

Arrancamos con un ejemplo...

Ejemplo 2:

Juan gusta de María.
 Pedro gusta de Ana y de Nora.
 Todos los que gustan de Nora gustan de Zulema.
 Julián gusta de las morochas y de las chicas con onda.
 Mario gusta de las morochas con onda y de Luisa.
 Todos los que gustan de Ana y de Luisa, gustan de Laura.
 Después cambiar ese "y" por un "o".

Cómo sería cada uno de los ejemplos:

“Juan gusta de María”

`gusta(juan, maria).`

¿Es un predicado monádico o poliádico? Poliádico, porque expresa una relación entre juan y maría.

Recordemos que juan y maria van en minúscula, porque son los individuos (átomos) que componen la relación.

“Pedro gusta de Ana y de Nora”

El castellano es engañoso, a veces digo "y" cuando quiero decir "o".

`gusta(pedro, ana).`

`gusta(pedro, nora).`

Es decir “Pedro gusta de Ana” es verdadero, lo mismo que “Pedro gusta de Nora”.

Entonces en realidad esto es un or; repetir la cláusula es otra forma de decir:

`gusta(pedro, ana) or gusta(pedro, nora) se cumple.`

Todos los que gustan de Nora gustan de Zulema quiere decir: a alguien le gusta Zulema si le gusta Nora.

$p \Rightarrow q$ lo transformamos a “q si p”.

`gusta(Alguien, zulema) :- gusta(Alguien, nora).`

Vamos con el de Julián, las morochas y las chicas con onda.

“Julián gusta de las morochas y de las chicas con onda”

Si una chica tiene onda, ¿le gusta a Julián? ¿importa si es morocha o rubia?

Si lo doy vuelta se entiende mejor: Julián gusta de una chica si es morocha o si tiene onda.

Podemos escribirlo:

`gusta(julian, Chica) :- chica(Chica), morocha(Chica).`

`gusta(julian, Chica) :- chica(Chica), tiene_onda(Chica).`

Que es en realidad:

`gusta(julian, Chica) :- chica(Chica), (morocha(Chica) or tiene_onda(Chica)).`

Pero la primera forma de escribirlo es la que vamos a preferir.

“Mario gusta de las morochas con onda y de Luisa”

`gusta(mario, Chica) :- chica(Chica), morocha(Chica), tiene_onda(Chica).`

`gusta(mario, luisa).`

“Todos los que gustan de Ana y de Luisa, gustan de Laura”

Si es con “y”:

`gusta(Alguien, laura) :- gusta(Alguien, ana), gusta(Alguien, luisa).`

Y si es con “o”:

```
gusta(Alguien, laura) :- gusta(Alguien, ana).
gusta(Alguien, laura) :- gusta(Alguien, luisa).
```

Consultas

Las consultas pueden ser **existenciales** o **individuales** (porque se refieren a individuos).

Ejemplo de Consulta Existencial: ¿existe alguien al que le guste Nora?

¿Cuántos individuos va a encontrar? Si yo lo se de antemano, se dice que la consulta es determinística. Si no lo se, la consulta es... no-determinística.

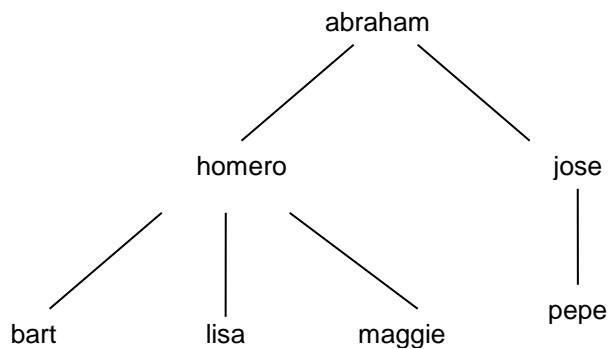
Ejemplo de Consulta individual: ¿a Julián le gusta Nora?

Contamos un poco cómo trabajar en Entorno SWI (new, consult, make). Recordar el gráfico de una aplicación en Paradigma lógico y mostrarles conceptualmente cómo se labura.

?- help. → me permite acceder a la ayuda on-line

Más ejemplos:

Asentamos conceptos con ejemplo de la familia: hermano, tío, primo, abuelo.



Dados los siguientes padres:

```
padre(homero, bart).
padre(homero, lisa).
padre(homero, maggie).
padre(abraham, homero).
padre(abraham, jose).
padre(jose, pepe).
```

Resolver los predicados hermano, tío, primo y abuelo.

hermano(X,Y):-padre(Z, X), padre(Z, Y), X \= Y. → El símbolo \= significa “distinto”

tio(X, Y):-padre(Z, X), hermano(Z, Y). → si el primero es el sobrino y el segundo es el tío.

tio(X, Y):-hermano(X, Z), padre(Z, Y). → si el primero es el tío y el segundo el sobrino.

primo(X, Y):-padre(Z, X), padre(T, Y), hermano(Z, T).

abuelo(X, Y):-padre(X, Z), padre(Z, Y).

¿Vieron que es fácil trabajar en Lógico? El tema es armar una base de conocimientos que me permita resolver las consultas que yo quiero pedir. Por ejemplo:

¿Quiénes son los abuelos de bart?

```
?- abuelo(Quien, bart).
```

```
Quien = abraham
```

Acá el SWI Prolog me pide que yo le diga qué hacer: ¿Sigo buscando más soluciones? ¿Paro acá? Con “;” sigo buscando, con Enter dejo de buscar más soluciones.

Si le doy punto y coma al final me dice “false”, no porque falló el predicado: es porque no encontró más soluciones.

¿Qué pasó en el medio?

El motor de inferencia de PROLOG **unificó** la variable *Quien* con el valor “abraham”. Recordemos que, dentro del programa lógico, las variables son **incógnitas** que el motor trata de resolver.

Cuando hay múltiples resultados las variables se van unificando a distintos valores, como es el caso de:

```
?- padre(homero, Quien).
```

Acá el motor encontrará 3 resultados posibles para *Quien*:

```
Quien = bart ;
```

```
Quien = lisa ;
```

```
Quien = maggie ;
```

```
false
```

Tres ideas sobre unificación

- Unificación en valores devueltos por las consultas existenciales

Cuando yo pregunto `padre(homero, Quien)`, si hay múltiples resultados es porque el motor encuentra más de una manera de resolver / unificar las incógnitas (*Quien* puede unificarse con *bart*, *lisa* o *maggie*).

- Unificación vs. Asignación:

Si yo tengo una cláusula

```
sumarUno(X, ValorNuevo) :- ValorNuevo is X + 1.
```

Cuando *ValorNuevo* es una incógnita la despeja, entonces pasa a tener un valor conocido. Pero no es una posición de memoria, no puedo hacer `ValorNuevo is ValorNuevo + 1`. No puedo almacenar valores intermedios, por lo tanto recalcamos: **No hay efecto colateral**.

- Unificación relacionada con pattern matching.

Cuando yo pregunto

```
? sumarUno(1, N)
```

1 se unifica con *X* (unificación de incógnita con valor)

N se unifica con *ValorNuevo* (unificación de incógnita con incógnita).

Ejemplo:

Hago una consulta individual sobre un predicado definido por comprensión y muestro cómo la “variable” se reemplaza/unifica con el valor del individuo consultado.

```
persona(juan, 89).
```

```
persona(fer, 34).
```

```
persona(melina, 6).
```

```
persona(chiara, 2).
```

```
viejo(X):-persona(X, Edad), Edad > 65.
```

Cuando nosotros preguntamos:

```
? viejo(juan).
```

La variable `x` se unifica a `juan`. Entonces lo que sucede es que el motor busca unificar `persona(juan, Edad)` y ahí termina resolviéndose la cuestión.

Lógico vs. Funcional/Imperativo

En C no puedo dejar libre a la variable dentro de una consulta. Yo no puedo hacer: `viejo(Quien)` en C/Pascal. Para eso tengo que diseñar un algoritmo que me haga la búsqueda.

```
for each persona in personas;
    if viejo(persona)
        agregar persona a las soluciones a devolver
    end if
end for;
devolver soluciones.
```

Y así con cada una de las consultas posibles.

También tenemos la opción de trabajar con listas (que vamos a ver más adelante), pero está bueno trabajar con predicados individuales, porque el motor se puede encargar de encontrar:

- Todos juntos (o sea, en formato de lista)

```
?- amigos(Quienes)
Quienes = [ flor, leo, nico, gaby ]
```
- Por separado:

```
?- amigo(Quien)
Quien = flor
Quien = leo
Quien = nico
Quien = gaby
```

Pattern Matching

Predicado saludo/2

```
saludo("Hola", "Hola muchachos"). (1)
saludo("Chau", "Chau muchachos"). (2)
saludo(X, "Me parece bien"). (3)
```

Si yo le envío

```
saludo("Hola", Respuesta)
Respuesta = "Hola muchachos"
Respuesta = "Me parece bien"
```

El pattern matching se produce en todos los argumentos donde pueda unificarse valor o variable contra lo que yo mandé. Por eso tengo 2 “matchings” posibles:

```
"Hola" = "Hola" (1º predicado) y
"Hola" = X (3º predicado - variable sin unificar).
```

Intro a Functores

Predicado cumple/... ¿?

```
nacio(mati, 3, 8, 1981).
```

¿Cuántas cosas relaciona el predicado cumple? ¿Y cuántas queremos que relacione?

OK... hay que cambiar algo, evidentemente. Si queremos relacionar dos cosas, la fecha tiene que ser una cosa única. Esto lo hacemos mediante functores:

```
nacio(mati, fecha(3, 8, 1981)).
```

¿Qué consultas podemos hacer con este predicado?

```
?- nacio(mati, fecha(3, 8, 1981)).
```

```
?- nacio(mati, Fecha).
```

```
?- nacio(Quien, Fecha).
```

```
?- nacio(Quien, fecha(3, 8, 1981)).
```

OK, todas esas ya las conocíamos, pero también podemos querer consultar... ¿quién nació en 1981?

```
?- nacio(Quien, fecha(_,_,1981)).
```

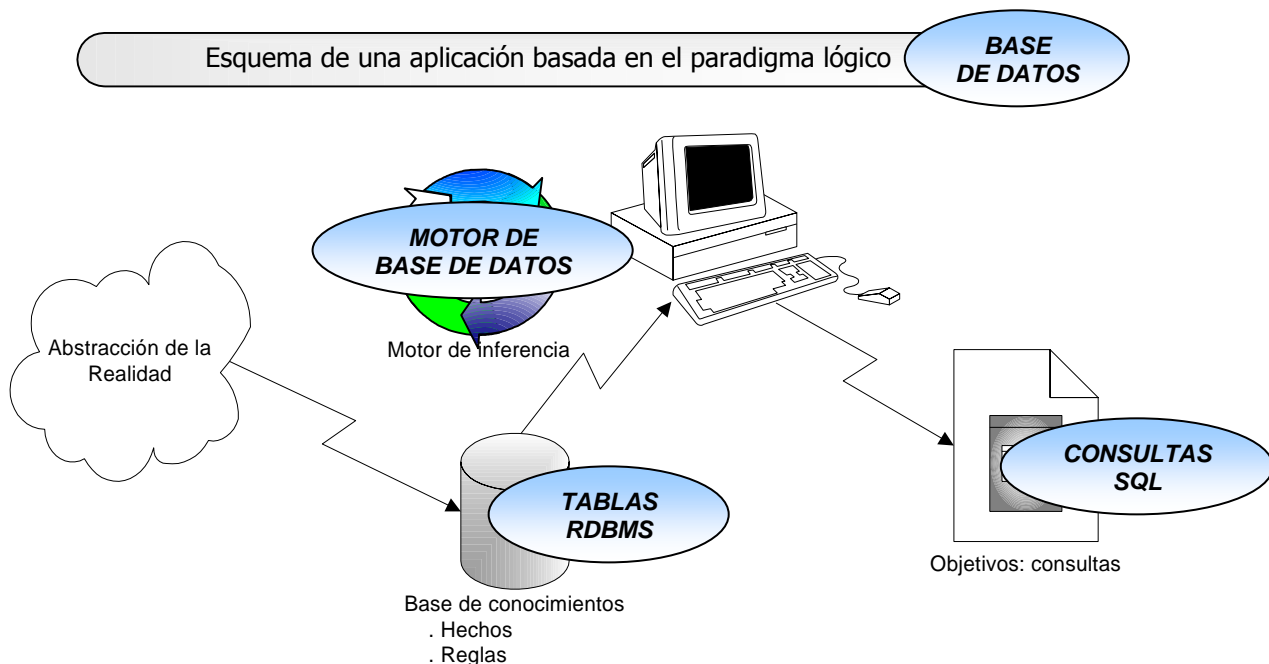
O al revés... ¿en qué mes nació Mati?

```
?- nacio(mati, fecha(_,Mes,_)).
```

¿Por qué trabajamos de esta forma?

Nos gustaría contarles que en Gestión de Datos van a ver cómo trabajar con tablas según el modelo relacional, y nuestra esperanza es que ustedes se acuerden de que trabajar con SQL es muy similar a trabajar en el paradigma lógico, porque tenemos:

- Una base de datos (que es la base de conocimientos de Lógico) donde voy agregando información
- Un lenguaje de consulta bien declarativo (SQL / consultas de Lógico) que determina el qué
- Un motor (el de la base de datos / el de inferencia de Lógico) que termina resolviendo el cómo



Un ejemplito en SQL. Tengo la tabla

Personas

ID_PERSONA INT (PK)
 NOMBRE VARCHAR(30)
 EDAD INT

| ID_PERSONA | NOMBRE | EDAD |
|------------|-----------|------|
| 1 | JUAN | 23 |
| 2 | SATURNINO | 59 |
| 3 | OSVALDO | 10 |
| 4 | MARIO | 45 |

Si quiero seleccionar las personas mayores de 30: “Dame las personas cuya edad sea mayor a 30”

```
SELECT *
  FROM PERSONAS
 WHERE EDAD > 30
```

¿Cómo hago para encontrar los registros?

¿Están en una tabla de hashing?

¿Accedo por un índice único clustered?

¿Cuántos registros tiene cada página del buffer?

¿Cómo los trae en memoria?

No me interesa, yo confío en que Oracle/SQL Server/MySQL/Sybase/Informix me va a traer la información que le pedí.

Lo mismo va a pasar en Lógico:

persona(juan, 23).

persona(saturnino, 59).

persona(osvaldo, 10).

persona(mario, 45).

Si yo pido:

persona(X, Edad), Edad > 30.

No me voy a preocupar para saber cómo hace el motor de inferencia de PROLOG para encontrar todas las soluciones. Sé que, si especifico correctamente lo que quiero, eso es lo que voy a obtener.