Bases de datos y estructuras - LISP

Databases and structures – LISP

Autor: Luis Fernando Martínez Muñoz
IS&C, Universidad Tecnológica de Pereira, Pereira, Colombia
Correo-e: f.martinez@utp.edu.co

***Resumen*— El presente artículo se ha escrito con el propósito de describir la construcción y el funcionamiento de las bases de datos en el lenguaje LISP.**

***Palabras clave*— LISP, p-lista, función, bases de datos.**

***Abstract*— This article has been written with the purpose of describing the construction and behavior of databases in LISP language.**

***Key Word*— LISP, p-list, function, database.**

I. CONTENIDO

A. Fundamento teórico

Las P-listas son listas que contienen pares de datos relacionados, en estos pares, el primer elemento funciona como una llave, y el segundo es el elemento relacionado a esa llave, su funcionalidad es similar a la de los diccionarios en Python.

```
(list :A 1 :B 2 :C 3)
```

Figura 1. Sintaxis de la construcción de una p-lista

Mediante la función "getf" se pueden extraer los elementos de la p-lista, enviándole la p-lista y alguna llave existente dentro de la p-lista, como se puede observar en el siguiente ejemplo:

```
CL-USER 1 > (print (getf (list :A 1 :B 2 :C 3) :A))

1
1
```

Figura 2. Función getf

Como se puede observar, se envían la p-lista y la llave del elemento que queremos obtener, es importante la sintaxis :nombre_de_la_llave, para que LISP la reconozca como una llave.

Gracias a la característica de relacionar elementos con llaves, se pueden usar las p-listas para crear registros, si se crea una función que reciba los campos del registro y los inserte en una p-lista con las llaves como nombres de los campos.

```
CL-USER 2 >(defun make-cd (title artist rating ripped)
(list :title title :artist artist :rating rating :ripped ri »
pped))
MAKE-CD

CL-USER 3 > (make-cd "Roses" "Kathy Mattea" 7 t)
(:TITLE "Roses" :ARTIST "Kathy Mattea" :RATING 7 :RIPPED T)
```

Figura 3. Creación de registros

En el ejemplo anterior se crea una función que creará una p-lista con los campos "título", "artista", "calificación", y estado "ripped" que indica la conversión a mp3.

En el llamado de la función se envían las instancias de un único registro, indicando el valor de cada uno de los campos, en el caso del ejemplo, se envía "Roses" como el valor del campo "título", "Katy Mattea" como el valor del campo "artista", 7 como la calificación, y t (true) como el valor del campo "ripped" que indica que está convertido a mp3.

Para crear una base de datos aprovechando el comportamiento de las p-listas, se debe declarar una variable global con el nombre que llevará la base de datos y asignarle un valor inicial, lo más recomendable y lógico es asignarle un valor nulo, ya que no tiene ningún registro dentro de ella.

```
(defvar *db* nil)
```

Figura 4. declaración del nombre de la base de datos

El nombre entre asteriscos *nombre* indica que dicha variable será global.

Ahora, para agregar registros a la base de datos, se aprovecha la funcionalidad de la función push, la cual "empuja" elementos dentro de una lista. Si la incluimos dentro de una función, podemos crear una función capaz de añadir registros a la base de datos.

```
CL-USER 5 > (defun add-record (cd) (push cd *db*))
ADD-RECORD

CL-USER 6 > (add-record (make-cd "Roses" "Kathy Mattear" »
7 t))
((:TITLE "Roses" :ARTIST "Kathy Mattear" :RATING 7 :RIPPE»
D T))

CL-USER 7 > (add-record (make-cd "Fly" "Dixie Chicks" 8 t»
))
((:TITLE "Fly" :ARTIST "Dixie Chicks" :RATING 8 :RIPPED T»
) (:TITLE "Roses" :ARTIST "Kathy Mattear" :RATING 7 :RIPP»
ED T))

CL-USER 8 > (add-record (make-cd "Home" "Dixie Chicks" 9 »
t))
((:TITLE "Home" :ARTIST "Dixie Chicks" :RATING 9 :RIPPED »
T) (:TITLE "Fly" :ARTIST "Dixie Chicks" :RATING 8 :RIPPED»
 T) (:TITLE "Roses" :ARTIST "Kathy Mattear" :RATING 7 :RI»
PPED T))
```

Figura 5 . Agregar registros a la base de datos

La función que se creó recibe como parámetro el elemento que se desea añadir a la base de datos y posteriormente lo añade, en este caso, se utiliza la función “make-cd”, que se había creado anteriormente, para enviarla como parámetro, es decir que a la final se envía un registro completo como elemento a añadir a la base de datos. Hay que tener en cuenta que la función “push” “empuja” los elementos hacia la primera posición de la lista.

A continuación, se desea observar el estado de la base de datos, para ello, se utiliza simplemente la función print con el nombre de la base.

```
CL-USER 9 > (print *db*)

((:TITLE "Home" :ARTIST "Dixie Chicks" :RATING 9 :RIPPED »
T) (:TITLE "Fly" :ARTIST "Dixie Chicks" :RATING 8 :RIPPED»
 T) (:TITLE "Roses" :ARTIST "Kathy Mattear" :RATING 7 :RI»
PPED T))
((:TITLE "Home" :ARTIST "Dixie Chicks" :RATING 9 :RIPPED »
T) (:TITLE "Fly" :ARTIST "Dixie Chicks" :RATING 8 :RIPPED»
 T) (:TITLE "Roses" :ARTIST "Kathy Mattear" :RATING 7 :RI»
PPED T))
```

Figura 6. Ver la base de datos

Evidentemente, se confirma que la base de datos ha recibido correctamente los registros, pero, la presentación de los datos con el comando “print” retorna una vista poco amigable de los datos, y puede llegar a ser confuso para aquel que no sea el diseñador de la base, y si la base tuviera muchos registros, sería confuso incluso para el diseñador y el programador.

Para solucionar este inconveniente, se le puede dar un formato a la impresión de los datos, como se ilustrará en la figura 7.

```
CL-USER 10 > (defun dump-db ()
(dolist (cd *db*)
(format t "~{~a:~10t~a~%~}~%" cd)))
DUMP-DB

CL-USER 11 > (dump-db)
TITLE:    Home
ARTIST:   Dixie Chicks
RATING:   9
RIPPED:   T

TITLE:    Fly
ARTIST:   Dixie Chicks
RATING:   8
RIPPED:   T

TITLE:    Roses
ARTIST:   Kathy Mattear
RATING:   7
RIPPED:   T

NIL
```

Figura 7. Función que da formato de impresión a la base de datos

Lo que hace la función “dolist” es iterar por cada elemento dado dentro de la base de datos y realizar una acción, en este caso se da formato a los elementos de los registros.

Existe una función que permite filtrar la impresión por criterios de cumplimiento, o sea, evalúa un criterio dado en cada registro y solo imprime aquellos que cumplan con dicho criterio.

```
CL-USER 12 > (print (remove-if-not
#'(lambda (cd) (equal (getf cd :artist) "Dixie Chicks")) »
*db*))

((:TITLE "Home" :ARTIST "Dixie Chicks" :RATING 9 :RIPPED »
T) (:TITLE "Fly" :ARTIST "Dixie Chicks" :RATING 8 :RIPPED»
 T))
((:TITLE "Home" :ARTIST "Dixie Chicks" :RATING 9 :RIPPED »
T) (:TITLE "Fly" :ARTIST "Dixie Chicks" :RATING 8 :RIPPED»
 T))
```

Figura 8. Filtrado de registros

La función “remove-if-not” crea una lista con los elementos de una lista que cumplen una condición específica. El #’ hace que se aplique la función a la lista.

En el caso del ejemplo, se crea una lista eliminando los registros de la base de datos db cuyo artista no sea “Dixie Chicks”, mejor dicho, crea una losta nueva con los cd del artista Dixie Chicks.

Ahora bien, se filtraron los datos, pero la función solo admite datos específicos, queriendo decir que es necesario escribir el criterio exacto para poder hacerlo. Para solucionar esto se puede crear una función que admita el criterio como parámetro, y así personalizar los filtros.

```
CL-USER 13 > (defun select-by-artist (artist)
(remove-if-not
#' (lambda (cd) (equal (getf cd :artist) artist)) *db*))
SELECT-BY-ARTIST

CL-USER 14 > (select-by-artist "Kathy Mattear")
((:TITLE "Roses" :ARTIST "Kathy Mattear" :RATING 7 :RIPPE»
D T))
```

Figura 9. Función select

De la forma ilustrada en la figura 9, se observa la creación de una función “select” que permite filtrar los registros por nombre del artista, como las funciones select en las bases de datos SQL.

B. Conclusión

Las p-listas, permiten la creación de bases de datos en el lenguaje LISP gracias a sus características de referenciación de datos, además con el uso de funciones propias del lenguaje se puede llegar a crear las funciones necesarias para la manipulación de datos dentro de las bases de datos.

REFERENCIAS

Vargas,G (2020). *Bases de datos y estructuras, LISP*. Pereira: Grupo ADA, Universidad Tecnológica de Pereira.