consulta:

```
4 ?- descendiente(maria, luis).  
true |
```



LISTAS



- Términos

- Lista vacia: []

- Lista compuesta: [<termino>+f| <lista>g]

<términos>: sucesión de los primeros elementos de la lista

<lista>: lista con los restantes elementos

$[a, b, c] = [a, b, c | []] = [a, b | [c]] = [a | [b, c]]$

- Algunas relaciones que trabajan con listas:

- append(L1, L2, L3) :- La lista L3 única con la concatenación de las listas L1 y L2

- member(E, L) :- E unica con alguno de los elementos de la lista L

- reverse(L1, L2) :- La inversa de la lista L1 unica con la lista L2

Si se tiene la lista [a, b, c, d], la a es la cabeza y la cola es la lista [b, c, d]

- Una lista cuya cabeza es A y cola es B se anota como [A | B]

- El predicado

`primer_elemento(X, [X|_]).`

tiene éxito si X es el primer elemento de la lista.

`[1,2,3] = [1|[2|[3|[]]]].`

```
7 ?- [X|Xs]=[1,2,3].  
X = 1,  
Xs = [2, 3].
```



LISTAS



Existe un modulo que se carga por defecto para el manejo de listas y es el que vamos a tratar ahora. Para manejar listas SWI-Prolog proporciona los dos términos constructores de la lista: `[]`, la constante lista vaca; y `[X|R]` el operador (función infija) concatenar por la cabeza, donde `R` debe ser una lista a su vez. Para una definición mas formal de lista consultar el predicado `is list/1`. También se proporciona predicados para el manejo de listas.

`is list(+Term)`: cierto si `Term` es una lista.

`length(?List, ?Int)`: `Int` es el numero de elementos de la lista `List`.

`sort(+List, -Sorted)`: `Sorted` es la lista ordenada de los elementos de `List` sin duplicados.

`append(?List1, ?List2, ?List3)`: `List3` es la concaténation de `List1` y `List2`

`member(?Elem, ?List)`: `Elem` es elemento de `List`.

`nextto(?X, ?Y, ?List)`: `Y` está despues de `X` en la lista `List`.

`delete(+List1, ?Elem, ?List2)`: `List2` es la eliminación de todos los elementos que unifican simultaneamente con `Elem` de `List1`.

`nth0(?Index, ?List, ?Elem)`: `Elem` es el `Index` -ésimo elemento de `List`, comenzando por el 0.

`reverse(+List1, -List2)`: `List2` es `List1` pero con el orden de los elementos cambiado.



LISTAS



listing.

```
dios_egipcio(amon).  
dios_egipcio(anubis).  
dios_egipcio(apis).  
dios_egipcio(ra).
```

La orden **findall** realiza todo el árbol

```
3 ?- findall(X,dios_egipcio(X), L).  
L = [amon, anubis, apis, ra].
```



EJEMPLOS



```
listing(append/3).  
lists:append([], A, A).  
lists:append([A | B], C, [A | D]) :-  
append(B, C, D).
```

```
listing(member).  
lists:member(A, [A | B]).  
lists:member(A, [B | C]) :-  
member(A, C).
```

```
listing(reverse).  
lists:reverse(A, B) :-  
reverse(A, [], B, B).
```

```
lists:reverse([], A, A, []).  
lists:reverse([A | B], C, D, [E | F]) :-  
reverse(B, [A | C], D, F).
```

El predicado `\+` tiene éxito sólo si fracasa su argumento.

- Considere los siguientes hechos:

`roja(rosa).`

`verde(hierba).`

`blanca(margarita).`

- Suponga la siguiente consulta:

```
3 ?- roja(margarita).  
false.  
  
4 ?- \+roja(margarita).  
true.
```


Ejemplos clásicos

```
1 /// Autor: %AUTHOR%  
2 /// Datum: %DATE%  
3  
4 %%  
5 %% declaraciones  
6 %%  
7  
8 padrede('juan', 'maria'). % juan es padre de maria  
9 padrede('pablo', 'juan'). % pablo es padre de juan  
10 padrede('pablo', 'marcela').  
11 padrede('carlos', 'debora').  
12  
13 % A es hijo de B si B es padre de A  
14 hijode(A,B) :- padrede(B,A).  
15 % A es abuelo de B si A es padre de C y C es padre B  
16 abuelode(A,B) :-  
17     padrede(A,C),  
18     padrede(C,B).  
19 % A y B son hermanos si el padre de A es también el padre de B y si A y B no son lo mismo  
20 hermanode(A,B) :-  
21     padrede(C,A) ,  
22     padrede(C,B) ,  
23     A \== B.  
24  
25 % A y B son familiares si A es padre de B o A es hijo de B o A es hermano de B  
26 familiarde(A,B) :-  
27     padrede(A,B).  
28 familiarde(A,B) :-  
29     hijode(A,B).  
30 familiarde(A,B) :-  
31     hermanode(A,B).
```



Ejemplos clásicos

Basándonos en el ejemplo anterior haremos las siguientes consultas:

%%% consultas %%%
% juan es hermano de marcela?

?- hermanode('juan', 'marcela').

yes

% carlos es hermano de juan?

?- hermanode('carlos', 'juan').

no

% pablo es abuelo de maria?

?- abuelode('pablo', 'maria').

yes

% maria es abuela de pablo?

?- abuelode('maria', 'pablo'). no

```
1 ?- consult('E:/Prolog/family.pl').  
% E:/Prolog/family.pl compiled 0.00 sec, 1,908 bytes  
true.  
  
3 ?- hermanode('juan', 'marcela').  
true  
.  
  
4 ?- hermanode('carlos', 'juan').  
false.  
  
5 ?- abuelode('pablo', 'maria').  
true  
.  
  
6 ?- abuelode('maria', 'pablo').  
false.
```



Factorial (Recursividad)

Ahora veremos como Prolog al igual que otro lenguaje también tiene recursividad, y que mejor que con el clásico ejemplo de un factorial:

```
1 %// Autor: %GENIZ%
2 %// Datum: %18/03/09%
3
4 factorial(0, 1) :- !.
5 factorial(N, F) :- N1 is N - 1, factorial(N1, F1), F is N*F1.
```

```
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.6.64)
Copyright (c) 1990-2008 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.
```

```
For help, use ?- help(Topic). or ?- apropos(Word).
```

```
1 ?- consult('E:/Prolog/Factorial.pl').
% E:/Prolog/Factorial.pl compiled 0.00 sec, 688 bytes
true.

3 ?- factorial(5,120).
true.
```



Mas sobre listas(creación)

A continuación haremos operaciones básicas con listas, nótese la sintaxis para definir las, recuerde que una lista esta compuesta de cabeza(primer elemento) y cola(resto de los elementos):

```
1 %// Autor: %GENIZ%
2 %// Datum: %2009%
3
4 mis_lenguajes_favoritos([prolog, java, c, perl, gcc, net, javascript, foxpro, delphi, python, php, jsp, haskell, mondrian]).
5 lista_numeros([1,2,3,4,5,6,7,8,9,10]).
```

SWI-Prolog comes with **ABSOLUTELY NO WARRANTY**. This is free software, and you are welcome to redistribute it under certain conditions. Please visit <http://www.swi-prolog.org> for details.

For help, use `?- help(Topic)`. or `?- apropos(Word)`.

```
1 ?- consult('E:/Prolog/creacion de listas.pl').
% E:/Prolog/creacion de listas.pl compiled 0.00 sec, 1,452 bytes
true.

3 ?- mis_lenguajes_favoritos([Cabeza|Cola]).
Cabeza = prolog,
Cola = [java, c, perl, gcc, net, javascript, foxpro, delphi, python|...].

4 ?-
| lista_numeros([Cabeza|Cola]).
Cabeza = 1,
Cola = [2, 3, 4, 5, 6, 7, 8, 9, 10].
```



Mas sobre listas(Longitud)

% Si queremos hallar la longitud de una lista.

% La longitud de una lista vacía es 0.

% La longitud de cualquier lista es la longitud de la cola + 1.

```
1 % // Autor: %AUTHOR%
2 % // Datum: %DATE%
3 longitud([],0).
4 longitud([_|T],N):-longitud(T,NO), N is NO + 1.
5
```

For help, use ?- help(Topic). or ?- apropos(Word).

```
1 ?- consult('E:/Prolog/longitud.pl').
```

```
Warning: e:/prolog/longitud.pl:4:
```

```
Singleton variables: [H]
```

```
% E:/Prolog/longitud.pl compiled 0.02 sec, 672 bytes
```

```
true.
```

```
3 ?- longitud([a,b,c],L).
```

```
L = 3.
```

```
4 ?- longitud([a,b,c],4).
```

```
false.
```

Mas sobre listas(Búsqueda)

% Si queremos determinar si un elemento pertenece a una lista.

% El elemento pertenece a la lista si coincide con la cabeza de la lista.

% El elemento pertenece a la lista si se encuentra en la cola de la lista.

```
1 % // Autor: %GENIZ%
2 % // Datum: %2009%
3
4 pertenece(X,[X|_]) :- !.
5 pertenece(X,[_|R]) :- pertenece(X,R).
```

```
1 ?- consult('E:/Prolog/busqueda.pl').
% E:/Prolog/busqueda.pl compiled 0.00 sec, 608 bytes
true.

3 ?- pertenece(b,[a,b,c]).
true.

4 ?- pertenece(b,[a,[b,c]]).
false.

5 ?- pertenece([b,c],[a,[b,c]]).
true.
```



Mas sobre listas(Eliminación)

% Si queremos eliminar un elemento de la lista.

% Si X es la cabeza de la lista, la cola T es la lista sin X

% Si X no es la cabeza de la lista, conservamos la cabeza de la lista

% como parte de la respuesta y continuamos eliminando X de la cola T.

```
1 % // Autor: %GENIZ%
2 % // Datum: %DATE%
3
4 elimina(X,[X|T],T).
5 elimina(X,[H|T],[H|T1]):- elimina(X,T,T1).
```

For help, use ?- help(Topic). or ?- apropos(Word).

```
1 ?- consult('E:/Prolog/eliminar.pl').
% E:/Prolog/eliminar.pl compiled 0.00 sec, 652 bytes
true.
```

```
3 ?- elimina(1,[1,2,3,4],R).
R = [2, 3, 4] ;
false.
```

```
4 ?- elimina(1,R,[2,3]).
R = [1, 2, 3] ;
R = [2, 1, 3] ;
R = [2, 3, 1] ;
false.
```



Mas sobre listas(Concatenar)

% Si queremos concatenar dos listas lista.
% Concatenar una lista vacía con L es L.
% Concatenar X|L1 con L2 es poner el primer
% elemento de la primera lista (X) más la
% concatenación del resto de la lista (L1) con L2

```
1 % // Autor: %GENIZ%  
2 % // Datum: %DATE%  
3  
4 concatenar([],L,L).  
5 concatenar([X|L1],L2,[X|L3]):-concatenar(L1,L2,L3).
```

For help, use ?- help(*Topic*). or ?- apropos(*Word*).

```
1 ?- consult('E:/Prolog/concatenar.pl').  
% E:/Prolog/concatenar.pl compiled 0.00 sec, 644 bytes  
true.
```

```
3 ?- concatenar([1,2],[3,4],R).  
R = [1, 2, 3, 4].
```




PROLOG & MySQL

Para conectar a Prolog con una base de datos de MySQL, recuerde que únicamente se puede hacer vía ODBC por lo cual deberá tener instalado el controlador ODBC de MySQL. En este manual partiremos suponiendo que Connector ODBC se encuentra instalado ya que no hay gran ciencia en la instalación, ahora abra MySQL y cree una base de datos, ejemplo:

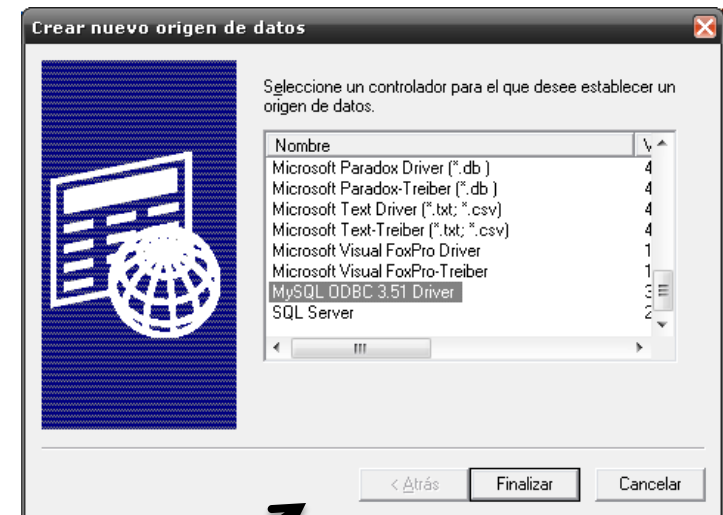
```
CREATE DATABASE Prolog;  
USE Prolog;
```

```
CREATE TABLE Datos(  
Nombre VARCHAR(30) ,  
Apellidos VARCHAR(35) );
```

```
INSERT INTO Datos SET Nombre='javier', Apellidos='Geniz';
```

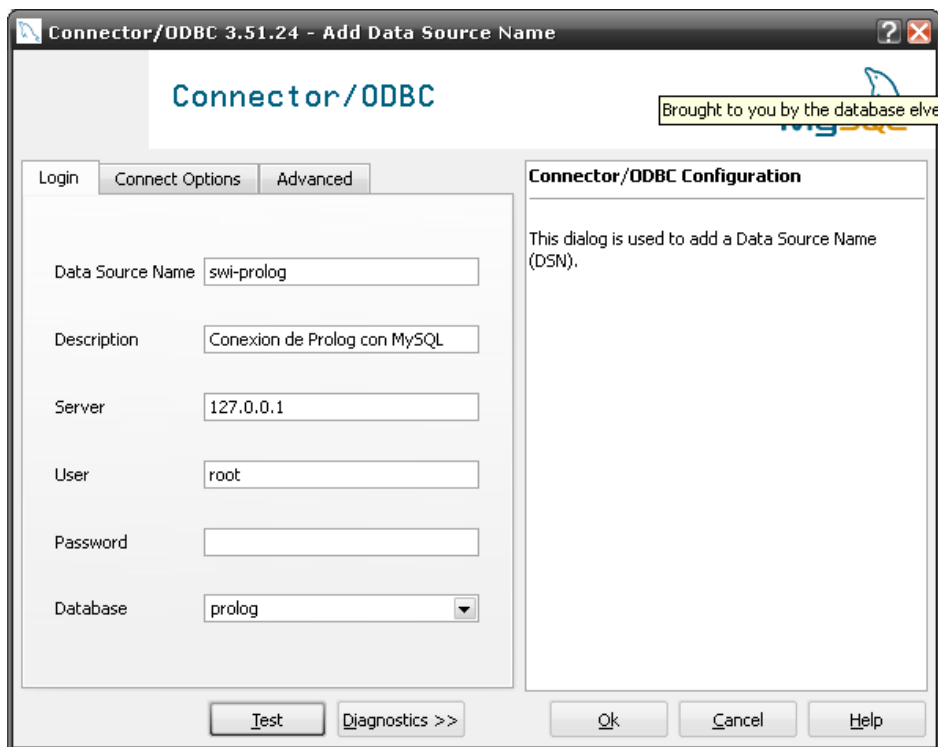
Una vez que tenemos nuestra base de datos, es hora de hacer nuestro Origen de datos ODBC, ejemplo:

1. Abra “Ejecutar” o presione Windows + R y escribe lo siguiente: **odbcad32**
2. Se abra el Administrador de Origenes ODBC:



3. Ahora haga clic en agregar y seleccione MySQL ODBC y despues haga clic en el boton finalizar.

5. Ahora configure el origen de manera análoga a lo siguiente:



Connector/ODBC 3.51.24 - Add Data Source Name

Connector/ODBC

Brought to you by the database elve

Login | **Connect Options** | Advanced

Data Source Name: swi-prolog

Description: Conexion de Prolog con MySQL

Server: 127.0.0.1

User: root

Password:

Database: prolog

Connector/ODBC Configuration

This dialog is used to add a Data Source Name (DSN).

Test | Diagnostics >> | Ok | Cancel | Help

Orígenes de datos de usuario:

Nombre	Controlador	Agregar...
dBASE Files	Microsoft Access dBASE Driver (*.dbf, *.ndx)	Quitar
Excel Files	Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.x)	
MS Access Database	Microsoft Access Driver (*.mdb, *.accdb)	
swi-prolog	MySQL ODBC 3.51 Driver	
		Configurar...

6. Listo, puede hacer clic en el botón Test para probar la conexión y después en OK, recuerde el nombre del origen de datos ya que será utilizado para establecer la conexión.



PROLOG & MySQL

7. Ahora abra el editor de SWI-PROLOG y teclee el siguiente código:

```
1 % Autor: %GENIZ%
2 % Datum: %18/03/09%
3
4 conexion :-
5   odbc_connect('swi-prolog', _, [user(root),password(''), alias(base_datos),open(once)]),
6
7   odbc_prepare(base_datos,'SELECT Apellidos FROM Datos WHERE Nombre=?',
8   [atom>varchar(255)],
9   Handle,
10  [types([atom])]),
11  abolish(odbc_handle/1),
12  assert(odbc_handle(Handle)).
13
14 run_stmt(Registro) :-
15   odbc_handle(Handle),
16   odbc_execute(Handle, ['javier'],Registro).
17
18 ejecutar(Registro):-
19   conexion,run_stmt(Registro).
20
```



PROLOG & MySQL

8. Una vez que este código este listo, presione F9 para compilar el código(no tiene errores) y después en la parte del interprete teclee la siguiente consulta:

?- ejecutar(Registro).

```
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.6.64)
Copyright (c) 1990-2008 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- consult('E:/Prolog/MySQL.pl').
% E:/Prolog/MySQL.pl compiled 0.00 sec, 2,016 bytes
true.

3 ?- ejecutar(Registro).
Registro = row('Geniz').
```

Registro devuelto por MySQL



EJERCICIO

- Elabore un programa en Prolog con su árbol genealógico, donde los hechos sean únicamente predicados del tipo padre (-, -) ó madre (-, -)
- Programe los predicados con las reglas necesarias para encontrar las relaciones de parentesco más comunes, tales como:

- a) hermano(A,B).
- b) primo(A,B).
- c) tio(A,B).
- d) hijo(A,B).
- e) nieto(A,B).
- f) abuelo(A,B).
- g) bisabuelo(A,B).
- h) bisnieto(A,B).
- i) cuñado(A,B).
- j) concuño(A,B).

```

1 % Autor: Geniz
2 % Fecha: 17/06/2009
3 %NOTA: javier soy yo(geniz).
4 papa('julian', 'juan'). % julian es papa de juan
5 papa('juan', 'jose'). % juan es papa de jose
6 papa('juan', 'emiliano'). % juan es papa de jose
7 papa('jose', 'christian'). % jose es papa de christian
8 papa('jose', 'javier'). % jose es papa de christian
9 esposa('ivonne', 'christian'). %ivonne es esposa de christian
10 mama('catalina', 'javier'). % catalina es mama de javier
11 papa('emiliano', 'luis'). % emiliano es papa de luis
12
13 % A es BISABUELO de B si A es papa de C, C es papa de B y C
14 bisabuelo(A,B):-papa(A,C),papa(C,D),papa(D,B).
15
16 %A es TIO de B si el papa de B es C y si D es papa de C y si D es papa de A
17 tio(A,B):-papa(C,B),papa(D,C),papa(D,A).
18
19 %A es CUÑADO de B si A es esposa de C y si el papa de C es el mismo que el papa de B
20 cuñado(A,B):-esposa(A,C),papa(D,C),papa(D,B).
21
22 % A es ABUELO de B si A es papa de C y C es papa B
23 abuelo(A,B):-papa(A,C),papa(C,B).
24
25 % A es HIJO de B si B es papa de A
26 hijo(A,B):-papa(B,A).
27
28 % A y B son HERMANOS si el papa de A es también el papa de B y si A y B no son lo mismo
29 hermano(A,B):-papa(C,A),papa(C,B),A \== B.
30
31 %A y B son PRIMOS si A es hijo del tio de B
32 primo(A,B):-papa(C,A),papa(D,B),hermano(C,D).
33
34 % A y B son FAMILIARES si A es papa de B o A es hijo de B o A es hermano de B
35 familiar(A,B):-papa(A,B).
36 familiar(A,B):-hijo(A,B).
37 familiar(A,B):-hermano(A,B).

```



Para más información:

?? ***Programming in XPCE/Prolog: Guía de usuario para aprender a***
programar en *Prolog* con el *XPCE*, desde lo más básico hasta lo más complejo.

<http://www.swi.psy.uva.nl/projects/xpce/UserGuide/>

?? ***Class summary descriptions: Página donde se puede encontrar información***
sobre gran numero de clases y algunos de sus métodos, con algún ejemplo.

<http://www.swi.psy.uva.nl/projects/xpce/UserGuide/summary.html>

?? ***Pagina principal de SWI-Prolog: Página de donde descargar un interprete***
o la documentación para programar en Prolog y XPCE

<http://www.swi-prolog.org/>

?? ***The XPCE online reference manual: manual de referencia con todos los***
metodos y objetos que pueden ser creados y referenciados.

<http://gollem.swi.psy.uva.nl:8080/>