LOGALI

TEORÍA
Tablas internas

SAP ABAP Programación

Contenido

1. Estructuras

1.1. Definición

Las estructuras son variables complejas compuestas de componentes que a su vez pueden ser elementales o complejos.

La estructura es una unidad lógica, con un nombre que la identifica, y con la que se puede operar como si fuera una entidad única. Sin embargo, también podemos acceder a cada uno de los componentes de la estructura de forma individualizada.

Las tablas son un tipo particular dentro de las estructuras con algunas características que las hacen diferentes al resto. Una estructura, a la que llamaremos casa, se crea con la instrucción TYPES.

Por ejemplo:

```
TYPES: BEGIN OF casa,
         calle     TYPE string,
         numero TYPE n,
       END OF casa.
```

Una variable del tipo casa se declara con la instrucción DATA. Por ejemplo:

```
DATA mi_casa TYPE casa.
```

También podemos declarar una variable de tipo estructura llamada casa sin haber antes definido el tipo de estructura.

Por ejemplo:

```
DATA: BEGIN OF casa,
        calle     TYPE string,
        numero TYPE n,
```

```
END OF casa.
```

Los distintos componentes de una estructura están encadenados a través de punteros, y el orden en que están definidos en la declaración de tipos es el mismo en el que se va a almacenar la información en la memoria del sistema.

Una estructura es plana (flat) si contiene sólo variables elementales del tipo i, p, f, c, n, d, t, x.

Una estructura el profunda (deep) sí contiene entre sus componentes variables de tipo string (un string es realmente un puntero al primer carácter de una cadena), punteros o tablas.

Una variable del tipo tabla es realmente un puntero a la primera dirección de memoria de la tabla física.

1.2. Componentes

Para acceder al contenido de un componente de la estructura debemos escribir el nombre de la estructura y del componente separado por un guion, por ejemplo:

casa-calle….

1.3. Estructuras anidadas

Una estructura puede tener otras estructuras entre sus componentes. Varias estructuras anidadas componen una estructura plana si todos sus componentes son variables elementales. Si alguno de sus componentes es del tipo string, puntero o tabla, entonces la estructura será profunda.

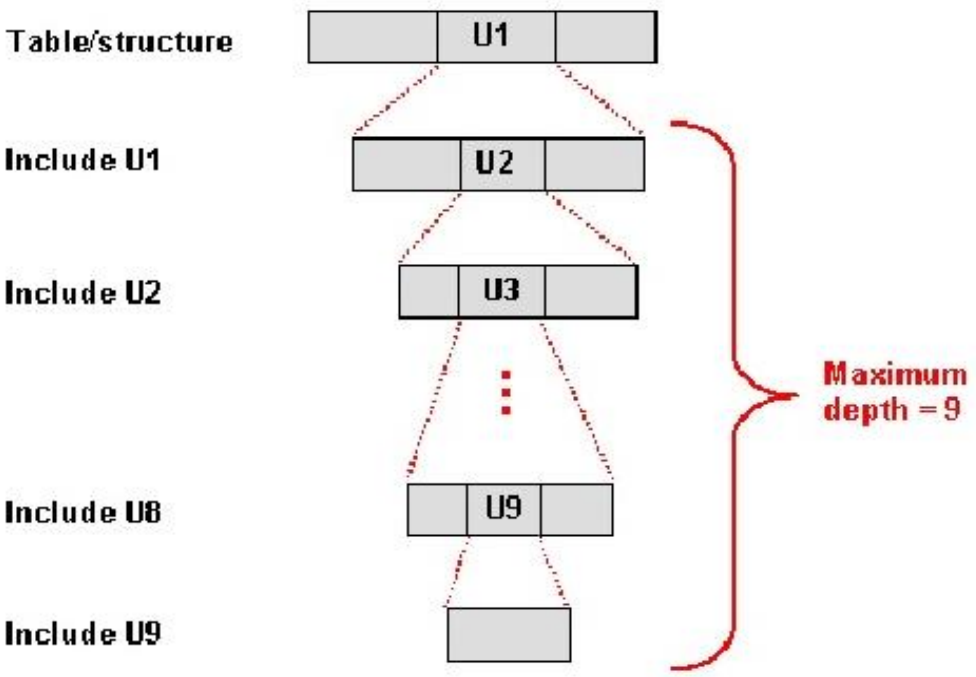
Veamos un ejemplo de estructura anidada:

```
TYPES: BEGIN OF poblacion,
        cod_postal TYPE n,
        ciudad(25) TYPE c,
```

```
        END OF poblacion.

TYPES: BEGIN OF domicilio,
          calle     TYPE string,
          numero TYPE n,
          ciudad   TYPE población,
       END OF domicilio.
```

En este caso el componente **domicilio-ciudad** es una estructura dentro de otra estructura. Veamos un ejemplo gráfico:



Para declarar una variable del tipo domicilio lo hacemos igual que para cualquier otra estructura:

DATA mi_domicilio **TYPE** domicilio.

Para acceder al contenido de un componente dentro de una estructura anidada tenemos que escribir el nombre de las dos estructuras y del componente separados por un guión, por ejemplo:

domicilio-ciudad-cod_postal

También podemos declarar una variable del tipo domicilio especificando todos sus componentes en la declaración de variables. Por ejemplo:

```
DATA: BEGIN OF mi_domicilio,
        calle     TYPE string,
        numero TYPE n,
       BEGIN OF mi_poblacion,
      cod_postal TYPE n,
      ciudad(25)  TYPE c,
       END OF mi_poblacion.
     END OF mi_domicilio.
```

Para asignar valores a cada uno de los componentes de la estructura lo podemos hacer a través del signo **=**. Por ejemplo:

mi_domicilio-calle = 'Calle Mayor'.

mi_domicilio-numero = '125'.

mi_domicilio-mi_población-cod_postal = '28056'.

mi_domicilio-mi_población-ciudad = 'Madrid'.

Para evitar este tipo de estructuras jerárquicas podemos incluir una estructura en otra estructura a través de la instrucción **INCLUDE**. En este caso todos los componentes de la estructura resultante estarían al mismo nivel. Sólo se pueden incluir estructuras planas a través de esta instrucción.

```
DATA: BEGIN OF casa.
```

```
      INCLUDE TYPE poblacion AS pob.
DATE: calle     TYPE string,
      numero TYPE n,
END OF casa.
```

En este caso los componentes de la población quedan incorporados a la estructura casa como un componente llamado **pob**.

Podemos operar con la estructura población como una unidad si escribimos su nuevo nombre. En el caso en que haya un conflicto de nombres podemos renombrar un campo añadiéndole un sufijo al nombre con la instrucción **RENAMING WITH SUFFIX**, por ejemplo:

```
INCLUDE STRUCTURE población AS pob RENAMIG WITH SUFFIX _pob.
```

```
casa-ciudad_pob = 'Madrid'.
```

Por último, para mostrar el contenido de un componente dentro de una estructura utilizamos la instrucción WRITE. Por ejemplo:

```
WRITE: / casa-calle,
       / casa-numero,
       / casa-cod_postal_pob,
       / casa-ciudad_pob.
```

2. Tablas Internas

2.1. Introducción

Hay dos maneras de procesar datos en ABAP, tablas internas o un extracto de datos. Todas las entradas de una tabla interna deben tener la misma estructura. Podemos acceder a una línea mediante una clave o un índice. En ABAP las tablas internas hacen la función de los ARRAYS bidimensionales de otros lenguajes de programación. El uso más frecuente de una tabla interna es almacenar en memoria los datos de una tabla de una base de datos durante la ejecución de un programa ABAP. Por lo tanto, las tablas internas son memoria dinámica, es decir, se crean en tiempo de ejecución del programa en la parte de memoria RAM asignada para este propósito (HEAP), y desaparecen una vez que se ha ejecutado el bloque o programa para el que se habían creado. Una tabla almacena un determinado número de líneas o registros del tipo declarado.

Las tablas planas (flat structures) son estructuras complejas compuestas por elementos más simples o campos (por ejemplo, variables de tipo i, p, c, o d). Una tabla profunda (deep) puede incluir a su vez estructuras complejas de forma recursiva, por ejemplo otras tablas.

2.2. Tipo de línea de una tabla interna

Suele ser normalmente una **estructura**, aunque podría ser cualquier tipo. Cada componente de la estructura es una columna de la tabla interna. El tipo puede ser elemental (c, d, f, i, n, p, t, x) o complejo (incluso otra tabla)

2.3. La clave

La clave identifica cada entrada de una tabla. Hay dos tipos de claves:

- clave estándar
- clave definida por el usuario.

El programador puede decidir si la clave es única (UNIQUE) o no (NON-UNIQUE). Si la clave es única no puede haber entradas duplicadas. La clave

de las tablas estándar NO puede ser única. La clave de las tablas hashed siempre es única. En todas las tablas con un tipo de línea estructurada, la clave estándar la forman todas las columnas (excepto si son punteros o tablas internas).

Los campos claves podrían incluso ser estructuras anidadas. Por defecto una línea con estructuras anidadas tiene una clave que va extendiéndose componente a componente dentro del anidamiento.

Si la tabla tiene una línea estructurada, la clave por defecto son todos los campos de tipo carácter (c, d, t, n, x, string, xstring). Si la tabla interna tiene como tipo de línea una tabla interna no tiene clave por defecto. Si la tabla tiene un tipo de línea no estructurada, la clave estándar es toda la fila. Para declarar una clave por defecto usamos la instrucción UNIQUE/NON-UNIQUE DEFAULT KEY. Una clave definida por el usuario puede contener cualquier columna que no contenga una tabla interna. No es obligatorio declarar una clave, en este caso el sistema selecciona una clave arbitrariamente.

2.4. Tablas internas genéricas:

No es necesario declarar completamente el tipo de una tabla interna. Una tabla interna se puede declarar como una tabla genérica. Por ejemplo, podemos no especificar la clave, o el tipo de línea. Podemos usar tablas internas genéricas para declarar el tipo de un field symbol, o los parámetros de un procedimiento, pero no para declarar un objeto (data object).

3. Tipos de tablas internas según el tipo de acceso

3.1. Acceso a través de un índice:

El índice se guarda en la variable de sistema **sy-tabix**.

Este es el modo más rápido de acceder a un registro, pero sólo podemos usarlo si conocemos el índice de un determinado registro en la tabla.

No todas las tablas tienen asignado un índice interno (por ejemplo, las tablas hashed).

3.2. Acceso a través de una clave:

La clave puede ser única o múltiple. Este es el modo universal en que podemos acceder a un registro determinado de cualquier tipo de tabla.

4. Tipos de tablas internas según la ordenación

4.1. Tablas estándar (Standard table):

Este es el tipo de tabla que se crea por defecto si no se indica otra cosa en el programa. Las líneas de este tipo de tabla están organizadas según un índice interno.

Para buscar un registro determinado podemos hacerlo por el índice o por la clave. En este último caso el sistema compara uno a uno **todos** los registros de la tabla hasta encontrar el o los que buscamos (búsqueda lineal).

La clave de una tabla estándar siempre es múltiple e incluye todos los campos no numéricos de dicha tabla. Esto significa que el sistema no comprueba si se repiten entradas en una tabla.

Este tipo de tabla ocupa menos memoria y se le pueden añadir registros muy rápidamente, pero es poco eficiente si necesitamos buscar registros con frecuencia (sobre todo si la base de datos tiene muchas entradas). El tiempo de búsqueda se incrementa linealmente con el número de registros.

Las tablas estándar se deben rellenar añadiendo líneas mediante la instrucción APPEND.

Preferiblemente deben leerse, modificar o borrar entradas especificando el índice mediante la instrucción INDEX.

En caso de usar la clave, no se deben hacer las adiciones de líneas y las búsquedas al mismo tiempo. Sería mejor añadir las líneas y luego ordenar

la tabla mediante la instrucción SORT. Una vez ordenada la tabla podemos hacer una búsqueda binaria usando la clave mediante la instrucción BINARY. Si la tabla tiene muchas entradas la búsqueda binaria es mucho más eficiente que la búsqueda lineal.

4.2. Tablas ordenadas (Sorted table):

Las líneas o registros tienen también asignado un índice interno, pero aparecen ordenados en orden ascendente de acuerdo con la clave.

La clave puede ser simple (UNIQUE) o múltiple (NON-UNIQUE).

El acceso a este tipo de tablas se puede hacer a través de un índice o a través de la clave. En este último caso el sistema utiliza un algoritmo de búsqueda binaria.

La modificación de este tipo de tabla es más lenta, ya que necesita encontrar el lugar que debe ocupar el nuevo registro, pero el acceso es mucho más rápido ya que el algoritmo de búsqueda va dividiendo la base de datos en dos hasta llegar al registro que buscamos.

El tiempo de búsqueda se incrementa de forma logarítmica a medida que añadimos más registros a la tabla.

Para insertar líneas en una tabla ordenada usamos la instrucción INSERT.

Si el nuevo registro de la tabla entra en conflicto con un registro existente (por ejemplo, si se repite la clave) se detecta al momento.

Las tablas ordenadas son especialmente útiles para procesar secuencias parciales de una tabla en un LOOP si especificamos el inicio de la clave mediante una condición en el WHERE.

4.3. Tablas hashed (Hashed table):

En este tipo de tabla no podemos acceder a una entrada mediante el índice. El orden de los registros de este tipo de tablas no se hace mediante un índice sino a través de un algoritmo de tipo numérico (hashed function) que calcula la posición de un determinado registro partiendo de una

determinada clave. La clave siempre tiene que ser única. La función o algoritmo hashed debe intentar evitar (o al menos gestionar de alguna manera) el problema de las colisiones, es decir el hecho de que dos valores distintos de la clave den como resultado una misma posición en la tabla.

También está el problema de la fragmentación de la memoria, es decir que la función hashed sea tal que nunca de como resultado una determinada posición de la tabla, y por lo tanto esa área de memoria quede en blanco.

La modificación de la tabla es más lenta que una tabla estándar ya que para cada nuevo registro el sistema tiene que calcular antes mediante la función hashed el lugar que debe de ocupar.

El tiempo de acceso a un determinado registro es constante (independiente del número de entradas que tenga la base de datos) y es siempre menor que el de los otros tipos de tablas. Tan sólo depende del algoritmo y de la rapidez del sistema, y suele ser muy pequeño, unos 17 nanosegundos.

Este tipo de tablas es el más adecuado si la acción más frecuente es acceder a una línea por la clave. Esto será así cuando queramos crear una tabla interna que se parezca a una tabla de la base de datos. También es el más adecuado si deseamos procesar grandes cantidades de datos.

5. Creación de tablas internas

Lo primero es definir la estructura de la tabla interna como un tipo de datos abstracto en el programa ABAP (mediante la instrucción TYPES) o en el Dictionary. Luego, basándonos en ese tipo de datos, creamos la tabla interna mediante la instrucción DATA. El tamaño de cada uno de los componentes de la cabecera de una tabla es de ocho bytes. El tamaño total de la tabla interna no está definido al declarar la tabla, puesto que los distintos registros de una tabla se añaden de forma dinámica (en tiempo de ejecución del programa).

Para declarar una estructura de una tabla:

```
TYPES: BEGIN OF tabla,
```

```
    Columna1 TYPE c,
    Columna2 TYPE c,
END OF tabla.
```

Para declarar un tipo de tabla interna (estándar/ordenada), con un tipo de línea como la tabla anterior, con la columna1 como clave única y un tamaño inicial de 100 líneas:

```
TYPES t1 TYPE STANDARD TABLE OF tabla INITIAL SIZE 100.

TYPES t2 TYPE SORTED TABLE OF tabla
        WITH UNIQUE KEY columna1 INITIAL SIZE 100.
```

Si declaramos un tamaño inicial de la tabla (en este caso 100 líneas) el sistema reserva espacio en memoria para ese tamaño. Cuando llegamos al final de las 100 líneas el sistema automáticamente reserva espacio en memoria para otras 100 líneas. Normalmente no es necesario declarar el tamaño inicial de la tabla. Para declarar una tabla interna hashed sobre un tipo existente en el Dictionary:

```
DATA t TYPE HASHED TABLE OF tabla
        WITH UNIQUE KEY columna1 INITIAL SIZE 100.
```

Se considera obsoleto definir una tabla con una cabecera (WITH HEADER LINE). Esto supone crear un nuevo objeto (header line-work area) con el mismo nombre y tipo que la tabla interna. Se usa como un área de trabajo cuando operamos con la tabla. Cuando se trabaja con cabeceras ABAP entiende que cuando nos dirigimos a una tabla en realidad nos dirigimos a la cabecera de la tabla. Para hacer saber al sistema que hacemos referencia al cuerpo de la tabla debemos escribir los paréntesis cuadrados detrás (por ejemplo tabla1[ ]) detrás del nombre de la tabla. Las tablas con otras tablas internas anidadas no permiten el uso de las cabeceras (header line).

6. Asignaciones y ordenaciones de tablas internas

Podemos copiar tablas internas de dos maneras:

MOVE tabla1 TO tabla2.

tabla2 = tabla1.

Estas instrucciones asignan el contenido completo de la tabla1 a la tabla2, incluyendo cualquier anidamiento. Los contenidos de la tabla2 se sobrescriben con los de la tabla1. Las dos tablas deben ser compatibles o bien convertibles entre sí. Si asignamos una tabla no ordenada a una tabla ordenada, los contenidos se ordenan automáticamente por la clave de la segunda.

7. Inicialización de una tabla interna

Podemos inicializar una tabla interna con la instrucción:

CLEAR tabla1.

Si queremos inicializar el cuerpo de una tabla con cabecera (header line-work area) debemos usar paréntesis:

CLEAR tabla1[].

Para asegurarse de que la tabla ha sido inicializada podemos usar la instrucción:

REFRESH tabla1.

Esto último siempre se realiza sobre el cuerpo de la tabla. Si queremos eliminar el requisito inicial de memoria que el sistema reserva por defecto para una tabla interna usamos la siguiente instrucción:

```
FREE tabla1.
```

Después de ejecutar esta instrucción de la tabla sólo queda la cabecera, que tan sólo ocupa 256 bytes. A medida que llenamos la tabla el sistema reserva más espacio para esta tabla. También podemos comprobar si la tabla está vacía con la instrucción **IS INITIAL** de la siguiente manera:

```
IF tabla1 IS INITIAL.
    WRITE 'la tabla está vacia'.
ENDIF.
```

8. Comparar tablas internas

Podemos usar tablas internas como operando en expresiones lógicas. Una de las expresiones lógicas más comunes es ver si dos tablas internas son iguales, por ejemplo:

```
IF tabla1 EQ tabla2.
   WRITE 'las dos tablas son iguales'.
ENDIF.
```

9. Ordenar tablas internas

Para ordenar una tabla por su clave:

```
SORT tabla1.
```

Por defecto las tablas se ordenan en orden ascendente. La clave por defecto de una tabla estándar se compone de los campos NO numéricos. La secuencia de los campos determina cómo se van a ordenar los registros. Para ordenar una tabla en orden descendente:

SORT tabla1 DESCENDING.

Podemos ordenar una tabla interna por uno o varios campos:

SORT tabla1 BY columna1 columna2.

Podemos ordenar una tabla por varios campos en orden inverso:

SORT tabla1 BY DESCENDIGN columna1 ASCENDING columna2.

Podemos ordenar una tabla por un campo que se determina dinámicamente:

DATA: a(15) TYPE c VALUE 'columna1'.

SORT tabla1 BY (a).

Si el campo es un puntero podemos usar el objeto al que apunta para ordenar la tabla:

SORT tabla1 BY puntero1->variable1.

Podemos ordenar una tabla por la clave y pedir que la tabla se ordene en orden alfabético:

SORT tabla1 AS TEXT.

Esto afecta el modo en que se ordenan los campos tipo carácter. Si no se indica nada el sistema ordena por el orden alfabético del código usado en

la plataforma (por ejemplo, ASCII). Si se indica este atributo, el sistema usa la lengua de SAP Logon (por ejemplo, inglés).

Si queremos cambiar la lengua de forma local para ese programa:

SET LOCALE LANGUAGE…

10. Tablas internas como parámetros:

Podemos pasar tablas internas a un procedimiento como parámetros por valor o por referencia. Podemos especificar parámetros en subrutinas y funciones como tablas internas con la instrucción TABLES. Esta instrucción define el parámetro como una tabla interna estándar con cabecera y la clave por defecto. Si le pasamos a la función una tabla sin cabecera el sistema la crea por defecto.

11. Utilizando los atributos de una tabla interna

Para saber los atributos de una tabla interna en tiempo de ejecución utilizamos la instrucción:

DESCRIBE TABLE tabla LINES x OCCURS n KIND clase.

El parámetro **LINES** asigna el número de líneas que tiene la tabla a la variable **x**.

El parámetro **OCCURS** asigna el tamaño inicial de la tabla a la variable **n**.

El parámetro **KIND** asigna el tipo de tabla a la variable **clase**.

12. Trabajando con una línea individual - work area

El principal uso de las tablas internas es cargar una gran cantidad de datos de la base de datos a la memoria interna del sistema para, de esta manera, trabajar rápido y eficientemente. Para acceder a una línea individual de una tabla podemos hacerlo según el índice interno o la clave. Las tablas estándar y las tablas ordenadas tienen un índice interno que nos indica el número de línea. Las tablas hashed no tienen índice. Para acceder a una línea en una tabla hashed debemos usar la clave. El acceso a través del índice siempre es el más rápido, puesto que la referencia del índice se guarda internamente en el sistema. Sin embargo, no siempre sabemos la asociación entre el contenido de una línea y el índice. Por eso la mayoría de los programas acceden a una tabla a través de una clave. Cuando accedemos a una línea individual la variable de sistema **sy-tabix** guarda el índice de la línea.

12.1. Tabla estándar

Podemos acceder a una línea por medio del índice o de la clave. Preferiblemente debemos acceder a través del índice. La clave nunca es única.

12.2. Tablas ordenadas (Sorted)

Podemos acceder a una línea por medio de una clave o de un índice. Preferiblemente debemos acceder a través de la clave. La clave puede ser única o no.

12.3. Tablas hashed

Podemos acceder a una línea tan sólo mediante la clave, que siempre es única.

13. Métodos de acceso a una línea individual

13.1. Área de trabajo

Un área de trabajo es una interfaz a la tabla interna, y debe ser compatible con el tipo de línea de la tabla interna. Al leer una entrada de una tabla, los datos se sobrescriben sobre los contenidos previos del área de trabajo. Estos datos se pueden usar en programas. Para escribir datos en una tabla interna primero se escriben en un área de trabajo y luego se transfieren a la tabla interna mediante la instrucción:

APPEND…

INSERT….

Para modificar datos mediante un área de trabajo usamos la instrucción **MODIFY**. Si la tabla interna tiene una cabecera, ésta se puede usar como área de trabajo.

13.2. Field Symbol:

Al usar un field symbol no es necesario copiar los datos a un área de trabajo. Simplemente asignamos la línea de la tabla interna al field symbol. El field symbol debería tener el mismo tipo que la tabla interna. Una vez asignado, trabajar con el field symbol es lo mismo que trabajar con la línea directamente. Cuando leemos entradas de una tabla, bien sea mediante la instrucción READ o mediante la instrucción LOOP, podemos asignarlas a un field symbol añadiendo ASSIGNING <f>, siendo f el nombre del field symbol. En este momento el field symbol apunta a la dirección de memoria que contiene la línea. El field symbol permite cambiar el contenido de una tabla directamente. Es importante NO cambiar el contenido de los campos claves de una tabla ordenada o hashed, lo que daría un error en tiempo de ejecución. El field symbol es más eficiente que un área de trabajo a la hora de modificar datos en tablas complicadas, especialmente si tienen otras tablas anidadas. A la hora de leer tablas internas mediante la instrucción READ usando un field symbol el sistema tiene primero que registrar la

asignación (es decir copiar la dirección de la tabla en el field symbol). Si borramos una línea a la que apunta un field symbol el sistema tiene primero que desasignarlo para evitar errores.

Para leer (READ) tablas cuya línea tenga una longitud de más de 1000 bytes, los field symbols resultan más eficientes que las áreas de trabajo.

Para modificar (MODIFY) tablas cuya línea tenga una longitud de más de 100 bytes los field symbols resultan más eficientes que las áreas de trabajo.

A la hora de procesar líneas de una tabla mediante la instrucción LOOP usando un field symbol el sistema no registra la asignación del field symbol a cada línea leída, sino que registra una asignación general a la tabla. Al terminar el loop el field symbol queda asignado a la última tabla leída. Cuando una tabla tiene más de 10 entradas resulta más eficiente usar un field symbol. Dentro del loop el field symbol no puede reasignarse a otra tabla, es decir, no podemos usar la instrucción ASSIGN dentro del loop para el mismo field symbol. Veamos un ejemplo:

```
TYPES: BEGIN OF fecha,
         mes TYPE i,
         dia  TYPE i,
       END OF fecha.

DATA: tabla TYPE SORTED TABLE OF fecha
            WITH UNIQUE KEY mes,
         edad  TYPE fecha. " work área

FIELD-SYMBOLS <f> LIKE LINE OF tabla.
```

```
DO 4 TIMES.

  edad-mes = sy-index.
  edad-dia   = sy-index.
  APPEND edad TO tabla.

ENDDO.

READ TABLE tabla WITH TABLE KEY mes = 2 ASSIGNING <f>.

IF <f> IS ASSIGNED.
  WRITE: / 'Asignado',
         / <f>-mes,<f>-dia.
ENDIF.

DELETE tabla INDEX 2.

WRITE / 'Registros que han quedado'.

LOOP AT tabla ASSIGNING <f>.
  WRITE: / <f>-mes,<f>-dia.
ENDLOOP.
```

14. Insertando líneas

Cuando queremos añadir una línea a una tabla hashed (o a una tabla genérica) debemos usar la instrucción:

INSERT…INTO TABLE….

Si añadimos la instrucción **INITIAL LINE** se inserta una línea con los valores por defecto para cada uno de los campos de la tabla. También usamos la instrucción INSERT cuando queremos insertar una línea en cualquier tipo de tabla. En el caso de las tablas estándar es lo mismo que si usamos el comando APPEND. Los campos de la línea deben ser compatibles. Si la tabla tiene una clave única no podemos insertar una línea con el valor de la clave repetida. Si la clave no es única los duplicados se insertan encima de la línea existente. Para insertar varias líneas usamos la instrucción:

INSERT LINES OF tabla1 FROM x TO y INTO TABLE tabla2.

Para indicar las líneas que se van a insertar debe ser una tabla indexada. Este método es más rápido que insertarlas en un loop. Podemos insertar varias líneas usando el índice:

INSERT LINES OF tabla1 **FROM** x **TO** y **INTO** tabla2 **INDEX** n.

Podemos insertar una línea usando el índice:

INSERT linea **INTO** tabla **INDEX** n.

Sin el añadido INDEX tan sólo podemos usar esta instrucción en un loop. La línea se inserta justo antes de la línea con el índice n. También podemos sustituir linea por **INITIAL LINE**.

15. Sumar líneas

Para crear tablas internas agregadas usamos la instrucción COLLECT.

COLLECT wa **INTO** tabla.

Siendo wa un área de trabajo. La tabla debe tener una estructura plana y los campos que no son claves deben ser numéricos (f, i, p). La instrucción COLLECT no añade una nueva línea, sino que añade los contenidos de los campos numéricos a los correspondientes campos de la tabla, siempre que la clave sea la misma.

Por ejemplo:

```
TYPES: BEGIN OF linea,
         col1(1) TYPE c,
         col2(1) TYPE c,
         col3 TYPE i,
       END OF linea.

DATA: tabla TYPE SORTED TABLE OF linea WITH NON-UNIQUE KEY col1 col2,
      wa    TYPE linea.
wa-col1 = 'a'.
wa-col2 = 'b'.
wa-col3 = 1.
COLLECT wa INTO tabla.

wa-col1 = 'a'.
wa-col2 = 'b'.
wa-col3 = 1.
COLLECT wa INTO tabla.

READ TABLE tabla INTO wa INDEX 1.
WRITE wa-col3.
```

En este caso la última instrucción mostrará el valor de 2.

16. Añadir líneas

Cuando queremos añadir una línea a una tabla indexada debemos usar la instrucción:

APPEND…TO.

Puesto que la línea se añade al final de la tabla no tiene que comprobar las demás líneas.

APPEND linea **TO** tabla**.**

Actualmente los índices de una tabla indexada se estructuran en forma de árbol para que el acceso a una determinada línea sea más eficiente. Podemos sustituir línea con la expresión **INITIAL LINE**, que añade una línea con los valores por defecto para cada campo. Después de insertar una nueva línea, la variable **sy-tabix** guarda el índice de la línea añadida. No se pueden añadir líneas con la clave repetida a tablas ordenadas con clave única o a tablas hashed.

Podemos añadir líneas de una tabla a otra:

APPEND LINES OF tabla1 **FROM** x **TO** y b tabla2.

Podemos añadir una línea y ordenar una tabla al mismo tiempo. Para ello primero creamos una tabla vacía mediante la instrucción INITIAL SIZE y luego añadimos una o varias líneas a la tabla y ordenamos esta última por un campo:

APPEND linea **TO** tabla **SORTED BY** campo.

Esta instrucción no se puede usar con tablas ya ordenadas (sorted). Por defecto la tabla se ordena en orden descendiente. La tabla interna no puede contener más entradas que las especificadas en la declaración inicial (INITIAL SIZE). Si la tabla tiene muchas líneas es preferible no usar esta instrucción.

17. Leer líneas de una tabla interna

Para leer una sola línea de una tabla usamos la instrucción **READ TABLE**. Si se encuentra una entrada para leer la variable de sistema **sy-subrc** guarda un **0**, si no guarda un **4**. La variable **sy-tabix** guarda e índice de la línea que encuentra.

Para leer de acuerdo con la clave de la tabla:

READ TABLE tabla **WITH TABLE KEY** k = x.

Si el valor x no es compatible con el campo el sistema lo convierte. También se puede buscar en una parte del campo mediante offset y length:

READ TABLE tabla **WITH TABLE KEY** k+2(3) = x.

En este caso el sistema busca 3 caracteres del campo k a partir del segundo carácter. La clave de búsqueda puede ser la clave de la tabla u otro campo cualquiera. En este último caso se puede especificar que la búsqueda sea binaria:

READ TABLE tabla **WITH KEY** k = x **BINARY SEARCH**.

Podemos especificar el nombre de la clave de forma dinámica:

n = k.

READ TABLE tabla **WITH KEY** (n) = x.

Podemos leer entradas de una tabla en un área de trabajo:

READ TABLE tabla **WITH KEY** k = x **INTO** wa.

Podemos comparar los campos de la tabla y del área de trabajo antes de leerlos:

READ TABLE tabla **INTO** wa **COMPARING** campo2.

En este caso el sistema compara el campo2 y guarda un **0** en la variable **sy-subrc**

si son iguales y un **2** si no son iguales.

También podemos comparar todos los campos:

READ TABLE tabla **INTO** wa **COMPARING ALL FIELDS**.

Podemos copiar tan solo algunos campos:

READ TABLE tabla **INTO** wa **TRANSPORTING** campo2.

Incluso podemos no copiar ningún campo:

READ TABLE tabla **TRANSPORTING NO FIELDS**.

En este caso el sistema tan sólo llena el contenido de las variables sy-tabix y sy-subrc.

Podemos usar un field symbol para leer una entrada de una tabla:

READ TABLE tabla WITH KEY k = x ASSIGNING <f>.

El field symbol <f> apunta a la línea leída.

También podemos usar el índice para leer una línea de una tabla:

READ TABLE tabla **ASSIGNING** <f> **INDEX** n.

Usar el índice para leer una tabla es más rápido que usar la clave.

18. Procesando tablas con loops

Podemos procesar entradas de una tabla mediante la instrucción:

LOOP AT

…

ENDLOOP.

Si no se especifica nada, ningún filtro, se procesan todas las entradas de la tabla. Las tablas indexadas se procesan según el índice. Las tablas hashed según el orden en que se insertaron. Podemos anidar varios loops. En un loop no podemos usar instrucciones que afecten a toda la tabla.

Para rellenar un área de trabajo mediante un loop:

LOOP AT tabla **INTO** wa.

Para asignar líneas a un field symbol mediante un loop:

LOOP AT tabla **ASSIGNING** <f>.

Si tan solo queremos usar el índice de una tabla y no los contenidos usamos la instrucción:

LOOP AT tabla **TRANSPORTING NO FIELDS**.

Podemos incluir condiciones lógicas en un loop:

LOOP AT tabla **WHERE**…

Podemos controlar el nivel de procesado dividiendo la tabla en grupos según el contenido de los campos, primero por la primera columna, luego la siguiente, etcétera. Para ello usamos la instrucción:

AT…**ENDAT**

y los modificadores:

- **FIRST** - primera línea de la tabla
- **LAST** - última línea de la tabla
- **NEW** nuevo grupo
- **END OF** final del grupo.

Por ejemplo, podemos dividir una tabla de alumnos de un colegio por clases:

LOOP AT tabla **INTO** linea.

AT NEW clase.

…

ENDAT.

ENDLOOP.

También podemos añadir una línea en blanco al final de un grupo:

AT END OF clase.

ULINE.

ENDAT.

Podemos sumar el contenido de unos campos numéricos:

AT END OF columna1.

SUM.

ENDAT.

Podemos usar el índice de una tabla en un loop:

LOOP AT tabla **FROM** x **TO** y.

19. Modificar líneas en una tabla interna

Para cambiar el contenido de una sola entrada usamos la instrucción:

MODIFY TABLE tabla **FROM** wa **TRANSPORTING** columna1.

El sistema busca en la tabla interna la línea cuya clave coincide con la del área de trabajo. El contenido de los campos que no son clave se copia en los correspondientes campos de la tabla. En este caso tan sólo se copia la columna1. El uso del añadido TRANSPORTING es opcional.

También podemos modificar varias líneas mediante una condición:

MODIFY tabla **FROM** wa **TRANSPORTING** cloumna1 **WHERE**…

En este caso el uso del añadido TRANSPORTING es obligatorio.

También podemos usar el índice:

MODIFY tabla **FROM** wa **INDEX** n.

En este caso el contenido del área de trabajo sustituye el contenido de la línea con el índice n. Es importante recordar que en una tabla ordenada no podemos cambiar el contenido de la clave.

20. Borrar líneas en una tabla interna

Podemos borrar una línea desde un área de trabajo:

DELETE TABLE tabla **FROM** wa.

En este caso se borrarán las entradas de la tabla cuya clave coincida con la clave del área de trabajo wa.

También podemos borrar una línea por la clave:

DELETE TABLE tabla **WITH TABLE KEY** k = x.

También podemos borrar una línea poniendo una condición (expresión lógica):

DELETE tabla **WHERE**…

Podemos borrar entradas duplicadas adyacentes:

DELETE ADJACENT DUPLICATE ENTRIES FROM tabla.

En este caso borramos aquellas entradas que tienen los mismos valores en los campos claves.

También podemos borrar entradas que tengan el mismo valor en los campos especificados mediante el añadido **COMPARING**:

DELETE ADJACENT DUPLICATE ENTRIES FROM table **COMPARING** campo1 campo2…

También podemos borrar entradas por el índice:

DELETE taba **INDEX** n.

Mediante el índice podemos borrar varias entradas:

DELETE tabla **FROM** x **TO** y.

21. Buscar entradas en una tabla interna

Podemos buscar una determinada palabra o cadena de caracteres en el contenido de una tabla:

FIND 'palabra' **IN TABLE** tabla **IN CHARACTER MODE**.

FIND 'palabra' **IN TABLE** tabla **IN BYTE MODE**.

También podemos buscar la primera vez que aparece la palabra:

FIND FIRST OCCURRENCE OF 'palabra' **IN TABLE** tabla.

O bien todas las veces que aparece esa palabra:

FIND ALL OCCURRENCES OF 'palabra' **IN TABLE** tabla.

Para buscar una palabra y reemplazarla por otra:

REPLACE 'palabra' **IN TABLE** tabla **WITH** 'nueva palabra'.

Aunque está obsoleto también podemos buscar una cadena de caracteres de la siguiente manera:

SEARCH tabla **FOR** 'abc'.

En este caso la variable de sistema sy-tabix muestra el índice de la línea donde se encuentra esa cadena de caracteres. Obviamente sólo funciona con tablas indexadas.

22. Trabajando con líneas de cabecera (header line)

Cuando creamos una tabla interna también podemos declarar una línea de cabecera con el mismo nombre que la tabla. Podemos usar esa línea de cabecera como área de trabajo para esa tabla. Aunque hoy en día se considera obsoleto, si trabajamos con cabeceras podríamos usar las siguientes instrucciones equivalentes (estas instrucciones asumen que la línea de cabecera es el área de trabajo):

Declaración de tabla interna sin línea de cabecera

DATA: tabla **TYPE TABLE OF**…,

wa **TYPE**

Declaración de tabla interna con línea de cabecera

DATA: tabla **TYPE TABLE OF** …. **WITH HEADER LINE**.

INSERT wa **INTO TABLE** tabla. - con work area

INSERT TABLE tabla. - con línea de cabecera

COLLECT wa **INTO** table.

COLLECT tabla.

READ TABLE tabla **INTO** wa.

READ TABLE tabla.

MODIFY TABLE tabla **FROM** wa.

MODIFY TABLE tabla.

MODIFY tabla **FROM** wa **WHERE**….

MODIFY tabla **WHERE**…

LOOP AT tabla **INTO** wa.

LOOP AT tabla.

APPEND wa **TO** tabla.

APPEND tabla.

INSERT wa **INTO** tabla.

INSERT tabla.

MODIFY tabla **FROM** wa.

MODIFY tabla.

El problema de estas instrucciones más cortas es que no se ve claramente el origen y destino de los datos. El hecho de que la línea de cabecera tenga el mismo nombre que la tabla crea confusión al trabajar con toda la tabla (por eso las áreas de trabajo deben tener un nombre distinto a la tabla).