

Práctica II : VEHICLE SALES MANAGEMENT SYSTEM.

Joan Stiven Peralta Bedoya

Universidad EAFIT

Lenguajes de Programación

Alexander Narváez Berrío.

Facultad de Ingeniería

Ingeniería de Sistemas

Medellín

2025

Contenido

INTRODUCCIÓN.....	3
CATÁLOGO DE VEHÍCULOS.....	4
CONSULTAS BÁSICAS Y FILTROS.....	7
GENERACIÓN DE REPORTES	11
CASOS DE PRUEBAS	15

INTRODUCCIÓN

Este proyecto consiste en la implementación de un sistema de gestión de ventas de vehículos utilizando el lenguaje de programación lógica Prolog. El sistema permite representar un catálogo de vehículos, realizar consultas por atributos clave (marca, tipo, precio y año), y generar reportes filtrados aplicando restricciones como el presupuesto máximo.

Tiene como objetivo aplicar conceptos de lógica, programación declarativa y generación de reportes mediante predicados como `findall/3` y `bagof/3`.

El sistema permite:

- Agregar nuevos vehículos al inventario evitando duplicados.
- Consultar vehículos por tipo y presupuesto.
- Listar referencias por marca.
- Generar reportes económicos priorizando vehículos más baratos hasta un límite total.

CATÁLOGO DE VEHÍCULOS

El conocimiento del sistema se representa mediante hechos en Prolog utilizando el predicado:

`vehicle(Brand, Reference, Type, Price, Year).`

Donde cada argumento representa:

Brand: la marca del vehículo (toyota, ford, bmw, etc.).

Reference: el nombre o referencia del modelo (corolla, mustang, civic, etc.).

Type: el tipo de vehículo (sedan, suv, pickup, sport).

Price: el precio del vehículo en dólares.

Year: el año del modelo.

Todos los hechos están declarados como dinámicos mediante:

`:- dynamic vehicle/5.`

Esto permite modificar la base de conocimiento en tiempo de ejecución, es decir, agregar o eliminar vehículos mientras se ejecuta el programa.

Ejemplo de hechos definidos

`vehicle(toyota, corolla, sedan, 22000, 2023).`

vehicle(ford, mustang, sport, 48000, 2023).

vehicle(bmw, x5, suv, 65000, 2022).

vehicle(honda, civic, sedan, 23500, 2023).

Esta estructura permite realizar consultas sobre el inventario de forma lógica y declarativa, aplicando filtros por marca, tipo, año o presupuesto.

add_vehicle/5

Este predicado permite añadir nuevos vehículos al catálogo, asegurando que no existan duplicados por referencia dentro de la misma marca:

`add_vehicle(Brand0, Ref, Type0, Price, Year).`

Funcionamiento:

- Convierte Brand0 y Type0 a minúsculas para mantener consistencia.
- Verifica que no exista ya un vehículo con la misma marca y referencia.
- Si no existe, lo agrega con assertz/1.

Ejemplo de uso:

?- add_vehicle(hyundai, tucson, Suv, 27000, 2023).

true.

?- vehicle(hyundai, tucson, suv, 27000, 2023).

true.

Intento duplicado:

?- add_vehicle(toyota, corolla, sedan, 22000, 2023).

false. Ya existe, no se agrega

```

1 % PARTE 1: Catálogo de Vehículos (Hechos)
2
3 % Definición: vehicle(Marca, Referencia, Tipo, Precio, Año).
4
5 :- dynamic vehicle/5.
6
7 % Hechos iniciales (Brand en minúscula, tipos: sedan, suv, pickup, sport)
8 vehicle(toyota, corolla, sedan, 22000, 2023).
9 vehicle(toyota, camry, sedan, 28000, 2022).
10 vehicle(toyota, rav4, suv, 29500, 2023).
11 vehicle(toyota, hilux, pickup, 35000, 2022).
12 vehicle(ford, focus, sedan, 21000, 2022).
13 vehicle(ford, escape, suv, 31000, 2023).
14 vehicle(ford, f150, pickup, 40000, 2022).
15 vehicle(ford, mustang, sport, 48000, 2023).
16 vehicle(bmw, serie3, sedan, 45000, 2023).
17 vehicle(bmw, x5, suv, 65000, 2022).
18 vehicle(honda, civic, sedan, 23500, 2023).
19
20 add_vehicle(Brand0, Ref, Type0, Price, Year) :-
21     % Normalizar cadenas a minúsculas
22     downcase_atom(Brand0, Brand),
23     downcase_atom(Type0, Type),
24     % Evitar duplicados
25     \+ vehicle(Brand, Ref, _, _, _),
26     assertz(vehicle(Brand, Ref, Type, Price, Year)).
27

```

CONSULTAS BÁSICAS Y FILTROS

Esta sección incluye predicados que permiten realizar consultas al catálogo de vehículos filtrando por atributos como tipo, marca y presupuesto. Se utilizan herramientas clave de Prolog como `findall/3` y `bagof/3` para generar listas con los resultados deseados.

```
meet_budget(Ref, Type, BudgetMax) :-  
    vehicle(_, Ref, Type, Price, _),  
    Price <= BudgetMax.
```

¿Qué hace?

Filtrar los vehículos por su tipo (Type) y devolver sus referencias (Ref) si el precio está dentro del presupuesto máximo (BudgetMax).

¿Cómo funciona?

_ en la primera posición significa que no importa la marca.

Solo selecciona vehículos cuyo Price sea menor o igual al BudgetMax.

Ejemplo:

```
?- meet_budget(Ref, suv, 30000).
```

Ref = rav4.

```
refs_by_brand(Brand, Refs) :-  
    bagof(Ref, vehicle(Brand, Ref, _, _, _), Refs).
```

¿Qué hace?

Devuelve todas las referencias (Refs) asociadas a una marca específica (Brand).

¿Cómo funciona?

bagof/3 genera una lista agrupada de todos los valores de Ref que cumplen el patrón vehicle(Brand, Ref, _, _, _).

Si no hay resultados, falla (a diferencia de findall/3 que devolvería una lista vacía).

¿Qué es bagof/3?

bagof(Template, Goal, List):

Template: lo que se quiere recopilar (Ref).

Goal: la condición que deben cumplir los elementos.

List: variable donde se almacena el resultado.

Ejemplo:

```
?- refs_by_brand(toyota, L).
```

L = [corolla, camry, rav4, hilux].

```
vehicles_by_brand(Brand, List) :-  
    findall((Ref, Type, Price, Year),  
            vehicle(Brand, Ref, Type, Price, Year),  
            List).
```

¿Qué hace?

Devuelve una lista de tuplas con la información detallada de los vehículos pertenecientes a una marca específica.

¿Cómo funciona?

Usa `findall/3` para reunir todos los elementos que cumplen la condición.

¿Qué es `findall/3`?

`findall(Template, Goal, List):`

Siempre devuelve una lista, incluso si no hay resultados (retorna `[]`).

No agrupa automáticamente por valores, ni falla como `bagof/3`.

Ejemplo:

```
?- vehicles_by_brand(ford, L).  
L = [(focus, sedan, 21000, 2022),  
      (escape, suv, 31000, 2023),  
      (f150, pickup, 40000, 2022),
```

(mustang, sport, 48000, 2023)].

```

28 % PARTE 2: Consultas Básicas y Filtros
29
30 meet_budget(Ref, Type, BudgetMax) :-  

31   vehicle(_, Ref, Type, Price, _),  

32   Price <= BudgetMax.
33
34 % Lista todas las referencias (Ref) de una marca dada
35
36 refs_by_brand(Brand, Refs) :-  

37   bagof(Ref, vehicle(Brand, Ref, _, _, _), Refs).
38
39 % Detalles completos de vehículos de una marca
40 vehicles_by_brand(Brand, List) :-  

41   findall((Ref, Type, Price, Year),  

42     vehicle(Brand, Ref, Type, Price, Year),  

43     List).
44

```

⚙ **meet_budget**(Ref, suv, 30000).

↳ Singleton variables: [Brand,Type]
 ↳ Singleton variable in branch: Brand
 ↳ Singleton variable in branch: Type

Ref = rav4

Next	10	100	1,000	Stop
------	----	-----	-------	------

?- **meet_budget**(Ref, suv, 30000).

⚙ **refs_by_brand**(toyota, L).

↳ Singleton variables: [Brand,Type]
 ↳ Singleton variable in branch: Brand
 ↳ Singleton variable in branch: Type

L = [hilux]
 L = [corolla]
 L = [camry]
 L = [rav4]

?- **refs_by_brand**(toyota, L).

⚙ **vehicles_by_brand**(ford, L).

↳ Singleton variables: [Brand,Type]
 ↳ Singleton variable in branch: Brand
 ↳ Singleton variable in branch: Type

L =
 [(focus,sedan,21000,2022), (escape,suv,31000,2023), (f150,pickup,40000,2022), (mustang,sport,48000,2023)]

?- **vehicles_by_brand**(ford, L).

GENERACIÓN DE REPORTES

Esta parte del código se encarga de generar un reporte de vehículos basándose en los siguientes parámetros:

1. Marca del vehículo.
2. Tipo de vehículo (por ejemplo: sedan, SUV, etc.).
3. Presupuesto máximo por vehículo (es decir, el precio que una persona está dispuesta a pagar por un vehículo).
4. Presupuesto total máximo (el límite total que puede gastar en todos los vehículos seleccionados).

```
sum_list([], 0).  
  
sum_list([H|T], Sum) :-  
    sum_list(T, Rest),  
    Sum is H + Rest.
```

El predicado `sum_list` calcula la suma de una lista de números. En este caso, es utilizado para calcular el total de los precios de los vehículos seleccionados.

- Si la lista está vacía, la suma es 0.
- Si la lista tiene elementos, toma el primer elemento (H) y lo suma al resultado de sumar los elementos restantes (T).

```
sum_prices([], 0).sum_prices([(_, _, _, Price, _) | R], Total) :-
```

```
    sum_prices(R, Rest),
```

Total is Price + Rest.

El predicado `sum_prices` se usa para sumar los precios de los vehículos. Cada vehículo está representado por una tupla (Marca, Referencia, Tipo, Precio, Año), y el precio está en la cuarta posición de la tupla (Price).

El predicado recorre la lista de vehículos y suma los precios de todos los vehículos que quedan en la lista.

```
generate_report(Brand, Type, Budget, MaxTotal, (Selected, Total)) :-
```

```
    setof((P, Ref, Y), (vehicle(Brand, Ref, Type, P, Y), P <= Budget), Sorted),
```

```
    select_under_budget(Sorted, MaxTotal, [], SelRev, 0, Total),
```

```
    reverse(SelRev, Selected).
```

Este predicado genera el reporte con los vehículos seleccionados según los filtros definidos (marca, tipo, presupuesto y presupuesto total máximo).

1. `setof`:

- La primera parte utiliza `setof` para recopilar todos los vehículos que coinciden con la marca (Brand) y el tipo (Type), y cuyo precio (P) es menor o igual al presupuesto máximo (Budget).
- El resultado es una lista ordenada de tuplas (P, Ref, Y) con el precio (P), la referencia del vehículo (Ref) y el año (Y).
- Esta lista ordenada se guarda en la variable Sorted.

2. `select_under_budget`:

- Luego, el predicado `select_under_budget` selecciona los vehículos de la lista `Sorted` de acuerdo con el presupuesto total máximo (`MaxTotal`).
- A medida que recorre la lista de vehículos, va acumulando los precios en una suma (`Sum`). Si la suma no excede el `MaxTotal`, se agrega el vehículo a la lista de vehículos seleccionados (`SelRev`).
- Si la suma excede el presupuesto total máximo, deja de agregar vehículos a la selección.
- Al final, el predicado devuelve una lista de vehículos seleccionados en `SelRev` y el total de los precios acumulados en `Total`.

3. `reverse`:

- Como `select_under_budget` va agregando vehículos a la lista en orden inverso, se utiliza `reverse` para invertir la lista y devolver los vehículos en el orden en que fueron seleccionados.

El resultado final es una lista de vehículos seleccionados

```
select_under_budget([], _, Acc, Acc, Tot, Tot).
```

```
select_under_budget([(P,Ref,Y)|Rest], Max, Acc, Sel, Sum0, Sum) :-
```

 NewSum is Sum0 + P,

 (NewSum <= Max ->

```
        select_under_budget(Rest, Max, [(Brand,Ref>Type,P,Y)|Acc], Sel, NewSum, Sum)
```

 ; Sel = Acc, Sum = Sum0).

Este predicado recursivo selecciona los vehículos cuyo precio total acumulado no excede el MaxTotal.

- Caso base: Si la lista de vehículos está vacía (`[]`), se devuelve la lista acumulada Acc como resultado de los vehículos seleccionados (Sel), y el total acumulado Tot como el precio total de esos vehículos.
- Caso recursivo: Si hay vehículos en la lista, se toma el primer vehículo (P, Ref, Y) y se agrega su precio (P) al total acumulado (Sum0).

```

45 % PARTE 3: Generación de Reportes
46
47 sum_list([], 0).
48 sum_list([H|T], Sum) :-
49   sum_list(T, Rest),
50   Sum is H + Rest.
51
52 sum_prices([], 0).
53 sum_prices([(_, _, Price, _)|R], Total) :-
54   sum_ Uppercase atom due to var_prefix flag
55   Total is Price + Rest.
56
57 generate_report(Brand, Type, Budget, MaxTotal, (Selected, Total)) :-
58   setof((P,Ref,Y), (vehicle(Brand,Ref>Type, P, Y), P =< Budget), Sorted),
59   select_under_budget(Sorted, MaxTotal, [], SelRev, 0, Total),
60   reverse(SelRev, Selected).
61
62 select_under_budget([], _, Acc, Acc, Tot, Tot).
63 select_under_budget([(P,Ref,Y)|Rest], Max, Acc, Sel, Sum0, Sum) :-
64   NewSum is Sum0 + P,
65   ( NewSum =< Max ->
66     select_under_budget(Rest, Max, [(Brand,Ref>Type, P, Y)|Acc], Sel, NewSum, Sum)
67 ; Sel = Acc, Sum = Sum0 ) .
68

```

CASOS DE PRUEBAS

Por último, hablare de los casos de pruebas, estos casos son mencionados en la práctica y lo que buscan es validar y mostrar la capacidad del código para validar y mostrar correctamente el inventario de los vehículos según lo que se le especifique.

PREDICADOS

```
calculate_total_value_by_type(Type, Max, Total) :-  
    findall(Price, vehicle(_,_,Type,Price,_), List),  
    sum_list(List, Sum),  
    Sum =< Max,  
    Total = Sum.
```

Este predicado busca todos los precios de los vehículos del tipo dado (Type) usando `findall/3`.

Suma todos esos precios con `sum_list/2`.

Solo tiene éxito si la suma total es menor o igual al presupuesto máximo (Max).

Retorna esa suma total en Total.

Este predicado fallará si el total excede el presupuesto.

```
check_total_value_by_type(Type, Max) :-  
    findall(Price, vehicle(_, _, Type, Price, _), List),  
    sum_list(List, Sum),  
    ( Sum =< Max ->  
        format('Valor total para ~w: $~w (dentro de $~w).~n',[Type,Sum,Max])  
    ; format('Valor total para ~w: $~w (EXCEDE $~w)!~n',[Type,Sum,Max]) ).
```

Este predicado hace lo mismo que el anterior, pero siempre da un mensaje de salida, sin importar si el total excede o no.

Calcula la suma de los precios para vehículos del tipo Type.

Imprime:

Un mensaje si el total está dentro del presupuesto.

Otro si lo excede.

Casos de prueba

Ahora veamos cómo usar los predicados en consultas dadas por la práctica:

Case 1: Listar todos los Toyota SUV por debajo de \$30,000

?- vehicle(toyota, Ref, suv, Price, Year), Price =< 30000.

Esto buscará todos los vehículos marca Toyota, tipo SUV, con precio menor o igual a \$30,000.

Se mostrarán sus referencias (Ref), precios y años.

Case 2: Mostrar vehículos Ford agrupados por tipo y año usando bagof/3

?- bagof((Type, Year, Ref, Price), vehicle(ford, Ref, Type, Price, Year), List).

- bagof/3 agrupa todos los vehículos Ford en una lista donde se incluyen:
 - tipo de vehículo (Type)
 - año (Year)
 - referencia (Ref)
 - precio (Price)

Case 3: Calcular el valor total de los sedán sin pasar los \$500,000

?- check_total_value_by_type(sedan, 500000).

Esto imprimirá en consola el valor total de todos los vehículos tipo sedán y dirá si está dentro o excede el presupuesto.

Si solo quieres obtener el total y fallar si se pasa, puedes usar:

?- calculate_total_value_by_type(sedan, 500000, Total).

```

70 % PARTE 4: Casos de Prueba (Consultas a ejecutar)
71
72 calculate_total_value_by_type(Type, Max, Total) :-
73     findall(Price, vehicle(_, _, Type, Price, _), List),
74     sum_list(List, Sum),
75     Sum =< Max,
76     Total = Sum.
77
78 % Imprime mensaje según cumpla o exceda
79 check_total_value_by_type(Type, Max) :-
80     findall(Price, vehicle(_, _, Type, Price, _), List),
81     sum_list(List, Sum),
82     ( Sum =< Max -->
83         format('Valor total para ~w: $~w (dentro de $~w).~n',[Type,Sum,Max])
84     ; format('Valor total para ~w: $~w (EXcede $~w)!~n',[Type,Sum,Max]) ).
```

vehicle(toyota, Ref, suv, Price, Year), Price =< 30000.

Singleton variables: [Brand,Type]

Singleton variable in branch: Brand
Singleton variable in branch: Type

Price = 29500,

Ref = rav4,

Year = 2023

?- vehicle(toyota, Ref, suv, Price, Year), Price =< 30000.

bagof((Type, Year, Ref, Price), vehicle(ford, Ref, Type, Price, Year), List).



Singleton variables: [Brand,Type]

Singleton variable in branch: Brand
Singleton variable in branch: Type

List = [(sedan,2022,focus,21000), (suv,2023,escape,31000), (pickup,2022,f150,40000), (sport,2023,mustang,48000)]

?- bagof((Type, Year, Ref, Price), vehicle(ford, Ref, Type, Price, Year), List).

check_total_value_by_type(sedan, 500000).

Singleton variables: [Brand,Type]

Singleton variable in branch: Brand
Singleton variable in branch: Type

Valor total para sedan: \$139500 (dentro de \$500000).

true

?- check_total_value_by_type(sedan, 500000).