SENTENCIAS

DEL

LENGUAJE

PROGRAMACION

ABAP /4

www.go.to/gaussr3

INSTRUCCIONES ABAP/4 sacado en:http://www.go.to/gaussr3

En esta página están todas las instrucciones ABAP/4 del sistema SAP R/3, además cada una de ellas contiene un pequeño ejemplo. Las sentencias están ordenadas alfabéticamente para una mejor búsqueda..

ADD

Definición
Añade el contenido del campo <n> sobre el campo <m>. Además del formato básico, la sentencia ADD tiene otra variante para la suma de campos adyacentes.
Sintaxis:
ADD <n> TO <m>.
ADD <n1> THEN <n2> UNTIL <nz> { GIVING | TO }<m>.
Suma la secuencia de campos <n1>, <n2>, ... , <nz>. <m>. <n1>, >n2>, ... , <nz> deben ser una secuencia de campos equidistantes en memoria del mismo tipo y longitud.

- Con la cláusula GIVING el campo <m> se actualiza con el resultado de la suma.
- Con la cláusula TO al campo <m> se le añade el resultado de la suma.

Ejemplo 1

```
DATA: NUMBER TYPE I VALUE 3,
    SUM    TYPE I VALUE 5.
ADD NUMBER TO SUM.
```

Ejemplo 2

```
DATA: BEGIN OF NUMBERS,
      ONE   TYPE P VALUE 10,
      TWO   TYPE P VALUE 20,
      THREE TYPE P VALUE 30,
      FOUR  TYPE P VALUE 40,
      FIVE  TYPE P VALUE 50,
      SIX   TYPE P VALUE 60,
    END   OF NUMBERS,
    SUM TYPE I VALUE 1000.
ADD NUMBERS-ONE THEN  NUMBERS-TWO
           UNTIL NUMBERS-FIVE GIVING SUM.
```

Errores

- BCD_BADDATA : El campo P tiene un formato BCD incorrecto.
  BCD_FIELD_OVERFLOW : El campo del resultado es demasiado pequeño.
  BCD_OVERFLOW : Overflow al hacer la operación aritmética .
  COMPUTE_INT_PLUS_OVERFLOW : Al añadir se ha producido un "Overflow" en un entero.

Vease también: COMPUTE, ADD-CORRESPONDING

ADD-CORRESPONDING

Definición
Añade el contenido de los componentes de un registro sobre otro.
Sintaxis:
ADD-CORRESPONDING <n> TO <m>.
Añade el contenido de los componentes del registro <n> sobre el registro <m>, para aquellos componentes que tengan el mismo nombre.
Ejemplo:

```
DATA: BEGIN OF VECTOR,
      X      TYPE I,
      Y      TYPE I,
      LENGTH TYPE I,
    END   OF VECTOR,
    BEGIN OF CIRCLE,
      VOLUME TYPE P
```

```
      Y      TYPE P,
      RADIUS TYPE I,
      X      TYPE I,
    END   OF CIRCLE.
...
ADD-CORRESPONDING VECTOR TO CIRCLE.
```

Vease también: ADD, MOVE-CORRESPONDING, SUBTRACT-CORRESPONDING, MULTIPLY-CORRESPONDING, DIVIDE-CORRESPONDING .

APPEND

Definición

Se utiliza para añadir registros en una tabla interna.

Sintaxis:

APPEND { <área-trabajo> TO | INITIAL LINE TO} <tabla>.

Esta sentencia añade una línea sobre la tabla <tabla>. La sentencia APPEND no comprueba duplicidades en la clave estándar. Después de cada sentencia APPEND, la variable del sistema SY-TABIX contiene el índice de la línea añadida.

- Con la cláusula <área-trabajo> TO se especifica el área de trabajo que queremos añadir. Con tablas internas con cabecera de línea se puede omitir esta cláusula, los datos serán leídos del área de trabajo de la tabla interna.
- En lugar de esta opción se puede utilizar INITIAL LINE TO, la cual añade una línea en la tabla interna con los valores iniciales correspondientes a cada componente de la tabla.

APPEND LINES OF <tabla1> { FROM <n1> } | {TO <n2>} TO <tabla2>.

Para añadir parte del contenido de una tabla interna (o toda la tabla) al final de otra tabla, se puede utilizar la sentencia APPEND con el formato anterior.

Después de ejecutarse la sentencia la variable del sistema SY-TABIX tiene el índice de la última línea añadida. Este método de añadir líneas de una tabla sobre otra es de tres a cuatro veces más rápido si lo hacemos línea a línea. Siempre que sea posible, es mejor utilizar este metodo.

- Si no se especifican las cláusulas FROM <n1> y TO <n2> la tabla <tabla1> se copia entera sobre la tabla <tabla2>. Con la cláusula FROM <n1> especificamos desde donde se copia la tabla. Con TO <n2> especificamos hasta dónde se copia la tabla. <n1> y <n2> son índices de la tabla interna.

APPEND { <área-trabajo> TO } <tabla> SORTED BY <campo>.

Con este formato de la sentencia las líneas de la tabla no son añadidas al final de la tabla, sino que son añadidas en la tabla <tabla> ordenadas por el campo <campo> de forma descendente. La tabla interna sólo puede contener las entradas especificadas en la cláusula OCCURS. Esto es una excepción a la regla general. Si se añade una línea y con ello rebasamos el límite definido, la última línea es eliminada. Por este motivo, esta sentencia para crear tablas *ranking* no muy grandes. Si queremos ordenar tablas de muchas entradas es preferible utilizar las sentencias SORT por razones de rendimiento.

Vease también: INSERT, MODIFY.

ASSIGN

Definición

Asigna con una referencia directa o dinámica el nombre del objeto que deseamos asignar a un campo.

Sintaxis:

Si conocemos el nombre del objeto de datos que deseamos asignar a un field-symbol antes de la ejecución del programa debemos realizar una asignación estática, es decir, con una referencia

directa al objeto de dato. Si no conocemos el objeto de dato hasta la ejecución del programa, la asignación deberá ser dinámica, es decir, con una referencia indirecta.
La sintaxis de la sentencia ASSIGN estática, sin especificación del offset, es la siguiente:
ASSIGN <campo> TO <FS> { <tipo> } { <decimales> }.
Después de la asignación, el field-symbol <FS> tiene los atributos del objeto de datos <campo>, y apunta a la misma posición de memoria. Las cláusulas <tipo> y <decimales> son opciones y las veremos en posteriores apartados.
La sintaxis de la sentencia ASSIGN estática, con especificación del offset, es la siguiente:
ASSIGN <campo> { +<offset> } { (<longitud>) } TO <FS> { <tipo> } { <decimales> }.
La única diferencia con la variante anterior es que en ésta especificamos el offset y la longitud del objeto de datos que hay que asignar. Las cláusulas <tipo> y <decimales> son opcionales y las veremos en posteriores apartados. <offset> y <longitud> pueden ser variables. El sistema no comprueba si la parte seleccionada en el campo, contando el offset y la longitud, es mayor que la longitud del campo. Se puede referenciar direcciones más allá de la longitud del campo, siempre y cuando no sobrepasemos el área de memoria asignada. Si no especificamos la longitud del campo con <longitud>, el field-symbol <FS> apunta al área perteneciente a los límites del campo. Si <offset> es mayor que 0, <FS> siempre apunta a un área perteneciente a los límites del campo. Si <offset> es menor que la longitud del campo, se puede especificar un asterisco (*) para <longitud>, para evitar sobrepasar el área asignada al campos. las cláusulas <tipo> y <decimales> son opcionales y las vceremos en posteriores apartados.
Si sólo en tiempo de ejecución conocemos el nombre del objeto de dato que hay que asignar al field-symbol debemos realizar una asignación dinámica. Para ello utilizaremos la sentencia ASSIGN con la siguiente sintaxis:
ASSIGN (<campo>) TO <FS> { <tipo> } { <decimales> }.
El nombre de objeto de dato que hay que asignar al field-symbol estará contenido en <campo> (asignación indirecta). En tiempo de ejecución, el sistema busca el campo referenciado siguiendo la siguiente secuencia:

- Si la asignación se realiza en una subrutina o módulo de función, el sistema busca el campo en la subrutina o módulo de función como objeto local.
- Si la asignación se realiza fuera de una subrutina o módulo de función, o el campo no se encuentra allí, el sistema busca el campo como objeto global al programa.
- Si el campo tampoco se encuentra como objeto global al programa, el sistema busca un área de trabajo declarada con la sentencia TABLES en el programa principal, o en el grupo de programas actual. La definición "grupo de programas" engloba un programa principal y todos los programa contenidos, incluyendo aquéllos en los que se realice una llamada a una subrutina.

Si la búsqueda del campo es satisfactoria y el campo puede ser asignado al field-symbol, el campo SY-SUBRC valdrá 0. En otro caso valdrá 4. Por razones de seguridad, se debe comprobar el valor de SY-SUBRC después de realizar la asignación para prevenir que un field-symbol apunte a un área indefinida. Las cláusulas <tipo> y <decimales> son opcionales y las veremos en posteriores apartados. Ya que este tipo de búsqueda tiene unos efectos adversos en cuanto al tiempo de respuesta del programa, debemos utilizar este tipo de asignación sólo si es absolutamente necesario. Si antes de la ejecución del programa sabemos que la asignación sólo será de áreas de trabajo, se puede utilizar la variante de la sentencia ASSIGN que veremos a continuación.
Si antes de la ejecución del programa sabemos que vamos a asignar un área de trabajo a un field-symbol, pero hasta la ejecución no sabemos de dicha área de trabajo se puede utilizar la siguiente variante dinámica de la sentencia ASSIGN:
ASSIGN TABLE FIELD (<campo>) TO <FS> { <tipo> } { <decimales> }.
El sistema busca el objeto de dato que vamos a asignar al field-symbol sólo en las áreas de trabajo declaradas con la sentencia TABLES en el programa principal de un grupo de programa. El sistema sólo realiza el paso 3 de la sentencia anteriormente vista. Si la búsqueda es satisfactoria y el campo puede ser asignado al field-symbol, SY-SUBRC es 0; en caso contrario es 4. Las cláusulas <tipo> y <decimales> son opcionales y las veremos en posteriores apartados.
Además de poder asignar objetos de datos a field-symbols, el sistema nos permite asignar un field-symbol sobre otro field-symbol. Para realizar esto se puede utilizar cualquier variante vista hasta el

momento de la sentencia ASSIGN, pero en lugar de utilizar un objeto de dato utilizaremos un field-symbol. Resumamos estas variantes a continuación:
ASSIGN (<FS1>) TO <FS2> { <tipo> } { <decimales> }.
ASSIGN (<FS1>) { +<offset> } { (<longitud>) }TO <FS2> { <tipo> } { <decimales> }.
ASSIGN (<campo>) TO <FS2> { <tipo> } { <decimales> }.
ASSIGN TABLE FIELD (<campo>) TO <FS> { <tipo> } { <decimales> }.
<campo> es un objeto de datos con el valor de un field-symbol. Las cláusulas <tipo> y <decimales> son opcionales y las veremos en posteriores apartados. Se puede especificar componentes de un field-string a un field-symbol con la siguiente variante de la sentencia ASSIGN:
ASSIGN COMPONENT <c> OF STRUCTURE <e> TO <FS> { <tipo> } { <decimales> }.
El sistema asigna el componente <c> de la estructura <e> al field-symbol <FS>. <c> puede ser un literal o una variable. Si la asignación es satisfactoria, SY-SUBRC es 0; en caso contrario es 4. Las cláusulas <tipo> y <decimales> son opcionales y las veremos en posteriores apartados.

- Se puede definir el tipo de un field-symbol utilizando la cláusula TYPE de la sentencia ASSIGN. La cláusula TYPE se puede utilizar con todas las variantes de la sentencia ASSIGN vistas anteriormente. <tipo> puede ser un literal a una variable. Se produce un error en tiempo de ejecución si el tipo de datos es desconocido, o si la longitud del tipo de datos especificado es incompatible con el tipo de dato asignado.
- También se puede especificar el número de decimales de un field-symbol si el campo asignado es del tipo P (empaquetado). Para ello se especifica la cláusula DECIMALS de la sentencia ASSIGN. La cláusula DECIMALS se puede utilizar con todas las variantes de la sentencia ASSIGN. Con esta cláusula se puede tener distinto número de decimales entre el objeto de dato asignado y el field-symbol. <decimales> puede ser un literal o una variable. Un erro en tiempo de ejecución se produce si <decimales> no tiene un valor comprendido entre 0 y 14, o si el objeto de dato asignado no es del tipo P.

Cuando trabajamos con subrutinas, se puede estar interesados en crear copias locales de datos globales sobre la pila de datos. Para realizar esto, disponemos de la siguiente cláusula de la sentencia ASSIGN:
ASSIGN LOCAL COPY OF ... TO <FS>.
El sistema es una copia del dato global especificado sobre la pila. En la subrutina, se puede acceder y cambiar esta copia sin cambiar el valor del dato global. Se puede utilizar esta cláusula con todas las variantes de la sentencia ASSIGN a excepción de la vista en el apartado "Sentencia ASSIGN con componentes de un field-string".

Ejemplo 1:

```
DATA NAME(4) VALUE 'JOHN'.
FIELD-SYMBOLS <F>.
ASSIGN NAME TO <F>.
WRITE <F>.
```

Salida: JOHN

Ejemplo 2:

```
DATA: NAME(12) VALUE 'JACKJOHNCARL',
    X(10)    VALUE 'XXXXXXXXXX'.
FIELD-SYMBOLS <F>.
ASSIGN NAME+4 TO <F>.
WRITE <F>.
ASSIGN NAME+4(*) TO <F>.
WRITE <F>.
```

Salida: JOHNCARLXXXX JOHNCARL

Ejemplo 3:

```
DATA SALES_DEC2(10) TYPE P DECIMALS 2 VALUE 1234567.
FIELD-SYMBOLS <SALES_DEC5>.

ASSIGN SALES_DEC2 TO <SALES_DEC5> DECIMALS 5.
WRITE: / SALES_DEC2,
    / <SALES_DEC5>.
```

Salida:

1,234,567.00
1,234.56700

Ejemplo 4:

```
DATA X(4) VALUE 'Carl'.
PERFORM U.
FORM U.
  FIELD-SYMBOLS <F>.
  ASSIGN LOCAL COPY OF X TO <F>.
  WRITE <F>.
  MOVE 'John' TO <F>.
  WRITE <F>.
  WRITE X.
ENDFORM.
```

Salida: Carl John Carl

Ejemplo 5:

```
DATA: NAME(4) VALUE 'XYZ',       XYZ VALUE '5'.
FIELD-SYMBOLS <F>.
ASSIGN (NAME) TO <F>.
WRITE <F>.
```

Salida: 5

Ejemplo 6:

```
TABLES TRDIR.
DATA NAME(10) VALUE 'TRDIR-NAME'.
FIELD-SYMBOLS <F>.
MOVE 'XYZ_PROG' TO TRDIR-NAME.
ASSIGN TABLE FIELD (NAME) TO <F>.
WRITE <F>.
```

Salida: XYZ_PROG

Ejemplo 7:

```
TABLES TRDIR.
DATA: F(8) VALUE 'F_global',
      G(8) VALUE 'G_global'.
MOVE 'XYZ_PROG' TO TRDIR-NAME.
PERFORM U.
FORM U.
  DATA: F(8)     VALUE 'F_local',
        NAME(30) VALUE 'F'.
  FIELD-SYMBOLS <F>.
  ASSIGN (NAME) TO <F>.
  WRITE <F>.
  MOVE 'G' TO NAME.
  ASSIGN (NAME) TO <F>.
  WRITE <F>.
  MOVE 'TRDIR-NAME' TO NAME.
  ASSIGN (NAME) TO <F>.
  WRITE <F>.
ENDFORM.
```

Salida: F_local G_global XYZ_PROG

Ejemplo 8:

```
PROGRAM P1MAIN.
  TABLES TRDIR.
  DATA NAME(30) VALUE 'TFDIR-PNAME'.
  FIELD-SYMBOLS <F>.
  MOVE 'XYZ_PROG' TO TRDIR-NAME.
  PERFORM U(P1SUB).
  ASSIGN (NAME) TO <F>.
```

```
  WRITE <F>.
  CALL FUNCTION 'EXAMPLE'.
PROGRAM P1SUB.
  TABLES TFDIR.
  ...
  FORM U.
    FIELD-SYMBOLS <F>.
    DATA NAME(30) VALUE 'TRDIR-NAME'.
    ASSIGN TABLE FIELD (NAME) TO <F>.
    WRITE <F>.
    MOVE 'FCT_PROG' TO TFDIR-PNAME.
  ENDFORM.
FUNCTION-POOL FUN1.
  FUNCTION EXAMPLE.
    DATA NAME(30) VALUE 'TRDIR-NAME'.
    FIELD-SYMBOLS <F>.
    ASSIGN (NAME) TO <F>.
    IF SY-SUBRC = 0.
      WRITE <F>.
    ELSE.
      WRITE / 'TRDIR-NAME cannot be accessed'.
    ENDIF.
  ENDFUNCTION.
```

Salida: XYZ_PROG FCT_PROG
TRDIR-NAME no se tiene acceso.

Ejemplo 9:

```
PROGRAM P1MAIN.
  TABLES TRDIR.
  DATA NAME(30) VALUE 'TFDIR-PNAME'.
  FIELD-SYMBOLS <F>.
  MOVE 'XYZ_PROG' TO TRDIR-NAME.
  CALL FUNCTION 'EXAMPLE'.
FUNCTION-POOL FUN1.
  FUNCTION EXAMPLE.
    DATA NAME(30) VALUE 'TRDIR-NAME'.
    FIELD-SYMBOLS <F>.
    ASSIGN LOCAL COPY OF MAIN
      TABLE FIELD (NAME) TO <F>.
    IF SY-SUBRC = 0.
      WRITE <F>.
    ELSE.
      WRITE / 'TRDIR-NAME cannot be accessed'.
    ENDIF.
  ENDFUNCTION.
```

Salida: XYZ_PROG

Ejemplo 10:

```
PROGRAM P1MAIN.
  DATA: BEGIN OF REC,
          A VALUE 'a',
          B VALUE 'b',
          C VALUE 'c',
          D VALUE 'd',
        END OF REC,
        CN(5) VALUE 'D'.
  FIELD-SYMBOLS <FS>.
  DO 3 TIMES.
```

```
    ASSIGN COMPONENT SY-INDEX OF
        STRUCTURE REC TO <FS>.
    IF SY-SUBRC <> 0. EXIT. ENDIF.
    WRITE <FS>.
  ENDDO.
  ASSIGN COMPONENT CN OF STRUCTURE REC TO <FS>.
  WRITE <FS>.
```

Output: a b c d

Vease también: DESCRIBE FIELD, MOVE.

AT .. ENDAT

Definición

La sentencia AT <fg> .. ENDAT sólo puede ser utilizada dentro de un bucle LOOP .. ENDLOOP para extractos, e identifica un bloque de proceso. La sentencia se ejecuta cuando se detecta que cambia algún valor en el field-group.

Sintaxis:

AT <zfg> { WITH <fg1> }.
<bloque-sentencias>
ENDAT.

La setencia AT <fg> se cierra con ENDAT, identificando de esta forma un bloque de proceso.

- La cláusula WITH se utiliza para indicar que la sentencia AT se debe ejecutar si para el field-group <fg> le sigue el field-group <fg1>.

Ejemplo 1:

```
DATA: NAME(30),
      SALES TYPE I.
FIELD-GROUPS: HEADER, INFOS.
INSERT: NAME  INTO HEADER,
        SALES INTO INFOS.
...
LOOP.
  AT NEW NAME.
    NEW-PAGE.
  ENDAT.
  ...
  AT END OF NAME.
    WRITE: / NAME, SUM(SALES).
  ENDAT.
ENDLOOP.
```

Vease también: LOOP, EXTRACT.

AT END OF .. ENDAT

Definición

La sentencia AT END OF .. ENDAT sólo puede ser utilizada dentro de un bucle LOOP .. ENDLOOP e identifica un bloque de proceso. La sentencia se ejecuta cuando se detecta que cambia algún valor para el campo especificado.

Sintaxis:

AT END OF <campo>.
<bloque-sentencias>
ENDAT.

La sentencia AT END OF se cierra con ENDAT, identificando de esta forma un bloque de proceso. En un bloque AT END OF .. ENDAT el área de trabajo no se rellena con la línea actual de la tabla interna. Todos los campos que no forman parte de la clave estándar de la tabla toman el valor inicial. Para la condición de línea END OF <campo> el sistema sobrescribe todos los campos de la clave estándar, que se encuentran a la derecha del campo <campo> con asterisco (*). No se debe utilizar la sentencia AT END OF .. ENDAT en combinación con la sentencia LOOP .. ENDLOOP con las cláusulas FROM, TO o WHERE.

Ejemplo 1:

```
DATA: BEGIN OF COMPANIES OCCURS 20,
        NAME(30),
        PRODUCT(20),
        SALES TYPE I,
      END   OF COMPANIES.
...
LOOP AT COMPANIES.
  AT NEW NAME.
    NEW-PAGE.
    WRITE / COMPANIES-NAME.
  ENDAT.
  WRITE: / COMPANIES-PRODUCT, COMPANIES-SALES.
  AT END OF NAME.
    SUM.
    WRITE: / COMPANIES-NAME, COMPANIES-SALES.
  ENDAT.
ENDLOOP.
```

Vease también: LOOP.

AT FIRST .. ENDAT

Definición

La sentencia AT FIRST .. ENDAT sólo puede ser utilizada dentro de un bucle LOOP .. ENDLOOP e identifica un bloque de proceso. La sentencia se ejecuta con el primer valor o primer registro de la sentencia LOOP .. ENDLOOP.

Sintaxis:

```
AT FIRST.
<bloque-sentencias>
ENDAT.
```

La sentencia AT FIRST se cierra con ENDAT, identificando de esta forma un bloque de proceso. En un bloque AT FIRST .. ENDAT el área de trabajo no se rellena con la línea actual de la tabla interna. Todos los campos que no forman parte de la clave estándar de la tabla toman el valor inicial. Para la condición de línea FIRST el sistema sobrescribe todos los campos de la clave estándar, que se encuentran a la derecha del campo <campo> con asterisco (*). No se debe utilizar la sentencia AT FIRST .. ENDAT en combinación con la sentencia LOOP .. ENDLOOP con las cláusulas FROM, TO o WHERE.

Ejemplo 1:

```
DATA: BEGIN OF COMPANIES OCCURS 20,
        NAME(30),
        PRODUCT(20),
        SALES TYPE I,
      END   OF COMPANIES.
...
LOOP AT COMPANIES.
  AT FIRST.
    NEW-PAGE.
    WRITE / COMPANIES-BUKRS.
```

```
  ENDAT.
  WRITE: / COMPANIES-PRODUCT, COMPANIES-SALES.
ENDLOOP.
```
Vease también: LOOP.

AT LAST .. ENDAT

Definición

La sentencia AT LAST .. ENDAT sólo puede ser utilizada dentro de un bucle LOOP .. ENDLOOP e identifica un bloque de proceso. La sentencia se ejecuta con el último valor o último registro de la sentencia LOOP .. ENDLOOP.

Sintaxis:

```
AT LAST.
<bloque-sentencias>
ENDAT.
```

La sentencia AT LAST se cierra con ENDAT, identificando de esta forma un bloque de proceso. En un bloque AT LAST .. ENDAT el área de trabajo no se rellena con la línea actual de la tabla interna. Todos los campos que no forman parte de la clave estándar de la tabla toman el valor inicial. Para la condición de línea LAST el sistema sobrescribe todos los campos de la clave estándar, que se encuentran a la derecha del campo <campo> con asterisco (*). No se debe utilizar la sentencia AT LAST .. ENDAT en combinación con la sentencia LOOP .. ENDLOOP con las cláusulas FROM, TO o WHERE.

Ejemplo 1:

```
DATA: BEGIN OF COMPANIES OCCURS 20,
        NAME(30),
        PRODUCT(20),
        SALES TYPE I,
      END   OF COMPANIES.
...
LOOP AT COMPANIES.
  AT NEW NAME.
    NEW-PAGE.
  ENDAT.
  WRITE: / COMPANIES-PRODUCT, COMPANIES-SALES.
  AT LAST.
    SUM.
    WRITE: / COMPANIES-NAME, COMPANIES-SALES.
  ENDAT.
ENDLOOP.
```
Vease también: LOOP.

AT LINE-SELECTION

Definición

Para permitir al usuario seleccionar una línea de un listado y realizar alguna acción, se puede escribir un bloque de proceso en el programa para el evento AT LINE-SELECTION.

Sintaxis:

```
AT LINE-SELECTION..
<bloque-de-proceso>
ENDAT.
```

Este evento define un bloque de proceso que se activa cuando seleccionamos una línea del informe. Este evento se utiliza en los informes interactivos cuando generan salidas secundarias.

Este evento se describe con mayor detalle en el Capítulo 15: "Programas interactivos". Si no definimos una interfaz particular para el listado (a través de la transacción "menu painter") el sistema crea una interfaz estándar. La apariencia de esta interfaz es la misma que la de cualquier informe, interactivo o no. El usuario podrá activar este evento a través de las siguientes acciones:

- En el menú "edición", se elige la opción "seleccionar".
- Presionando la tecla de función F2.
- Realizando un doble-click sobre la línea o un solo click sobre un *hotspot* (ver las opciones sentencia WRITE).

después de posicionar el cursor sobre una línea y realizar alguna de las acciones anteriores, el evento AT LINE-SELECTION se activa.
Internamente, el código de función PICK activa el evento AT LINE-SELECTION. En la interfaz predefinida, la opción de menú "edición" -> "seleccionar" y la tecla de función F" están asignadas al código de función PICK.
Ejemplo 1:

```
DATA TEXT(20).

START-OF-SELECTION.
  PERFORM WRITE_AND_HIDE USING SPACE SPACE.

AT LINE-SELECTION.
  CASE TEXT.
    WHEN 'List index'.
      PERFORM WRITE_AND_HIDE USING 'X' SPACE.
    WHEN 'User command'.
      PERFORM WRITE_AND_HIDE USING SPACE 'X'.
    WHEN OTHERS.
      SUBTRACT 2 FROM SY-LSIND.
      PERFORM WRITE_AND_HIDE USING SPACE SPACE.
  ENDCASE.
  CLEAR TEXT.

FORM WRITE_AND_HIDE USING P_FLAG_LSIND P_FLAG_UCOMM.
  WRITE / 'SY-LSIND:'.
  PERFORM WRITE_WITH_COLOR USING SY-LSIND P_FLAG_LSIND.
  TEXT = 'List index'.
  HIDE TEXT.
  WRITE / 'SY-UCOMM:'.
  PERFORM WRITE_WITH_COLOR USING SY-UCOMM P_FLAG_UCOMM.
  TEXT = 'User command'.
  HIDE TEXT.
  IF SY-LSIND > 0.
    WRITE / 'PICK here to go back one list level'.
  ENDIF.
ENDFORM.

FORM WRITE_WITH_COLOR USING P_VALUE
                            P_FLAG_POSITIVE.
  IF P_FLAG_POSITIVE = SPACE.
    WRITE P_VALUE COLOR COL_NORMAL.
  ELSE.
    WRITE P_VALUE COLOR COL_POSITIVE.
  ENDIF.
ENDFORM.
```

Vease también: HIDE, WINDOW, SCROLL LIST.

AT NEW .. ENDAT

Definición

La sentencia AT NEW .. ENDAT sólo puede ser utilizada dentro de un bucle LOOP .. ENDLOOP e identifica un bloque de proceso. La sentencia se ejecuta cuando se detecta que para el campo especificado se va a producir un nuevo valor

Sintaxis:

AT NEW <campo>.
<bloque-sentencias>
ENDAT.

La sentencia AT NEW se cierra con ENDAT, identificando de esta forma un bloque de proceso. En un bloque AT .. ENDAT el área de trabajo no se rellena con la línea actual de la tabla interna. Todos los campos que no forman parte de la clave estándar de la tabla toman el valor inicial. Para la condición de línea NEW el sistema sobrescribe todos los campos de la clave estándar, que se encuentran a la derecha del campo <campo> con asterisco (*). No se debe utilizar la sentencia AT .. ENDAT en combinación con la sentencia LOOP .. ENDLOOP con las cláusulas FROM, TO o WHERE.

Ejemplo 1:

```
DATA: BEGIN OF COMPANIES OCCURS 20,
        NAME(30),
        PRODUCT(20),
        SALES TYPE I,
      END   OF COMPANIES.
...
LOOP AT COMPANIES.
  AT NEW NAME.
    NEW-PAGE.
    WRITE / COMPANIES-NAME.
  ENDAT.
  WRITE: / COMPANIES-PRODUCT, COMPANIES-SALES.
ENDLOOP.
```

Vease también: LOOP.

AT PFnn

Definición

Para permitir que el usuario realice una acción tras pulsar una tecla de función, se puede escribir un bloque de proceso en el programa, encabezado por el evento ATPFnn.

Sintaxis:

AT PFnn
<bloque-de-proceso>

Este evento define un bloque de proceso que se activa cuando pulsamos la tecla de función PFnn, siendo "nn" un valor comprendido entre 1 y 24. Este evento se utiliza en los informes interactivos cuando generan salidas secundarias. Este evento se describe con mayor detalle en el Capítulo 15: "Programas interactivos".

De igual forma que en los eventos anteriores, si no definimos una interfaz de usuario particular para el programa, el sistema incorpora una estándar. Si el usuario presenta una tecla de función el sistema procesará el bloque de proceso de esa tecla de función (si existe). La posición del cursor no es relevante. Para ver una lista de tecla de función predefinidas se puede crear un programa que tenga un evento AT PFnn, ejecutar el listado, y sobre éste pulsar el botón derecho del ratón. Aparecerá un listado con todas las teclas de función definidas con un texto. Recomiendo no utilizar este evento, es preferible utiliza el evento AT USER-COMMAND. Hace más amigable la interfaz del usuario el uso de botones que el uso de teclas de función.

Ejemplo 1:

```
DATA NUMBER LIKE SY-INDEX.
```

```
START-OF-SELECTION.
  DO 9 TIMES.
    WRITE: / 'Row', (2) SY-INDEX.
    NUMBER = SY-INDEX.
    HIDE NUMBER.
  ENDDO.

AT PF8.
  CHECK NOT NUMBER IS INITIAL.
  WRITE: / 'Cursor was in row', (2) NUMBER.
  CLEAR NUMBER.
```

Vease también: HIDE, SET PF-STATUS.

AT SELECTION-SCREEN

Definición

El evento AT SELECTION-SCREEN provee de varias posibilidades de bloques de proceso pero todos relacionados con la pantalla de selección. Algunos se ejecutan antes de la pantalla de selección y otros después.

Sintaxis:

AT SELECTION-SCREEN

Si utilizamos la palabra clave sin usar ninguna opción, el bloque de proceso correspondiente se ejecuta después de que el sistema procese la pantalla de selección. Si durante la ejecución del bloque de proceso se activa algún mensaje de error, vuelve a aparecer la pantalla de selección. Todos los campos de la pantalla de selección se pueden modificar. La sentencia MESSAGE activa mensajes. Los tipo de error pueden ser A (*abend*), E (Error), I (Informativo), S (siguiente pantalla) o W (*Warning*). EL grupo de mensajes se especifica en la sentencia REPORT (Cláusula MESSAGE-ID ...).

AT SELECTION-SCREEN ON <parámetro>.

La siguiente variante nos permite crear un bloque de proceso para un solo campo de la pantalla de selección. <parámetro> debe ser una parámetro. El bloque de proceso se arranca cuando el sistema ha procesado el campo en cuestión. El sistema realiza primero validaciones de formato. Una vez analizado el formato, se arranca el bloque de proceso. Si activamos un mensaje de error en este bloque de proceso, vuelve a aparecer la pantalla de selección y sólo se puede modificar el campo tratado.

AT SELECTION-SCREEN ON END OF <criterio>.

La siguiente variante nos permite validar un criterio de selección de la pantalla de selección. El bloque de proceso se ejecuta después de introducir valores en la pantalla "compleja" de introducción de datos en un criterio de selección.

AT SELECTION-SCREEN ON VALUE-REQUEST FOR <campo>.

La siguiente variante permite crear un bloque de proceso asociado cuando el usuario pulsa el botón de posibles valores (también se activa pulsando la tecla de función F4). Dicho botón aparece automáticamente a la derecha del campo (parámetro o criterio de selección) cuando se utiliza este evento. En el bloque de proceso se debe programar una lista de valores de proceso.

AT SELECTION-SCREEN ON HELP-REQUEST FOR <campo>.

La siguiente variante permite crear un bloque de proceso asociado a la tecla de función F1 (ayuda en el estándar). Normalmente lo que codifica en el bloque de proceso es una ventana de ayuda.

AT SELECTION-SCREEN ON RADIOBUTTON GROUP <botón>.

La siguiente variante nos permite asociar un bloque de proceso a un radiobutton (grupo de botones). El bloque proceso se activa después de que el sistema procesa el radiobutton definido en <botón>. Si activamos un mensaje de error en el bloque de proceso vuelve a aparecer la pantalla de selección. Sólo el radiobutton relacionado puede ser modificado.

AT SELECTION-SCREEN ON BLOCK <bloque>.

La siguiente variante nos permite activar un bloque de proceso cuando el sistema termina de procesar un bloque. Recordemos que en un bloque de pantalla de selección se define con la sentencia SELECTION-SCREEN. Si activamos un mensaje de error en el bloque de proceso aparece de nuevo la pantalla de selección. Sólo los campos del bloque relacionado se pueden modificar.

AT SELECTION-SCREEN OUTPUT.

La siguiente variante nos permite activar un bloque de proceso antes de que el sistema muestre la pantalla de selección (parte PBO de la lógica de proceso). Este bloque de proceso se puede utilizar, por ejemplo, para mover valores a los campos de la pantalla de selección, pero hay que tener en cuenta que cada vez que se procesa la pantalla de selección (puede haber mensajes de error en otros eventos AT SELECTION-SCREEN) se procesa este evento. En cierta ocasiones puede ser más recomendable utilizar el evento INITIALIZATION.

Ejemplo 1:

```
SELECT-OPTIONS NAME FOR SY-REPID MODIF ID XYZ.
...
AT SELECTION-SCREEN OUTPUT.
  LOOP AT SCREEN.
    CHECK SCREEN-GROUP1 = 'XYZ'.
    SCREEN-INTENSIFIED = '1'.
    MODIFY SCREEN.
  ENDLOOP.
```

Vease también: PARAMETERS, SELECT-OPTIONS.

AT USER-COMMAND

Definición

Para permitir que el programa reaccione a las funciones que el usuario active se utiliza el evento AT USER-COMMAND.

Sintaxis:

AT USER-COMMAND.
<bloque-de-proceso>

Este evento define un bloque de proceso que se activa cuando seleccionamos un comando. Los comandos se pueden seleccionar a través de los botones proporcionados en el programa, o a través del campo OK-CODE. Este evento se utiliza en los informes interactivos cuando generan salidas secundarias.

El bloque de proceso del evento AT USER-COMMAND se ejecuta cuando el usuario activa un código de función presente en el status activo en ese momento. Este evento no se activa por los códigos de función predefinido del sistema o por el código de función PICK que, como ya hemos visto, activa el evento AT LINE-SELECTION. El campo del sistema SY-UCOMM nos permite saber qué código de función ha activado el usuario.

Ejemplo 1:

```
DATA: NUMBER1 TYPE I VALUE 20,
      NUMBER2 TYPE I VALUE  5,
      RESULT  TYPE I.

START-OF-SELECTION.
  WRITE: / NUMBER1, '?', NUMBER2.

AT USER-COMMAND.
  CASE SY-UCOMM.
    WHEN 'ADD'.
      RESULT = NUMBER1 + NUMBER2.
    WHEN 'SUBT'.
      RESULT = NUMBER1 - NUMBER2.
```

```
  WHEN 'MULT'.
    RESULT = NUMBER1 * NUMBER2.
  WHEN 'DIVI'.
    RESULT = NUMBER1 / NUMBER2.
  WHEN OTHERS.
    WRITE 'Unknown function code'.
    EXIT.
ENDCASE.
WRITE: / 'Result:', RESULT.
```

Vease también: SCROLL LIST.

AUTHORITY-CHECK

Definición

Esta sentencia nos permite comprobar las autorizaciones de un usuario. Recordemos que dichas autorizaciones forman parte del perfil de usuario y el sistema las guarda en el maestro de usuarios.

Sintaxis:

```
AUTHORITY-CHECK OBJECT '<objeto>'
ID 'campo1> { FIELD <valor1> | DUMMY }
ID 'campo2> { FIELD <valor2> | DUMMY }
...
ID 'campon> { FIELD <valorn> | DUMMY }
```

<objeto> es el nombre del objeto de autorización que hay que comprobar.

- Un objeto de autorización esta compuesto de campos de autorización, todos ellos deben estar especificados a continuación de las cláusulas ID (<campo1>, <campo2> ..., <campon>).
- <valor1, <valor2>, ..., <valorn> son los valores de autorización que hay que comprobar. <valorx> puede ser un literal o una variable. El sistema busca en el perfil del usuario el objeto especificado en la sentencia, y comprueba si el usuario tiene autorización para todos los campos del objeto.
- Se puede saltar la comprobación de un campo sustituyendo la cláusula FIELD por DUMMY. Si SY-SUBRC vale 0, el usuario esta autorizado, en caso contrario, SY-SUBRC toma un valor distinto de 0.

Ejemplo 1:

```
AUTHORITY-CHECK OBJECT 'M_EINF_WRK'
  ID 'WERKS' FIELD '0002'
  ID 'ACTVT' FIELD '02'.

AUTHORITY-CHECK OBJECT 'M_EINF_WRK'
  ID 'WERKS' DUMMY
  ID 'ACTVT' FIELD '01'.
```

BACK

Definición

Para situar la línea de salida en la línea siguiente de la cabecera de página se puede utilizar la sentencia BACK.

Sintaxis:

```
BACK
```

Esta sentencia puede ser utilizada en combinación con la sentencia RESERVE. Si la sentencia BACK se utiliza sola, el sistema actualiza la línea actual de salida a la siguiente cabecera de página. Además actualiza la variable SY-COLNO a 1 y SY-LINNO al valor que corresponda en

función del número de líneas de cabecera. En combinación con la sentencia RESERVE se aplican otras condiciones. El sistema actualiza la línea actual de salida a la primera línea del bloque de líneas creado con la sentencia RESERVE.

Ejemplo 1:

```
DATA:  TOWN(10)      VALUE 'New York',
       CUSTOMER1(10) VALUE 'Charly',
       CUSTOMER2(10) VALUE 'Sam',
       SALES1 TYPE I VALUE 1100,
       SALES2 TYPE I VALUE 2200.
RESERVE 2 LINES.
WRITE:  TOWN, CUSTOMER1,
     /      CUSTOMER2 UNDER CUSTOMER1.
BACK.
WRITE: 50 SALES1,
       /  SALES2 UNDER SALES1.
```

Vease también: RESERVE.

BREAK

Definición

Con esta sentencia activamos la transacción de depuración de programas.

Sintaxis:

BREAK <usuario>.

Cuando esta sentencia se ejecuta, si el usuario que está ejecutando el programa es el especificado en el literal <usuario>, se activa la transacción de depuración de programas. <usuarios> no debe ir entre comillas como sería lógico en un literal. El efecto es el mismo que el de la sentencia BREAK-POINT, para un mayor detalle ver esta sentencia.

Vease también: BREAK-POINT.

BREAK-POINT

Definición

Sentencia utilizada para activar la transacción de depuración de programas.

Sintaxis:

BREAK-POINT <campo>.

La sentencia BREAK-POINT interrumpe el proceso y activa el depurador de programa en ese punto. Una vez activo el depurador se pueden realizar todas las funciones que permite el sistema de debugging. Si el sistema es incapaz de activar el modo depuración, por ejemplo, si el programa se está ejecutando en fondo (background) o si el programa es de actualización, el sistema genera un mensaje en el log del sistema. El contenido de <campo>, para ejecuciones en background o en procesos de actualización, se graba en el mensaje del log del sistema.

Después de procesar el sistema la sentencia BREAK-POINT se realiza automáticamente un COMMIT-WORK de la base de datos.

Vease también: BREAK.

CALL

Definición

Permite realizar una llamada a una función del sistema.

Sintaxis:

CALL <función> [ID <id1> FIELD <c1> ... ID <idn> FIELD <cn>].

La función <función> debe existir en el fichero sapactab.h. Si cambiamos o creamos nuevas funciones del sistema tendremos que compilar y linkar de nuevo el kernel de SAP, para ello sería necesario tener los códigos fuente de la aplicación SAP.

- La cláusula ID nos permite pasar parámetros por referencia. <id1> ... <idn> son los nombres de los parámetros de la función del sistema, <c1> ... <cn> son los campos (o literales) con los valores que hay que pasar.

Esta sentencia fue creada para uso exclusivo de los programa estándar de SAP R/3.
Vease también: CALL FUNCTION, CALL CUSTOMER-FUNCTION, CALL DIALOG.

CALL CUSTOMER-FUNCTION

Definición
Esta sentencia se utiliza para llamar a módulos de función, desarrollados por el usuario, que pueden o no estar activos.
Sintaxis:
CALL CUSTOMER-FUNCTION <función>.
Esta sentencia la utiliza el sistema de manera estándar para desarrollar el concepto de "ampliación". La idea es la siguiente, en los programas estándar del sistema, en aquellos puntos donde se ha considerado interesante, se ha incluido esta sentencia con llamadas a módulos existentes pero sin desarrollar y sin activar. El usuario puede incluir en estos módulos el desarrollo necesario para las especificaciones concretas de un sistema y activar posteriormente el módulo. De esta forma se evita que se modifiquen los programas estándar (como ha sucedido en tantas ocasiones en el sistema R/2 que no acepta este concepto).
El nombre del módulo de función tiene la siguiente nomenclatura: EXIT_XXXXXXXX_NNN, siendo XXXXXXXX el nombre del programa que realiza la llamada y NNN el número de función especificado en la llamada: <función>.
Aunque la sentencia no esta limitada al uso interno de SAP (como sucede con otras sentencias) realmente no veo la necesidad de utilizar esta sentencia. Si necesitamos crear un módulo de función os recomiendo que utilicéis CALL FUNCTION.
Vease también: CALL FUNCTION, CALL DIALOG, CALL.

CALL DIALOG

Definición
La sentencia CALL DIALOG llama a un módulo de diálogo.
Sintaxis:
CALL DIALOG <módulo-diálogo>
  [AND SKIP FIRST SCREEN]
  [EXPORTING <f1> [FROM <g1>] ... <fn> [FROM <gn>]]
  [IMPORTING <f1> [TO <g1>] ... <fn> [TO <gn>]]
  [USING <tabla> MODE <modo>].
<módulo-diálogo> puede ser un literal o una variable.

- Con la cláusula AND SKIP FIRST SCREEN el sistema procesa la primera pantalla del módulo de diálogo en background, si hemos rellenado todos los campos obligatorios, gracias a la sentencia SET PARAMETERS.
- Con la cláusula EXPORTING especificamos todos los objetos de datos (campos, registros, tablas) que se pasan al módulo de función. Si el objeto de dato en el módulo de diálogo y en el programa coinciden en el nombre, la opción FROM no es necesaria. Si no coinciden, <fi> apunta a los objetos de datos del módulo de diálogo y <gi> a los objetos de datos del programa.

- Con la cláusula IMPORTING especificamos todos los objetos de datos (campos, registros, tablas) que se devuelven desde el módulo de función. Si el objeto de dato en el módulo de diálogo y en el programa coinciden en el nombre, la opción TO no es necesaria. Si no coinciden, <fi> apunta a los objetos de datos del módulo de diálogo y <gi> a los objetos de datos del programa.
- El campo SY-SUBRC se exporta y se importa automáticamente. Los objetos de datos export o import desconocidos se ignoran en el módulo de función. Los objetos de datos pasados deberían tener el mismo tipo o estructura en el módulo de diálogo y en el programa.

CALL DIALOG <módulo-diálogo> USING <tabla> [MODE <modo>].
Esta sentencia permite implementar el método de batch-input CALL DIALOG USING. La sentencia llama al módulo de diálogo <módulo-diálogo> y le pasa la información, en formato de juego de datos, en la tabla interna <tabla>. Al igual que en la sentencia CALL TRANSACTION USING, el sistema rellena ciertas variables del sistema con información referente a un mensaje.

- La cláusula MODE tiene el mismo significado que en la sentencia CALL TRANSACTION USING.
- Los códigos de retorno devueltos por la sentencia son los mismos que en la sentencia CALL TRANSACTION USING. A diferencia de la anterior sentencia, CALL DIALOG USING no realiza COMMIT de la base de datos. Los errores en tiempo de ejecución que se pueden producir son los siguientes:
  CALL_DIALOG_NOT_FOUND -> El módulo de diálogo no existe.
  CALL_DIALOG_WRONG_TDCT_MODE -> El módulo de diálogo contiene errores.
  CALL_DIALOG_NAME_TOO_LONG -> El nombre de algún parámetro es más largo que el permitido.

Vease también: CALL FUNCTION, CALL CUSTOMER-FUNCTION, CALL.

CALL FUNCTION

Definición
La sentencia CALL FUNCTION se utiliza para llamar a un módulo de función. Esta sentencia tiene varias variantes en función de la tarea de trabajo que procesa el módulo de función.
Sintaxis:

```
CALL FUNCTION <función>
  [ EXPORTING <p1> = <f1> ... <pn> = <fn>  ]
  [ IMPORTING <p1> = <f1> ... <pn> = <fn>  ]
  [ TABLES      <p1> = <tabla1> ... <pn> = <tablan>  ]
  [ CHANGING <p1> = <f1> ... <pn> = <fn>  ]
  [ EXCEPTIONS <p1> = <exc1l> ... <pn> = <excn> [ OTHERS = <nn>] ].
```

Llamada básica a un módulo de función. Llama al módulo de función <función>. <función puede ser un literal o una variable. La asignación de los nombre entre el programa y el módulo de función se realiza por el nombre, no por la secuencia de los parámetros.

- Con la cláusula EXPORTING se puede pasar campos, field-strings o tablas al módulo de función. Estos parámetros se deben declarar en el módulo de función como parámetros IMPORT. Cuando se realiza la llamada debemos asignar valores a todos los parámetros que no se les haya definido un valor por defecto.
- La cláusula IMPORTING se utiliza para que el módulo de función devuelva al programa valores en campos, field-strings o tablas internas. Estos parámetros deben estar declarados como tablas en el módulo de función. Cuando llamamos al módulo de función debemos indicar todas las tablas a las que no se les haya asignado una tabla por defecto.
- Con la cláusula CHANGING se puede pasar valores en campos. field-strings o tablas internas al módulo de función, que podrán ser modificados en el módulo de función y devueltos al programa.

- Con la cláusula EXCEPTIONS especificamos las excepciones que queremos considerar en la llamada al módulo. Las excepciones se activan con las sentencias RAISE y MESSAGE (con la cláusula RAISING).

Una llamada a un módulo de función puede generar los siguientes mensajes de error:

- CALL_FUNCTION_NOT_FOUND -> Módulo de función desconocido.
- CALL_FUNCTION_NO_VB -> Sólo módulos de función de actualización pueden ser llamados desde la tarea de actualización.
- CALL_FUNCTION_NO_ACTIVE -> Módulo de función creado pero no activo.
- CALL_FUNCTION_PARM_MISSING -> El módulo de función espera un parámetro que no ha sido utilizado en la llamada.
- CALL_FUNCTION_CONFLICT_LENG -> Parámetro con longitud errónea.
- CALL_FUNCTION_CONFLICT_TYPE -> Parámetro con tipo erróneo.
- CALL_FUNCTION_CONFILCT_GEN_TYPE -> El tipo de parámetro actual no satisface los requerimientos del parámetro definido en el módulo de función.
- CALL_FUNCTION_WRONG_ALIGNMENT -> El parémetro actual no satisface los requerimientos del parámetro definido en el módulo de función.
- CALL_FUNCTION_BASE_LITL -> Se ha pasado un literal en un parámetro estructurado.

CALL FUNCTION <función> STARTING NEW TASK <tarea>
[DESTINATION <destino>]
[DESTINATION IN GROUP { <grupo> | DEFAULT }]
[PERFORMING <rutina> ON END OF TASK]
[EXPORTING <p1> = <f1> ... <pn> = <fn>]
[TABLES <p1> = <tabla1> ... <pn> = <tablan>]
[EXCEPTIONS <p1> = <e1> MESSAGE <m1> ... <pn> = <en> MESSAGE <mm>].

El sistema arranca el módulo de función asíncronamente en un nuevo modo. En contraste con el formato básico, esta variante hace que el sistema continué con la ejecución del programa aunque el módulo de función no haya terminado. Las cláusula opcionales son las siguientes:

- Con la cláusula DESTINATION el módulo de función se ejecuta externamente como una "Remote Function Call" (RFC). <destino> puede ser una variable o un literal.
- Con la cláusula DESTINATION IN GROUP el módulo de función se ejecuta en todos los servidores definidos en un grupo. Los módulos de función se ejecutarán en paralelo.
- La cláusula PERFORMING nos permite controlar los errores que se puedan producir en la ejecución del módulo de función. <rutina> determina la rutina a ejecutar.
- La cláusula EXPORTING se utiliza para pasar parámetros al módulo de función, de la misma forma que en la variante 1.
- La cláusula TABLES se utiliza para pasar tablas internas al módulo de función, de la misma forma que en la variante 1.
- La cláusula EXCEPTIONS nos permite manejar dos excepciones del sistema (en ambos casos se puede utilizar la opción MESSAGE para activar un mensaje):
  SYSTEM_FAILURE -> Si el sistema destino no funciona.
  COMMUNICATION_FAILURE -> Si no se puede conectarnos o comunicarnos con el sistema destino.

Para poder utilizar esta variante de la sentencia CALL FUNCTION, tanto el sistema servidor como el sistema cliente deben estar en versión 3.0 o superior.

CALL FUNCTION <función> IN UPDATE TASK
[EXPORTING <p1> = <f1> ... <pn> = <fn>]
[TABLES <p1> = <tabla1> ... <pn> = <tablan>]

Con esta variante el módulo de función se ejecutará en la tarea de actualización. La ejecución no es inmediata, los parámetros traspasados con las cláusulas EXPORTING y TABLES se almacenan en la base de datos. En el siguiente COMMIT WORK causa que el módulo de función se ejecute en la tarea de actualización. Las cláusulas EXPORTING y TABLES tienen el mismo significado que en la variante 1.

CALL FUNCTION <función> DESTINATION <destino>
[EXPORTING <p1> = <f1> ... <pn> = <fn>]
[IMPORTING <p1> = <f1> ... <pn> = <fn>]
[TABLES <p1> = <tabla1> ... <pn> = <tablan>]
[CHANGING <p1> = <f1> ... <pn> = <fn>]
[EXCEPTIONS <p1> = <exc1l> ... <pn> = <excn> [OTHERS = <nn>]].

El módulo de función se ejcuta desde una fuente externa, a través de una llamada "Remote Function Call" (RFC). <destino> puede ser un literal o una variable. Dependiendo del destino especificado el módulo de función podrá ser ejecutado en un sistema R/3 o un sistema R/2. <destino> es una clave que permite al sistema saber qué sistema remoto estamos asignado. Los destinos se definen en la transacción SM59. La forma de llegar a través de los menús es la siguiente: (Pantalla inicial de SAP) -> *Herramientas -> Gestión; Gestión -> Red Destinos RFC*.
Hay ciertos destinos que tienen un significado especial:
NONE -> Este destino apunta al propio sistema donde se realiza la llamada al módulo de función.
BACK -> Este destino fuerza a que el programa que realiza la llamada al módulo de función tenga que ser llamado desde un sistema diferente. Si el programa se ejecuta desde el mismo sistema se produce la excepción COMMUNICATION_FAILURE.
Las cláusulas opcionales tienen el mismo significado que el de la variante 1 (llamada básica).

CALL FUNCTION <función> IN BACKGROUND TASK
[AS SEPARATE UNIT]
[DESTINATION <destino>]
[EXPORTING <p1> = <f1> ... <pn> = <fn>]
[TABLES <p1> = <tabla1> ... <pn> = <tablan>]

Llamada al módulo de función de modo asíncrono. Con la cláusula AS SEPARATE UNIT el módulo de función se ejecuta en una nueva LUW. El resto de las cláusulas ya han sido comentadas anteriormente.
Vease también: CALL DIALOG, CALL CUSTOMER-FUNCTION, CALL.

CALL METHOD

Definición
Llama a un método de un objeto externo. Utilizado para la programa OLE2.
Sintaxis:

CALL METHOD OF <objeto> <método> [= <f>]
[EXPORTING <p1> = <f1> ... <pn> = <fn>]
[NO FLUSH]

Llama al método <método> del objeto <objeto>. <método> puede ser un literal o una variable.

- En <f>, si se especifica, el sistema guarda el código de retorno de la sentencia.
- Con la cláusula EXPORTING se especifican parámetros a pasar al método.
- Con la cláusula NO FLUSH continuá con el proceso aunque la siguiente sentencia no sea una sentencia OLE.

CALL SCREEN

Definición
Se utiliza para llamar a una dynpro.
Sintaxis:

CALL SCREEN <dynpro> [STARTING AT <x1> <y1> [ENDING AT <x2> <y2>]]

Llama al dynpro <dynpro>. <dynpro> es un número de pantalla del programa principal. Para abandonar la pantalla se deben utilizar las sentencias SET SCREEN o LEAVE SCREEN.

- Con la cláusula STARTING/ENDING definimos el tamaño de la ventana. <x1> y <y1> definen la esquina superior izquierda de la ventana, <x2> y <y2> definen la esquina inferior derecha de la ventana. Si no se utiliza la cláusula ENDING la esquina inferior derecha será el límite de la pantalla. Se puede producir el siguiente error en tiempo de ejecución:

DYNP_TOO_MANY_CALL_SCREENS -> Hemos alcanzado el límite de pantallas abiertas para un programa. El límite actualmente es de 50 pantallas.

Vease también: CALL SUBSCREEN.

CALL SUBSCREEN

Definición

Una subscreen es una pantalla independiente que se muestra en un área de otra pantalla principal.

Sintaxis:

CALL SUBSCREEN <área> INCLUDING <programa> <pantalla>.

La sentencia se debe utilizar en ambos procesos PBO y PAI.

Vease también: CALL SCREEN.

CALL TRANSACTION

Definición

La sentencia CALL TRANSACTION permite ejecutar una transacción.

Sintaxis:

CALL TRANSACTION <transaccion> [AND SKIP FIRST SCREEN].

Esta sentencia llama a la transacción <transacción>. <transacción> puede ser un literal o una variable. Para retornar desde la transacción al programa se utiliza la sentencia LEAVE PROGRAM.

- Con la cláusula AND SKIP FIRST SCREEN además de llamar a la sentencia indicada se puede saltar la primera pantalla de la transacción. Esta acción debe ir combinada con el uso de parámetros SPA/GPA de los campos de la primera pantalla de la transacción (sentencia SET PARAMETERS) para que el momento de la ejecución de la transacción, ésta tome los parémtros de la memoria (sentencia GET PARAMETERS) y así, de esta forma, poder saltar la primera pantalla. Si para pasar la primera pantalla se necesita rellenar parñametros que no indicamos, el sistema presenta la primera pantalla y será el usuario el que realiza la entrada manualmente.
- Esta sentencia tiene una variante con la cláusula USING que cambia totalmente el significado de la sentencia. Se utiliza como método de batch-input.

Sentencia utilizada para implementar el método CALL TRANSACTION.

CALL TRANSACTION <transacción> USING <tabla>
[MODE <modo>]
[UPDATE <actualización>]
[MESSAGES INTO <tabla-mensajes].

<transacción> es el código de la transacción, a la cual vamos a realizar el batch-input. <tabla> es la tabla interna con los datos de batch-input (estructura BDCDATA). Si la transacción devuelve algún tipo de mensaje, las siguientes variables del sistema guardan la siguiente información:

- SY-MSGID -> Identificador de mensaje (REPORT ..... MESSAGE-ID xx).
- SY-MSGTY -> Tipo de mensaje (A -> Abend, E-> Error, ...)
- SY-MSGNO -> Número de mensaje.
- SY-MSGV1 -> Valor 1 del mensaje (si existe).
- SY-MSGV2 -> Valor 2 del mensaje (si existe).
- SY-MSGV3 -> Valor 3 del mensaje (si existe).
- SY-MSGV4 -> Valor 4 del mensaje (si existe).

La variable del sistema SY-SUBRC toma el valor 0 si el proceso ha sido satisfactorio. En caso contrario toma un valor distinto de 0. Una llamada a una transacción puede terminar satisfactoriamente si se realiza un COMMIT WORK, un CALL SCREEN con valor 0 o un LEAVE TO TRANSACTION <código-transacción>.

Con la cláusula MODE especificamos el modo de ejecución de la transacción. <modo> puede tener uno de los siguientes valores (el valor por defecto es A):

A -> Se muestran todas las pantallas.
E -> Se muestran sólo las pantallas con error.
N -> No se muestra ninguna pantalla.

Con la cláusula UPDATE especificamos el modo de actualización de la base de datos. <actualización> puede tener uno de los siguientes valores (el valor por defecto es A):

A -> Modo asíncrono.
S -> Modo síncrono.

Con la cláusula MESSAGES INTO el sistema actualiza la tabla interna <tabla-mensajes> con todos los mensajes que se generan en la transacción. La tabla interna debe tener la estructura BDCMSGCOLL.

Los errores que se pueden producir en tiempo de ejecución son los siguientes:

CALL_TRANSACTION_NOT_FOUND -> Transacción desconocida.
CALL_TRANSACTION_IS_MENU -> La transacción especificada es un menú.
CALL_TRANSACTION_USING_NESTED -> Se ha utilizado una llamada recursiva inválida para esta sentencia.
CALL_TRANSACTION_LOCKED -> Transacción bloqueda.

CASE .. ENDCASE

Definición

Para ejecutar diferentes bloques de sentencias en función del contenido de una variable, se utiliza la sentencia CASE.

Sintaxis:

```
CASE <c>.
  WHEN <c1> [ OR <c11> ].
    [ <bloque-sentencias> ]
  [ WHEN <c2> [ OR <C21> ]. ]
    [ <bloque-sentencias> ]
  [ WHEN OTHERS. ]
    [ <bloque-sentencias> ]
ENDCASE.
```

- El sistema ejecuta el bloque de sentencias correspodiente a la cláusula WHEN que cumpla que el valor de <c> coincida con el valor de <cn>, continuando el proceso a continuación de la cláusula ENDCASE.
- Con la cláusula OR, se puede indicar más de un campo para realizar la comparación. El bloque de sentencias correspondiente a la cláusula WHEN OTHERS se procesará si ninguna de las cláusulas WHEN se cumple.
- La cláusula ENDCASE es obligatoria.

Ejemplo:

```
DATA: ONE   TYPE I VALUE 1,
      THREE TYPE P VALUE 3.
DO 5 TIMES.
  CASE SY-INDEX.
    WHEN ONE.
      WRITE / 'Este es'.
    WHEN 2.
      WRITE   'un'.
    WHEN THREE.
      WRITE 'buen'.
      WRITE 'ejmeplo'.
    WHEN OTHERS.
      WRITE '!'.
  ENDCASE.
ENDDO.
```

Salida: " Este es un buen ejemplo ! ! "
Vease también: IF, ELSEIF.

CHAIN

Definición
La sentencia CHAIN permite agrupar campos de pantalla.
Sintaxis:
CHAIN.
...
ENDCHAIN.
Si ocurre algún error sobre algún campo de pantalla interno a un CHAIN .. ENDCHAIN, todos los campos de la agrupación permiten entrada de datos. Esto sirve para que campos relacionados, si se produce algún error, se puedan modificar conjuntamente.
Véase también: FIELD.

CHECK

Definición
Permite terminar con la ejecución de un bloque de proceso de forma condicional. Para terminar el paso de un bucle de forma condicional se utiliza la sentencia CHECK.
Sintaxis:
CHECK <condición>.
CHECK {<criterio-seleccion> | SELECT-OPTIONS}
Con la primera variante de la sentencia CHECK utilizamos una condición. Si <condicion> resulta FALSE, el sistema abandona el actual bloque de proceso para continuar con la próxima ocurrencia del mismo evento (para el evento GET) o el siguiente evento. <condición> puede ser cualquier expresión lógica.
La siguiente variante de la sentencia CHECK se utiliza conjuntamente con el evento GET. <criterio-selección> puede ser un criterio de selección con la sentencia SELECT-OPTIONS o con la sentencia RANGES. En la definición se asocia un criterio de selección a un campo de una tabla de la base de datos. En el evento GET con la tabla de la base de datos relacionada se puede utilizar esta variante. El campo debe cumplir las condiciones del criterio de selección.

- Con la cláusula SELECT-OPTIONS el sistema comprueba todos los criterios de selección definidos para campos de la tabla referenciada en el evento GET.

CHECK <condición>.
Si la condición resulta FALSE, el sistema deja de procesar el resto de sentencias del bloque para pasar al siguiente paso del bucle. <condición> es cualquier expresión lógica.
CHECK { <criterio> | SELECT-OPTIONS }.
<criterio> es un criterio de selección definido con la sentencia SELECT-OPTIONS. La sentencia CHECK comprueba si el dato leído con la sentencia GET cumple las condiciones del criterio de selección.

- Con la cláusula SELECT-OPTIONS la sentencia CHECK comprueba todos los criterios de selección definidos para la tabla leída con CHECK.

La sentencia CHECK termina una subrutina de forma condicional.
CHECK <condición>.
Si la condición es FALSE, el sistema abandona la subrutina. El programa continúa la ejecución en el punto donde se realizó la llamada.
Vease también: CONTINUE, EXIT, REJECT, STOP.

CLEAR

Definición

Con la sentencia CLEAR se puede inicializar, o la línea de cabecera de la tabla, o el contenido de la tabla.

Sintaxis:

CLEAR <campo>.

Esta sentencia inicia el contenido del objeto de datos <campo> a sus valores iniciales por defecto. Se puede distinguir entre los siguientes casos:

Tipos de datos elementales. El sistema inicia el valor del campo <campo> a su valor por defecto, no el valor inicial asignado con la sentencia DATA, con la cláusula VALUE. las constantes no se pueden iniciar.

Field-string. Aplicando la sentencia CLEAR sobre un field-string, el contenido de cada componente se inicia a su valor por defecto.

Tablas internas. En una tabla interna se inicia el área de trabajo de la tabla interna, y no su contenido.

CLEAR <tabla> | <tabla> [].

Con la <tabla> inicializamos la línea de cabecera de la tabla. Con <tabla> [ ] vaciamos el contenido de la tabla interna.

Ejemplo:

```
DATA: TEXT(10)       VALUE 'Hello',
      NUMBER TYPE I  VALUE 12345,
      ROW(10) TYPE N VALUE '1234567890',
      BEGIN OF PLAYER,
        NAME(10)      VALUE 'John',
        TEL(8) TYPE N VALUE '08154711',
        MONEY  TYPE P VALUE 30000,
      END   OF PLAYER.
...
CLEAR: TEXT, NUMBER, PLAYER.
```

El contenido de los campos serían los siguientes:

```
ROW = '1234567890'
TEXT = ' '
NUMBER = 0
PLAYER-NAME = ' '
PLAYER-TEL = '00000000'
PLAYER-MONEY = 0
```

Vease también: REFRESH.

CLOSE CURSOR

Definición

Sentencia utilizada para cerrar un cursor de una tabla de la base de datos.

Sintaxis:

CLOSE CURSOR <cursor>.

Debemos cerrar aquellos cursores que no vayan a ser utilizados más en un programa. El cursor se cierra automáticamente en los siguientes casos:

- Cuando se ejecuta una de las siguientes sentencias COMMIT WORK o ROLLBACK WORK.
- Cuando una sentencia nativa (native SQL) realiza una de las siguientes funciones commit o rollback (con la opción WITH HOLD no sucede esto).
- Cuando se produce un cambio de pantalla (reports interactivos o transacciones).
- Cuando se realiza una RFC o "Remote Function Call".

Vease también: OPEN CURSOR.

CLOSE DATASET

Definición
Esta sentencia cierra un fichero del servidor de aplicación.
Sintaxis:
CLOSE DATASET <fichero>.
Cierra el fichero <fichero>. <fichero> puede ser un literal o un campo, de la misma forma que en la sentencia OPEN DATASET. La operación de cerrar un fichero es sólo necesaria si quieres borrar el contenido de éste durante otra operación de escritura.
Para evitar errores y hacer que los programas sean más fáciles de leer, se debe cerrar un fichero antes de volver a utilizar la sentencia OPEN DATASET. Utilizando la sentencia CLOSE el programa se divide en bloques lógicos, haciendo más fácil el mantenimiento.
Vease también: OPEN DATASET.

CNT

Definición
La sentencia CNT no es realmente una sentencia, es una campo que el sistema crea y rellena en los tratamientos de extractos de datos.
Sintaxis:
CNT (<campo>).
Esta sentencia sólo puede ser utilizada dentro de un bucle LOOP .. ENDLOOP para el procesamiento de un extracto de datos.
Si la condición es FALSE, el sistema abandona la subrutina. El programa continúa la ejecución en el punto donde se realizó la llamada.
Vease también: SUM.

COLLECT

Definición
Para rellenar líneas en una tabla interna con la comprobación por parte del sistema de que la clave estándar de la tabla será única.
Sintaxis:
COLLECT [<área-trabajo> INTO]<tabla> [SORTED BY <campo>].

- Para especificar un área de trabajo distinta al área de trabajo de la tabla interna se utiliza la cláusula <área-trabajo> INTO.
- El sistema comprueba si existe alguna línea en la tabla interna con la misma clave estándar. Recordemos que la clave estándar esta compuesta de todos los campos no numéricos. De no existir una línea con la misma clave, el efecto de esta sentencia es el mismo que el de la sentencia APPEND, añade una entrada en la tabla. Si existe ya una línea con la misma clave estándar, no se inserta ninguna línea a la tabla interna; si hay campos numéricos, se sumarán.
- La cláusula SORTED BY <campo> no debe utilizarse. En futuras versiones esta cláusula desaparecerá. En su lugar se puede utilizar la sentencia APPEND.

Ejemplo:

```
DATA: BEGIN OF COMPANIES OCCURS 10,
      NAME(20),
      SALES TYPE I,
    END   OF COMPANIES.
COMPANIES-NAME = 'Duck'.  COMPANIES-SALES = 10.
COLLECT COMPANIES.
```

COMPANIES-NAME = 'Tiger'. COMPANIES-SALES = 20.
COLLECT COMPANIES.
COMPANIES-NAME = 'Duck'. COMPANIES-SALES = 30.
COLLECT COMPANIES.
El resultado en la tabla *companies* sería el siguiente:

NAMESALES
Duck 40
Tiger 20

Vease también: APPEND, MODIFY, INSERT.

COMMIT WORK

Definición
A veces es necesario asegurarse que los cambios en la base de datos se han realizado, antes de continuar con el proceso. Por el contrario, a veces es necesario deshacer algunos cambios realizados en la base de datos. Para confirmar los cambios realizados sobre la base de datos se utiliza la sentencia COMMIT WORK
Sintaxis:
COMMIT WORK [AND WAIT].
para deshacer los cambios realizados en la base de datos se utiliza la sentencia ROLLBACK WORK. Estas sentencias tienen un papel importante en la programación de transacciones de diálogo.
Con la cláusula AND WAIT, el programa se para hasta que la tarea de actualización termina. Si la actualización es satisfactoria, SY-SUBRC vale 0, en caso contrario, SY-SUBRC toma un valor distinto de 0.
Vease también: ROLLBACK WORK.

COMMUNICATION

Definición
Permite realizar comunicación directa "programa-a-programa" (CPI-C) definida por IBM en el contexto de los estándares SAA.
Sintaxis:
COMMUNICATION INIT DESTINATION <destino> ID <id>
[RETURNCODE <rc>].
Inicia la conexión "programa-a-programa". En el campo <rc> se actualiza con el código de retorno de la sentencia.
COMMUNICATION ALLOCATE ID <id>
[RETURNCODE <rc>].
Establece la conexión con el otro programa. Esta sentencia debe estar a continuación de la variante anterior. En el campos <rc> se actualiza con el código de retorno de la sentencia.
COMMUNICATION ACCEPT ID <id>
[RETURNCODE <rc>].
Acepta la conexión requerida por el programa que controla la comunicación. En el campo <rc> se actualiza con el código de retorno de la sentencia.
COMMUNICATION SEND ID <id> BUFFER <f>
[RETURNCODE <rc>]
[LENGTH <longitud>].
Envío de datos al programa principal. Los datos enviados se encuentran en el campo <f>. En el campo <rc> se actualiza con el código de retorno de la sentencia. La cláusula LENGTH se utiliza para especificar la longitud del campo <f>.
COMMUNICATION RECEIVE ID <id> BUFFER <f> DATAINFO <d> STATUSINFO <s>

[RETURNCODE <rc>]
[LENGTH <longitud>]
[RECEIVED <m>]
[HOLD].
Recibe datos de otro programa sobre el campo <f>. En el campo <rc> se actualiza con el código de retorno de la sentencia. La cláusula LENGTH se utiliza para especificar la longitud del campo <f>. Después de la llamada el campo <m> contiene el número de bytes recibidos. Con la cláusula HOLD el proceso espera la recepción completa de los datos.
COMMUNICATION DEALLOCATE ID <id>
[RETURNCODE <rc>].
Cierra la conexión "programa-a-programa" y libera todos los recursos utilizados. En el campo <rc> se actualiza con el código de retorno de la sentencia.

Ejemplo 1:

```
TYPES: CONVERSATION_ID(8) TYPE C,
       DESTINATION(8)     TYPE C,
       RETURN_CODE        LIKE SY-SUBRC.
DATA:  CONVID  TYPE CONVERSATION_ID,
       DEST    TYPE DESTINATION VALUE 'C00',
       CPIC_RC TYPE RETURN_CODE.
INCLUDE RSCPICDF.

COMMUNICATION INIT DESTINATION DEST
                 ID         CONVID
                 RETURNCODE  CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: /'COMMUNICATION INIT, RC = ', CPIC_RC.
  EXIT.
ENDIF.
```

Ejemplo 2:

```
TYPES: CONVERSATION_ID(8) TYPE C,
       DESTINATION(8)     TYPE C,
       RETURN_CODE        LIKE SY-SUBRC.
DATA:  CONVID  TYPE CONVERSATION_ID,
       DEST    TYPE DESTINATION VALUE 'C00',
       CPIC_RC TYPE RETURN_CODE.
INCLUDE RSCPICDF.

COMMUNICATION INIT DESTINATION DEST
                 ID         CONVID
                 RETURNCODE  CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: /'COMMUNICATION INIT, RC = ', CPIC_RC.
  EXIT.
ENDIF.
COMMUNICATION ALLOCATE ID CONVID RETURNCODE CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: /'COMMUNICATION ALLOCATE, RC = ', CPIC_RC.
  EXIT.
ENDIF.
```

Ejemplo 3:

```
FORM CPIC_EXAMPLE.
  TYPES: CONVERSATION_ID(8) TYPE C,
         RETURN_CODE        LIKE SY-SUBRC.
  DATA:  CONVID  TYPE CONVERSATION_ID,
```

```
      CPIC_RC TYPE RETURN_CODE.
  INCLUDE RSCPICDF.
  COMMUNICATION ACCEPT ID CONVID
                  RETURNCODE  CPIC_RC.
  IF CPIC_RC NE CM_OK.
    EXIT.
  ENDIF.
ENDFORM.
```

Ejemplo 4:

```
TYPES: CONVERSATION_ID(8) TYPE C,
      DESTINATION(8)     TYPE C,
      RETURN_CODE        LIKE SY-SUBRC.
DATA:  CONVID  TYPE CONVERSATION_ID,
      DEST    TYPE DESTINATION VALUE 'C00',
      CPIC_RC TYPE RETURN_CODE.
INCLUDE RSCPICDF.

COMMUNICATION INIT DESTINATION DEST
                ID          CONVID
                RETURNCODE  CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: /'COMMUNICATION INIT, RC = ', CPIC_RC.
  EXIT.
ENDIF.
COMMUNICATION ALLOCATE ID CONVID                    RETURNCODE
CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: /'COMMUNICATION ALLOCATE, RC = ', CPIC_RC.
  EXIT.
ENDIF.
RECORD = 'Prueba de mensaje'.
COMMUNICATION SEND ID     CONVID
                BUFFER RECORD
                LENGTH LENG
                RETURNCODE CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: / 'COMMUNICATION SEND, RC = ', CPIC_RC.
  EXIT.
ENDIF.
```

Ejemplo 5:

```
FORM CPIC_EXAMPLE.
  TYPES: CONVERSATION_ID(8) TYPE C,
        RETURN_CODE       LIKE SY-SUBRC,
        C_INFO(4)         TYPE X.
  DATA:  CONVID  TYPE CONVERSATION_ID,
        CPIC_RC TYPE RETURN_CODE,
        RECORD(80) TYPE C,
        DINFO      TYPE C_INFO,
        SINFO      TYPE C_INFO.
  INCLUDE RSCPICDF.
  COMMUNICATION ACCEPT ID CONVID
                  RETURNCODE  CPIC_RC.
  IF CPIC_RC NE CM_OK.
    EXIT.
```

```
  ENDIF.
  COMMUNICATION RECEIVE ID         CONVID
                    BUFFER     RECORD
                    STATUSINFO SINFO
                    DATAINFO   DINFO
                    RETURNCODE CPIC_RC.
  IF CPIC_RC NE CM_OK.
    EXIT.
  ENDIF.
ENDFORM.
```

Ejemplo 6:

```
TYPES: CONVERSATION_ID(8) TYPE C,
       DESTINATION(8)     TYPE C,
       RETURN_CODE        LIKE SY-SUBRC,
       C_INFO(4)          TYPE X.
DATA:  CONVID  TYPE CONVERSATION_ID,
       CPIC_RC TYPE RETURN_CODE,
       DEST    TYPE DESTINATION VALUE 'C00'.
DATA:  RECORD(80) TYPE C,
       LENG       TYPE I VALUE 20.
INCLUDE RSCPICDF.
COMMUNICATION INIT DESTINATION DEST
                 ID         CONVID
                 RETURNCODE  CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: / 'COMMUNICATION INIT, RC = ', CPIC_RC.
  EXIT.
ENDIF.
COMMUNICATION ALLOCATE ID CONVID
                   RETURNCODE CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: / 'COMMUNICATION ALLOCATE, RC = ', CPIC_RC.
  EXIT.
ENDIF.
RECORD = 'Prueba de mensaje'.
COMMUNICATION SEND ID     CONVID
                 BUFFER RECORD
                 LENGTH LENG
                 RETURNCODE CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: / 'COMMUNICATION SEND, RC = ', CPIC_RC.
  EXIT.
ENDIF.
COMMUNICATION DEALLOCATE ID CONVID
                    RETURNCODE CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: / 'COMMUNICATION DEALLOCATE, RC = ', CPIC_RC.
  EXIT.
ENDIF.
```

Ejemplo 7:

```
PROGRAM ZCPICTST.
TYPES: CONVERSATION_ID(8) TYPE C,
       DESTINATION(8)     TYPE C,
       RETURN_CODE        LIKE SY-SUBRC,
       C_INFO(4)          TYPE X.
DATA:  BEGIN OF CONNECT_STRING,
```

```
        REQID(4) VALUE 'CONN',
        TYPE(4)  VALUE 'CPIC',
        MODE(4)  VALUE '1   ',
        MANDT(3) VALUE '000',
        NAME(12) VALUE 'CPICUSER',
        PASSW(8) VALUE 'CPIC',
        LANGU(1) VALUE 'D',
        KORRV(1),
        REPORT(8) VALUE 'ZCPICTST',
        FORM(30)  VALUE 'CPIC_EXAMPLE',
      END OF CONNECT_STRING.
DATA:  CONVID   TYPE CONVERSATION_ID,
      DEST     TYPE DESTINATION VALUE 'R2-SYST',
      CPIC_RC  TYPE RETURN_CODE,
      DINFO    TYPE C_INFO,
      SINFO    TYPE C_INFO.
DATA:  RECORD(80) TYPE C,
      LENG       TYPE I VALUE 20.
INCLUDE RSCPICDF.
COMMUNICATION INIT DESTINATION DEST
                ID          CONVID
                RETURNCODE  CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: / 'COMMUNICATION INIT, RC = ', CPIC_RC.
  EXIT.
ENDIF.
COMMUNICATION ALLOCATE ID CONVID
                  RETURNCODE CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: / 'COMMUNICATION ALLOCATE, RC = ', CPIC_RC.
  EXIT.
ENDIF.
* Convert logon data to EBCDIC
TRANSLATE CONNECT_STRING TO CODE PAGE '0100'.
COMMUNICATION SEND ID CONVID BUFFER CONNECT_STRING.
IF CPIC_RC NE CM_OK.
  WRITE: / 'COMMUNICATION ALLOCATE, RC = ', CPIC_RC.
  EXIT.
ENDIF.
* Receive acknowledgement of logon
COMMUNICATION RECEIVE ID      CONVID
                BUFFER     RECORD
                DATAINFO   DINFO
                STATUSINFO SINFO
                RETURNCODE CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: / 'COMMUNICATION RECEIVE, RC = ', CPIC_RC.
  EXIT.
ENDIF.
* Convert acknowledgement to ASCII
TRANSLATE RECORD FROM CODE PAGE '0100'.
* Now begin user-specific data exchange
RECORD = 'The quick brown fox jumps over the lazy dog'.
* Depending on the partner system, convert to another
* character set
TRANSLATE RECORD TO CODE PAGE '0100'.
```

```
COMMUNICATION SEND ID     CONVID
                BUFFER RECORD
                LENGTH LENG
                RETURNCODE CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: / 'COMMUNICATION SEND, RC = ', CPIC_RC.
  EXIT.
ENDIF.
COMMUNICATION DEALLOCATE ID CONVID
                  RETURNCODE CPIC_RC.
IF CPIC_RC NE CM_OK.
  WRITE: / 'COMMUNICATION DEALLOCATE, RC = ', CPIC_RC.
  EXIT.
ENDIF.
PROGRAM ZCPICTST.
INCLUDE RSCPICDF.
* The receiving procedure in the relevant partner program follows
FORM CPIC_EXAMPLE.
 TYPES: CONVERSATION_ID(8) TYPE C,
       RETURN_CODE       LIKE SY-SUBRC,
       C_INFO(4)         TYPE X.
 DATA:  CONVID  TYPE CONVERSATION_ID,
       CPIC_RC TYPE RETURN_CODE,
       RECORD(80) TYPE C,
       DINFO      TYPE C_INFO,
       SINFO      TYPE C_INFO.
 COMMUNICATION ACCEPT ID CONVID
                 RETURNCODE  CPIC_RC.
 IF CPIC_RC NE CM_OK.
   EXIT.
 ENDIF.
 COMMUNICATION RECEIVE ID        CONVID
                 BUFFER     RECORD
                 STATUSINFO SINFO
                 DATAINFO   DINFO
                 RETURNCODE CPIC_RC.
 IF CPIC_RC NE CM_OK AND CPIC_RC NE CM_DEALLOCATED_NORMAL.
   EXIT.
 ENDIF.
ENDFORM.
```

COMPUTE

Definición

Para procesar objetos de datos numéricos al asignar el valor resultante a un objeto de datos, se puede utilizar la sentencia compute.

Sintaxis:

[COMPUTE]<campo> = <expresión>.

La palabra clave COMPUTE es opcional (única sentencia que su palabra clave es opcional). El resultado de la operación matemática especificada en <expresión> se asigna al campo <campo>. Si el resultado de la operación no tiene el mismo tipo de dato que el campo <campo>, el sistema realiza la conversión oportuna de forma automática.

Los operándos de <expresión> deben ser de tipo numérico.

Los operadores válidos son los siguientes:

Operador	Significado	Ejemplo
+	Suma	resultado = campo1 + campo2.
-	Diferencia	resultado = campo1 - campo2.
*	Multiplicación	resultado = campo1 * campo2.
/	División	resultado = campo1 / campo2.
DIV	División entera	resultado = campo1 DIV campo2.
MOD	Resto de una división entera	resultado = campo1 MOD campo2.
**	Exponenciación	resultado = campo1 ** campo2.

En lugar de utilizar los operadores básicos +, -, * y / se puede utilizar las sentencias ADD, SUBTRACT, MULTIPLY y DIVIDE respectivamente. Estas sentencias ya las veremos en posteriores apartados. Los operadores vistos anteriormente, así como los paréntesis, son palabras clave del lenguaje ABAP/4, por ellos, deben ir con al menos un espacio en blanco por delante y por detrás. En la división, el divisor no puede ser cero. Si combinamos varias expresiones, los operadores de igualdad y prioridad se evalúan de derecha a izquierda. Excepto en el caso del operador de exponenciación, que se evalúa de derecha a izquierda.
La prioridad a la hora de evaluar una expresión es la siguiente: paréntesis, funciones, exponenciación (**), operadores *, /, MOD y DIV y operadores "+" y "-".
Además de los operadores vistos hasta ahora se puede utilizar un conjunto de funciones predefinidas. El formato que hay que utilizar es el siguiente:
[COMPUTE] <campo> = <función> (<argumento>).

- Los espacios en blanco entre los paréntesis y los argumentos son obligatorios. El resultado de llamar a la función <función> con el argumento <argumento> es asignado a <campo>. Las funciones existentes pueden clasificarse en los siguientes grupos:
- Funciones válidas para todos los tipos numéricos (tipo F, I y P). El argumento no tiene por qué ser numérico. Si utilizamos otro tipo, éste es convertido a un tipo numérico. Por razón de optimización es recomendable utilizar tipo de datos numéricos.
- Funciones sólo para el tipo F. Para estas funciones existen las restricciones normales lógicas de las funciones matemáticas, por ejemplo, la raíz cuadrada sólo es válida con números positivos. El argumento no tiene por qué ser del tipo F. De no ser de este tipo el sistema realizará la conversión necesaria de forma automática.

Funciones válidas para todos los tipos numéricos (F, I y P):

Operador	Significado	Ejemplo
ABS	Valor absoluto del argumento	resultado = ABS(campo).
SIGN	Signo del argumento. La función de vuelve 1 si el argumento es positivo, 0 si el argumento es 0 y -1 si es negativo.	resultado = SIGN(campo).
CEIL	Valor entero inferior al argumento	resultado = CEIL(campo).
FLOOR	Valor entero superior al argumento	resultado = FLOOR(campo).
TRUNC	Parte entera del argumento	resultado = TRUNC(campo1).
FRAC	Parte fraccionaria del argumento	resultado = FRAC(campo1).

Funciones sólo para el tipo F:

Operador	Significado	Ejemplo
ACOS	Arcocoseno	resultado = ACOS(campo).
ASIN	Arcoseno	resultado = ASIN(campo).
ATAN	Arcotangente	resultado = ATAN(campo).
COS	Coseno	resultado = COS(campo).
SIN	Seno	resultado = SINcampo1).
TAN	Tangente	resultado = TAN(campo1).
COSH	Coseno hiperbólico	resultado = COSHcampo1).
SINH	Seno hiperbólico	resultado = SINHcampo1).
TANH	Tangente hiperbólico	resultado = TANH(campo1).

EXP	Exponenciación	resultado = EXP(campo1).
LOG	Logaritmo natural (base e)	resultado = LOG(campo1).
LOG10	Logaritmo base 10	resultado = LOG10(campo1).
SQRT	Raíz cuadrada	resultado = SQRT(campo1).

Si el atributo *Aritmética en coma fija* no está marcado, los campos empaquetados (tipo P) son enteros sin punto decimal. El parámetro DECIMALS de la sentencia DATA sólo es efectivo para la sentencia WRITE. Por esta razón SAP recomienda que siempre que trabajemos con campos de tipo P marquemos el atributo *Aritmética en coma fija*. Cuando tenemos marcado este atributo, no sólo tiene efecto con la sentencia WRITE, sino que también toma relevancia en las operaciones numéricas. Los resultados intermedios se guardan con 31 posiciones significativas antes y después del punto decimal.

Los tipos de datos fecha y hora no son tipo numéricos pero se pueden realizar operaciones aritméticas se suele usar el offset de los campos.

Existe una función que opera con los campos alfanuméricos:

Operador **Significado** **Ejemplo**

STRLEN Determina la longitud hasta el último carácter distinto del espacio en blanco. resultado = STRLEN(campo).

Vease también: ADD, SUBTRACT, MULTIPLY y DIVIDE.

CONCATENATE

Definición

Se utiliza para concatenar varios campos alfanuméricos en uso solo.

Sintaxis:

CONCATENATE <c1> ... <cn> INTO <campo> [SEPARATED BY <s>].

Esta sentencia concatena los campos campos <c1> ... <cn> en el campo <campo>. Los espacios en blanco se ignoran durante la operación.

Con la cláusula SEPARATED BY se puede especificar un campo alfanumérico (el campo <s>) que será utilizado como separador entre los campos <c1> ... <cn>. Si el resultado de la concatenación entra en el campo <campo>, SY-SUBRC = 0, si por el contrario, es necesario el truncamiento, SY-SUBRC = 4.

Ejemplo 1:

```
DATA: ONE(10)   VALUE 'Ivan',
      TWO(3)    VALUE ' Rodrigo',
      THREE(10) VALUE ' Baños',
      NAME(20).
CONCATENATE ONE TWO THREE INTO NAME.
```

La variable NAME valdría: *Ivan Rodrigo Baños*

Ejemplo 2:

```
DATA: ONE(10)   VALUE 'Ivan',
      TWO(3)    VALUE 'Rodrigo',
      THREE(10) VALUE 'Baños',
      NAME(20),
      SEPARATOR(4) VALUE 'GAUSS'.
CONCATENATE SPACE ONE TWO THREE INTO NAME
        SEPARATED BY SPACE.
```

La variable NAME valdría: *IvanGAUSSRodrigoGAUS* y SY-SUBRC valdría 4.

Vease también: SPLIT, SHIFT, REPLACE, TRANSLATE, CONDENSE.

CONDENSE

Definición
Para borrar espacios en blanco superfluos en campos alfanuméricos.
Sintaxis:
CONDENSE <campo> [NO-GAPS].
Borra cualquier secuencia de espacios en blanco, dejando sólo uno que exista entre palabras existentes en <campo>. Los espacios en blanco por la izquierda también desaparecen.
Con la cláusula NO-GAPS todos los espacios en blanco desaparecen.
Ejemplo 1:

```
DATA: BEGIN OF NAME,
      TITLE(8),      VALUE 'Dr.',
      FIRST_NAME(10), VALUE 'Michael',
      SURNAME(10),    VALUE 'Hofmann',
    END   OF NAME.
CONDENSE NAME.
WRITE NAME.
```

La salida en pantalla sería: *Dr. Michael Hofmann*
Ejemplo 2:

```
DATA: BEGIN OF NAME,
      TITLE(8),      VALUE 'Dr.',
      FIRST_NAME(10), VALUE 'Michael',
      SURNAME(10),   VALUE 'Hofmann',
    END   OF NAME.
CONDENSE NAME NO-GAPS.
```

La variable NAME valdría: *Dr.MichaelHofmann*
Vease también: SPLIT, SHIFT, REPLACE, TRANSLATE, CONCATENATE.

CONSTANTS

Definición
Declaración de constantes, es decir, variables o registros cuyos valores no pueden ser modificados durante la ejecución del programa.
Sintaxis:
CONSTANTS <constante> [(<longitud>)] [<tipo>] <valor> [<decimales>].
<longitud>, <tipo>, <valor> y <decimales> son las mismas opciones de la sentencia DATA. Debemos darnos cuenta de que en esta sentencia, la opción <valor> es obligatoria. El valor inicial indicado con <valor> no puede ser cambiado durante la ejecución del programa.

```
CONSTANTS: BEGIN OF <registro>,
             ....
           END OF <registro>.
```

La sintaxis de un registro de constantes es igual a la sintaxis de la sentencia DATA, con la diferencia de que la cláusula VALUE es obligatoria.
Ejemplo:

```
CONSTANTS  CHAR1 VALUE 'X'.

CONSTANTS  INT   TYPE I VALUE 99.

CONSTANTS: BEGIN OF CONST_REC,
           C(2) TYPE I VALUE 'XX',
           N(2) TYPE N VALUE '12',
           X    TYPE X VALUE 'FF',
           I    TYPE I VALUE 99,
           P    TYPE P VALUE 99,
           F    TYPE F VALUE '9.99E9',
```

```
  D    TYPE D VALUE '19950101',
  T    TYPE T VALUE '235959',
END OF CONST_REC.
```

CONTINUE

Definición
Se utiliza para terminar el paso de un bucle de forma incondicional.
Sintaxis:
CONTINUE
Después de la sentencia CONTINUE el sistema no procesa ninguna sentencia más del actual paso del bucle, y continua con el siguiente paso:
Ejemplo:

```
DO 100 TIMES.
  IF SY-INDEX >= 10 AND SY-INDEX <= 20.
    CONTINUE.
  ENDIF.
  ...
ENDDO.
```

Vease también: CHECK, EXIT.

CONTROLS

Definición
Declara un *control*, objeto de dato especial que de momento sólo se puede utilizar para la declaración de tablas *gráficas* en las pantallas:
Sintaxis:
CONTROLS <control> TYPE <tipo>.
<control> especifica el nombre del control, y <tipo> su tipo. hasta el momento el único tipo de dato válido es TABLEVIEW, tipo utilizado para presentar tablas en pantalla.
Vease también: TABLE CONTROL, REFRESH CONTROL.

CONVERT DATE

Definición
Se utiliza para convertir campos con formato fecha (tipo D) a un formato de fecha invertida, por ejemplo, ordenar dicho campo de manera descendente.
Sintaxis:
CONVERT DATE <fecha1> INTO INVERTED-DATE <fecha2>.
CONVERT INVERTED-DATE <fecha1> INTO DATE <fecha2>.
La primera sentencia convierte un campo fecha con formato fecha a formato fecha invertida. La segunda sentencia realiza la función contraria. Para la conversión el sistema utiliza el complemento a 9.
Ejemplo:

```
DATA DATE_INV LIKE SY-DATUM.
CONVERT DATE SY-DATUM INTO INVERTED-DATE DATE_INV.
```

Si por ejemplo la representación interna de la fecha 11.05.95 en SY-DATUM es 19950511, eL valor de la variable DATE_INV después de la ejecución de CONVERT sería 80049488.
Vease también: CONVERT TEXT.

CONVERT TEXT

Definición
Se utiliza para convertir un campo alfanumérico en un formato válido para la ordenación.
Sintaxis:
CONVERT TEXT <campo1> INTO SORTABLE CODE <campo2>.
Esta sentencia rellena el campo <campo2> con un código ordenable a partir del campo <campo1>. <campo1> debe ser del tipo C, y <campo> debe ser del tipo X con una tamaño mínimo de 16 veces el tamaño de <campo1>.
Vease también: CONVERT DATE.

CREATE OBJECT

Definición
Genera un objeto de una clase determinada. Sentencia utilizada en los programa OLE.
Sintaxis:
CREATE OBJECT <objeto> <clase>
[LANGUAGE <lenguaje>].
Para crear un <objeto> de una clase <clase> determinada desde ABAP/4, dicho objeto debe ser registrado en el sistema SAP a través de la transacción SOLE.

- Con la cláusula LANGUAGE determinamos el lenguaje del objeto. Si no especificamos esta cláusula se utilizará el idioma inglés.

Vease también: SET PROPERTY, GET PROPERTY, CALL METHOD, FREE OBJECT.

DATA

Definición
Declaración de objetos de datos.
Sintaxis:
DATA <campo> [(<longitud>)] <tipo> [<valor>] [<decimales>]
Variante utilizada para declarar variables. El nombre de la variable <campo> no debe tener más de 30 caracteres. Se puede utilizar cualquier carácter a excepción de: "+", ".", ",", ":", "(" y ")". Además el nombre no puede estar sólo compuesto de dígitos numéricos, ni coincidir con los objetos de datos predefinidos, ni con las palabras reservadas. Veamos algunas recomendaciones:

- Utilizar nombres significativos, que no requieran el uso de comentarios.
- No utilizar el "guión" ya que podríamos confundir la variable con un field-string.
- Utilizar el "guió bajo" para nombres especiales.
- No utilizar los caracteres especiales. Utilizar siempre una letra como primer carácter del nombre.

Con la opción <tipo> se especifica el tipo de variable. Tenemos dos opciones:
TYPE <tipo>
LIKE <objeto-dato>

- Con la cláusula TYPE se puede utilizar cualquier tipo de datos, predefinido o definido por el usuario. Para algunos tipos de datos (C, P, N, y X) se puede definir la longitud con la cláusula <longitud>. Si no se especifica longitud, estos tipos tienen una longitud por defecto. Si no se especifica la cláusula <tipo> el sistema utiliza el tipo alfanumérico C.
- Con la cláusula LIKE asignamos el tipo de dato de forma indirecta, con el tipo de dato del objeto de dato especificado. El objeto de dato debe estar ya declarado, y puede ser de cualquier tipo. A menudo se utiliza esta cláusula para crear variables que almacenarán datos de tablas externas. Si se modifican las características del campos de la tabla externa, de forma automática, se cambian los atributos de la variable.

- Una variante para las cláusulas TYPE y LIKE es la siguiente: TYPE LINE OF <tabla-intera> y LIKE LINE OF <tabla-interna>. Con esta variante creamos un objeto de dato con la estructura de línea de la tabla interna.

Con la opción <calor> se asigna un valor a la variable, distinto al valor inicial. El formato es el siguiente: VALUE <valor>. <valor> puede ser un literal, una constante o IS INITIAL.
Con la opción <decimales> se puede especificar el número de decimales para los campos de tipo P. El formato es el siguiente: DECIMALS <num>, siendo <num> el número de dígitos después del punto decimal. El número máximo de decimales es de 14.

```
DATA: BEGIN OF <registro>,
        ....
      END OF <registro>.
```

Variante utilizada para declarar registros o *field-strings*. Para declarar un *field-strings* se utiliza la sentencia DATA y ser marca el principio y el final de la agrupación con las cláusulas BEGIN OF y END OF. La opción <registro> da nombre al *field-string*.

```
DATA <tabla> <tipo> [ WITH HEADER LINE ].
```

Variante utiliza para declarar tablas internas.

- En la opción <tipo> se puede referenciar un tipo de dato de tabla interna con la cláusula TYPE o una tabla interna con la cláusula LIKE. El objeto de dato <tabla> será creado con la misma estructura que la referenciada en <tipo>.
- Si utilizamos la cláusula WITH HEADER LINE la tabla interna se crea con el área de trabajo <tabla>. El área de trabajo y la tabla interna tienen el mismo nombre. En función de la sentencia que estemos utilizando, el sistema sabe si queremos emplear el área de trabajo o la tabla.

```
DATA <tabla> <tipo> OCCURS <n> [ WITH HEADER LINE ].
```

Esta sentencia crea la tabla interna <tabla> por el uso de la cláusula OCCURS. en la opción <tipo> se especifica una estructura. Las líneas de la tabla interna tendrán la estructura referenciada en <tipo>. <n> tiene el mismo significado que para los tipos de datos de tablas internas, visto en la sentencia TYPES.

- Si utilizamos la cláusula WITH HEADER LINE la tabla se crea con el área de trabajo <tabla>.

```
DATA: BEGIN OF <tabla> OCCURS <n>,
        ....
      END OF <tabla>.
```

- Con esta sentencia definimos la tabla interna <tabla>. Los componentes de la línea de la tabla interna se definen entre las cláusula BEGIN OF y END OF.
- A excepción de la cláusula OCCURS, el resto de la sintaxis es igual a la definición de un field-string, visto anteriormente. Esta sentencia siempre crea un área de trabajo. <n> tiene el mismo significado que el visto anteriormente.

Ejemplo:

```
DATA:  BEGIN OF PERSON_TYPE,
         NAME(20),
         AGE TYPE I,
       END OF PERSON_TYPE,
      PERSONS LIKE PERSON_TYPE OCCURS 20 WITH HEADER LINE.
      PERSONS    TYPE LINE_TYPE OCCURS 20,
      PERSONS_WA TYPE LINE_TYPE.
DATA TAB TYP TYPE I OCCURS 10.
DATA TAB_WA TYPE LINE OF TAB.
DATA NUMBER      TYPE I       VALUE 123,
     FLAG                     VALUE 'X',
     TABLE_INDEX LIKE SY-TABIX VALUE 45.
```

Tipo de datos predefinido:

Tipo	Descripción	Lo	Valor inicial
C	Texto (carácter)	1	Blanco
N	Texto numérico	1	'00...0'

D	Fecha (YYYYMMDD)	8	'00000000'
T	Hora (HHMMSS)	6	'000000'
X	Hexadecimal	1	X'00'
I	Entero	4	0
P	Numero empaquetado	8	0
F	Número punto flotante	8	'0.0'

Vease también: INCLUDE STRUCTURE.

DEFINE .. END-OF-DEFINITION

Definición
Define una macro que contiene parte de código fuente, con la posibilidad de utilizar parámetros.
Sintaxis:
DEFINE <macro>.
... [<&1> ... <&9>]
END-OF-DEFINITION.
<macro> [<p1> ... <p9>]
Para definir una macro se utilizan las palabras claves DEFINE y END-OF-DEFINITION. Entre ellas se incluyen las sentencias que formarán la macro. Se pueden incluir parámetros (<&1> ... <&9>) que serán sustituidos en la llamada. La llamada a la macro se realiza con el nombre de la macro. Durante la generación del programa el sistema reemplaza los parámetros utilizados en la llamada.
Ejemplo:

```
DEFINE ++.
ADD 1 TO &1.
END-OF-DEFINITION.

DATA: NUMBER TYPE I VALUE 1.
...
++ NUMBER.
```

Vease también: FORM, FUNCTION.

DELETE

Definición
Para borrar líneas de una tabla interna contamos con la sentencia DELETE. Varios formatos de la sentencia nos permiten borrar líneas de una forma diferente.
Sintaxis:
DELETE <tabla>.
El sistema sólo puede procesar esta sentencia dentro de un bucle LOOP .. ENDLOOP. La sentencia DELETE borra la línea que estemos tratando en el bucle.
DELETE <tabla> INDEX <índice>.

- Con la cláusula INDEX, el sistema borra la línea con el índice <índice> en la tabla interna <tabla>. Después de borrar la línea, el índice de la siguiente línea es decrementado en 1. Si la operación es completada, la variable SY-SUBRC vale 0. Si la línea con el índice <índice> no existe, SY-SUBRC vale 4.

DELETE ADJACENT DUPLICATE ENTRIES FROM <tabla> [COMPARING <c>].
Esta sentencia borra todas las entradas duplicada adyacentes de una tabla interna. Dos líneas se consideran duplicadas si cumplen uno de los siguientes criterios de comparación:

- Sin la cláusula COMPARING, el contenido de los campos claves estándar deben ser iguales.

- Con la cláusula COMPARING con el formato siguiente: COMPARING <c1> <c2> .... El contenidos de los campos (c1, c2 ....) debe ser igual. También se puede especificar el nombre de los campos que hay que comparar en tiempo de ejecución utilizando la siguiente sintaxis: (<campo>). La variable <campo> contiene el nombre de la variable que se va a comparar. Si en el momento de procesar la sentencia DELETE, <campo> está vacío, el sistema la ignora. Si contiene el nombre inválido, se produce un error en tiempo de ejecución.
- Con la cláusula COMPARING con el siguiente formato: COMPARING ALL FIELDS, el contenido de todos los campos debe ser igual.

Si el sistema borra al menos una línea de la tabla la variable SY-SUBRC valdrá 0 si no borra ninguna línea, SY-SUBRC valdrá 4. Es recomendable que la tabla interna este ordenada antes de ejecutar esta sentencia.

DELETE <TABLA> [FROM <n1>] [TO <n2>] [WHERE <condición>].

Sentencia utilizada para borrar registros de una tabla de la base de datos. Con esta sentencia se puede borrar una o varias líneas de una tabla. Sólo se pueden borrar líneas de una tabla a través de una vista si ésta sólo hace referencia a una única tabla.

Se debe indicar al menos una de las tres cláusulas.

- Si utilizamos la sentencia sin la cláusula WHERE, el sistema borrará todas las líneas de la tabla <tabla> cuyo índice esté comprendido entre <n1> y <n2>.
- Si no utilizamos la cláusula FROM, el sistema borrará desde la primera línea.
- Si no utilizamos la cláusula TO, el sistema borrará hasta la última línea.
- Con la cláusula WHERE el sistema sólo borrará aquellas línea de la tabla <tabla> que cumplen la condición especificada en <condición>. En <condición> se puede utilizar cualquier expresión lógica. El primer operando debe ser un componente de la estrucutra de línea de la tabla interna. Si el sistema borra al menos una línea la variable SY-SUBRC valdrá 0, si no borra ninguna valdrá 4.

DELETE {<tabla> | (<campo>) } [CLIENT SPECIFIED] [FROM <área>].

- Con la cláusula FROM, la línea que se borra es la que coincide con la clave primara del área <área>. Sin la cláusula FROM, la línea que se borra es la que coincide con la clave primaria del área de trabajo de la tabla tabla>
- El área de trabajo <área> debe tener al menos, la longitud de la clave primaria. Si se borra al menos una línea de la tabla la variable SY-SUBRC valdrá 0, si no se borra ninguna valdrá 4.
- Con la especificación dinámica de la tabla, la cláusula FROM es obligatoria.

DELETE FROM {<tabla> | (<campo>) } [CLIENT SPECIFIED] [WHERE <condición>].

- Esta variante nos permite borrar una o varias líneas de la tabla <tabla> en función de las especificaciones de la cláusula WHERE. Dicha cláusula tiene las mismas opciones que las vistas para la sentencia SELECT. Si no especificamos la cláusula WHERE, todas las líneas de la tabla se borran. <tabla> debe estar declarada con la sentencia TABLES.
- La variable del sistema SY-DBCNT contiene el número de líneas borrar. Si se borra al menos una línea de la tabla la variable SY-SUBRC valdrá 0, si no se borra ninguna valdrá 4.

Hay que vigilar mucho con la cláusula WHERE para no borrar toda la tabla o líneas que no deseamos borrar. La única forma de recuperar lo borrada es restaurar el backup de la base de datos.

DELETE {<tabla> | (<tabla>) } [CLIENT SPECIFIED] FROM TABLE <tabla-interna>.

Con esta variante se puede borrar línea de una tabla de diccionario, basándose las líneas de una tabla interna. <tabla> y (<tabla>) sirve para especificar el nombre de la tabla de forma estática o dinámica. Esta sentencia borra aquellas líneas de la tabla cuya clave primaria coincida con la definida en una línea de la tabla interna. La tabla interna debe tener la longitud de la clave primaria de la tabla. Si el sistema no puede borrar ninguna entrada de la tabla porque no coincide ninguna clave primaria, el sistema continua con la siguiente línea de la tabla interna. Si todas las líneas de la tabla interna se procesan, SY-SUBRC vale 0, en caso contrario vale 4. SY-DBCNT nos indica el número de líneas borradas. Si la tabla interna esta vacía, SY-SUBRC y SY-DBCNT valen 0.

Ejemplo 1:

```
DATA: BEGIN OF NAMETAB OCCURS 100,
      NAME(30) TYPE C,      END OF NAMETAB.
...
DELETE NAMETAB FROM 5 TO 36 WHERE NAME CA 'ABC'.
```

Vease también: MODIFY, APPEND, INSERT.

DELETE DATASET

Definición

Para borrar ficheros del servidor de aplicación se utiliza esta sentencia.

Sintaxis:

DELETE DATASET <fichero>.

Esta sentencia borra el fichero <fichero>. <fichero> puede ser un literal o un campo, de la misma forma que en la sentencia OPEN DATASET. Si el sistema borra el fichero SY-SUBRC valdrá 0 en caso contrario valdrá 4.

Vease también: OPEN DATASET.

DELETE DYNPRO

Definición

Se utiliza para borrar un dynpro de la base de datos.

Sintaxis:

DELETE DYNPRO <dynpro>.

Borra el dynpro <dynpro> de la base de datos. La variable SY-SUBRC valdrá 0 si la dynpro se borra correctamente y valdrá 4 en caso contrario.

SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.

DELETE FROM DATABASE

Definición

Permite borrar clusters de un fichero cluster de la base de datos.

Sintaxis:

DELETE FROM DATABASE <tabla>(<área>) [CLIENT <mandante>] ID <clave>.

Borra el cluster con clave <clave> , en el fichero <tabla> y área <área>. <tabla> debe estar declarada con la sentencia TABLES.

- Si el sistema puede borrar el cluster, SY-SUBRC vale 0, en caso contrario, SY-SUBRC vale 4.

Ejemplo:

```
TABLES INDX.
DATA: BEGIN OF TAB OCCURS 1,
      CONT(30),
    END   OF TAB.
DATA: FLD(30) TYPE C.
...
EXPORT TAB FLD TO DATABASE INDX(AR) ID 'TEST'.
```

Véase también: EXPORT TO DATABASE.

DELETE FROM SHARED BUFFER

Definición
Borra un cluster de datos del buffer *cross-transaction application*. (Esta sentencia no debe ser utilizada ya que SAP la determina como de uso interno. SAP puede realizar modificaciones de su sintaxis sin previo aviso).
Sintaxis:
DELETE FROM SHARED BUFFER <tabla>(<id>) [CLIENT <mandante>] ID <clave>.
Borra el cluster de datos creado con la clave <clave> para la tabla <tabla> e identificación <id>. Los cluster de datos pueden ser creados normalmente en memoria o en ficheros especiales de la base de datos. La opción vista en esta sentencia es especial ya que el cluster se almacena en un área determinada de la memoria del servidor de aplicación.

- Con la cláusula CLIENT especificamos el mandante donde se borrará el cluster de datos.

Véase también: EXPORT TO SHARED BUFFER.

DELETE REPORT

Definición
Borra ciertos objetos parciales de un programa.
Sintaxis:
DELETE REPORT <programa>.
Borra el código fuente del programa <programa>, los elementos de texto y la versión generada. La sentencia no borra ni las variantes ni la documentación. SY-SUBRC vale 0 si el programa se borra. En caso contrario vale 4. El sistema proporciona la función RS_DELETE_PROGRAM para realizar la misma función.
SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Véase también: INSERT REPORT.

DELETE TEXTPOOL

Definición
Sentencia utilizada para borrar elementos de texto de la base de datos.
Sintaxis:
DELETE TEXTPOOL <programa> LANGUAGE <lenguaje>.
Borra todos los elementos de texto asociados al programa <programa> en el lenguaje <lenguaje>. Si en el campo <lenguaje> utilizamos un asterisco (*) se borran todos los elementos de texto en todos los lenguajes.
SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Véase también: INSERT TEXTPOOL.

DESCRIBE DISTANCE

Definición
Sentencia utilizada para borrar elementos de texto de la base de datos.
Sintaxis:
DESCRIBE DISTANCE BETWEEN <campo1> AND <campo2> INTO <campo3>.
Determina la distancia entre los campos <campo1> y <campo2> dejando el resultado (en bytes) en el campo <campo3>.

DESCRIBE FIELD

Definición
Recupera los atributos de una variable.
Sintaxis:
DESCRIBE FIELD <variable>
[LENGTH <longitud>]
[TYPE <tipo> [COMPONENTS <n>]]
[OUTPUT-LENGTH <salida>]
[DECIMALS <decimales>]
[EDIT MASK <mascara>].

- La opción LENGTH nos actualiza la variable <longitud> con la longitud del campos.
- La opción TYPE nos actualiza la variable <tipo> con el tipo del campo, que podrá ser uno de los siguientes valores: "C", "D", "F", "I", "N", "P", "T" y "X" para los tipos predefinidos. "s" para enteros de dos bytes con signo, "b" para enteros de un byte sin signo, "h" para tablas internas, y "C" para estructuras. Con la opción COMPONENTS, la sentencia devuelve en el campo <tipo>: "u" para estructuras sin una tabla como componente, y "v" para estructuras con al menos una tabla interna como componente o subcomponente; y sobre el campo <n> el número de componentes directos.
- La opción OUTPUT-LENGTH actualiza la variable <salida> con la longitud de salida de la variable especificada.
- La opción DECIMALS actualiza la variable <decimales> con el número de decimales de la variable especificada.
- Para determinar si existe una rutina de conversión para un campo en el diccionario de datos y además cuál es el nombre de esa rutina se utiliza la opción EDIT MASK. Si existe rutina de conversión, el campo del sistema SY-SUBRC contendrá el valor 0 y la variable <máscara> contendrá el nombre de la rutina de conversión. Si el campo no tiene rutina de conversión el campo del sistema SY-SUBRC tendrá el valor 4.

Ejemplo 1:
```
DATA: FLD(8),
    LEN TYPE P.
DESCRIBE FIELD FLD LENGTH LEN.
```
La variable LEN vale 8.
Ejemplo 2:
```
DATA: FLD(8) TYPE N,
    F_TYPE.
DESCRIBE FIELD FLD TYPE F_TYPE.
```
F_TYPE vale 'N'.

DESCRIBE LIST

Definición
Recupera cierta información sobre las líneas o páginas de un listado.
Sintaxis:
DESCRIBE LIST NUMER OF { LINES | PAGE } <n> [INDEX <índice>].
Esta variante permite recuperar el número de líneas o páginas de un listado.

- Con la cláusula LINES se recupera el número de líneas del listado sobre <n>.
- Con la cláusula PAGES recuperamos el número de páginas sobre <n>.
- Si utuilizamos la cláusula INDEX se puede determinar el listado que queremos analizar, listado básico (VALOR 0) o algún listado secundario (valor 1, 2, ...).

DESCRIBE LIST LINE <línea> PAGE <página> [INDEX <índice>].
Con esta variante obtenemos el número de página para un cierto número de línea.

- La cláusula INDEX tiene el mismo significado que el visto en la variante 1. <línea> determina la línea y <página> determina la página.

DESCRIBE LIST PAGE <página> [INDEX <índice>]
[LINE-SIZE <columnas>] [LINE-COUNT <filas>] [LINES <líneas>]
[FIRST-LINE <lin1>] [TOP-LINES <top>] [TITLE-LINES <título>]
[HEAD-LINES <cabecera>] [END-LINES <pié>].

Esta variante permite recuperar ciertos atributos de una página. La variable del sistema SY-SUBRC toma el valor 0 si la página <página> para el listado <índice> existe. Si el listado determinado por <índice> no existe, toma el valor 8, si el listado existe pero la página no, toma el valor 4. las distintas cláusulas de la sentencia tienen el siguiente significado:

- Cláusula LINE-SIZE. El sistema actualiza la variable <columnas> con el número de columnas de la página <página>.
- Cláusula LINE-COUNT. El sistema actualiza la variable <filas> con el número de filas de la página <página>.
- Cláusula LINES. El sistema actualiza la variable <líneas> con el número de líneas mostradas de la página <página>.
- Clásula FIRST-LINE. El sistema actualiza la variable <lin1> con el valor absoluto de la primera línea de la página <página>.
- Cláusula TOP-LINES. El sistema actualiza la variable <top> con el número de líneas de cabecera de la página <página>.
- Cláusula TITLE-LINES. El sistema actualiza la variable <título> con el número de líneas de la cabecera de página estándar de la página <página>.
- Cláusula HEAD-LINES. El sistema actualiza la variable <cabecera> con el número de líneas de la cabecera de columna estándar de la página <página>.
- Cláusula END-LINES. El sistema actualiza la variable <pie> con el número de líneas de pié de página de la página <página>.

DESCRIBE TABLE

Definición

Se utiliza para saber el número de líneas de una tabla interna y para saber el valor de la cláusula OCCURS de la definición de la tabla.

Sintaxis:

DESCRIBE TABLE <tabla> [LINES <línea>] [OCCURS <tamaño>].

- Si utilizamos la cláusula LINES el sistema actualiza <línea> con el número de entradas en la tabla interna <tabla>.
- Con la cláusula OCCURS el sistema actualiza <tamaño> con el tamaño definido de la tabla.

DETAIL

Definición

Se utiliza para que las líneas de salida se realicen en intensidad normal.

Sintaxis:

DETAIL

Después de esta sentencia, todas las sentencias de escritura se realizan en intensidad normal.

Véase también: FORMAT.

DIVIDE

Definición
Operación matemática para dividir dos campos.
Sintaxis:
DIVIDE <m> BY <n>.
Divide el contenido del campo <m> por <n>, dejando el resultado en <m>.
Véase también: COMPUTE, DIVIDE-CORRESPONDING.

DIVIDE-CORRESPONDING

Definición
Variante de DIVIDE que sólo dividirá aquellos componentes que se llamen igual.
Sintaxis:
DIVIDE-CORRESPONDING <m> BY <n>.
Divide el contenido de los componentes del registro <m> por los del registro <n>, para aquellos que se llamen igual. El resultado permanece en los componentes del registro <m>.
Ejemplo:

```
DATA: BEGIN OF MONEY,
       VALUE_IN(20) VALUE 'Marcos alemanes'.
       USA TYPE I VALUE 100,
       FRG TYPE I VALUE 200,
       AUT TYPE I VALUE 300,
      END   OF MONEY,
      BEGIN OF CHANGE,
       DESCRIPTION(30)
          VALUE 'DM en moneda nacional'.
       USA TYPE F VALUE '1.5',
       FRG TYPE F VALUE '1.0',
       AUT TYPE F VALUE '0.14286',
      END   OF CHANGE.
DIVIDE-CORRESPONDING MONEY BY CHANGE.
MONEY-VALUE_IN = 'Moneda nacional'.
```

Véase también: COMPUTE, DIVIDE.

DO .. ENDDO

Definición
Ejecución de sentencias indefinidamente hasta que se procese la sentencia EXIT, STOP o REJECT.
Sintaxis:

```
DO [<n> TIMES ] [ VARYING <c> FROM <c1> NEXT <c2> ]
  <bloque-de-sentencias>
ENDDO.
```

La sentencia DO sin cláusulas ejecuta el bloque de sentencias indefinidamente, o hasta que se procese una sentencia EXIT, STOP o REJECT. Para limitar el número de pasos de un bucle se puede utilizar la opción TIMES. <n> puede ser un literal o una variable. Si <n> es 0 o negativo, el sistema no procesará el bucle.

- La cláusula ENDDO es obligatoria (marca el fin del bloque de sentencias). La variable SY-INDEX contiene el número de veces que el bucle ha sido ejecutado.

- Utilizando la opción VARYING se pueden ir asignando valores a una variable <c> a partir de un conjunto de campos del mismo tipo y longitud de memoria por cada paso de bucle. En una sentencia DO se pueden utilizar varias opciones VARYING.

Ejemplo:

```
DO.
  WRITE: / 'SY-INDEX - Inicio:', (3) SY-INDEX.
  IF SY-INDEX = 10.
    EXIT.
  ENDIF.
  WRITE: 'Fin:', (3) SY-INDEX.
ENDDO.
```

Véase también: WHILE, STOP, EXIT, REJECT.

EDITOR-CALL FOR REPORT

Definición

Abre el editor de programas con el programa especificado.

Sintaxis:

EDITOR-CALL FOR REPORT <programa> [DISPLAY-MODE]

El programa <programa> se muestra en el editor ABAP/4.

- Con la opción DISPLAY-MODE el programa se muestra en modo visualización, se puede pasar al modo de actualización, a través de los botones del editor, siempre y cuando tengamos autorización.

END-OF-PAGE

Definición

Este evento define un bloque de proceso que se activa cuando el sistema detecta que hemos escrito en la última línea de la página actual.

Sintaxis:

END-OF-PAGE.

El número de líneas por página se define en la sentencia REPORT. Este evento se utiliza para componer pies de página.

Véase también: TOP-OF-PAGE.

END-OF-SELECTION

Definición

Este evento define un bloque de proceso que se ejecuta después de que se hayan procesado los eventos GET, es decir, después de haber sido leídas todas las tablas especificadas de la base de datos lógica asociada al programa.

Sintaxis:

END-OF-SELECTION.

Este evento puede ser utilizado, por ejemplo, para escribir la información que hemos leído de las tablas de diccionario y hemos grabado en tablas internas.

Véase también: START-OF-SELECTION.

EXEC SQL .. ENDEXEC

Definición
Una sentencia nativa debe incluirse en un bloque EXEC SQL .. ENDEXEC.
Sintaxis:
EXEC SQL [PERFORMING <rutina>].
<sentencia-nativa-SQL> [;]
ENDEXEC.
El *punto y coma* (;) es opcional. El *punto* (.) utilizado en cualquier sentencia ABAP/4 para marcar el final de ésta, no puede ser utilizado en el bloque. Dentro del bloque, la sentencia *doble comilla* (") no marca un comienzo de comentario. La tabla utilizada es una sentencia nativa no tiene por que estar definida en el diccionario ABAP/4, por lo tanto, no es necesario declarar la tabla con la sentencia TABLES. El sistema no procesa de forma automática el campo mandante. La comunicación entre la tabla de la base de datos y el programa se realiza a través de las variables de entorno, que se identifican en la sentencia nativa gracias al carácter *dos puntos* (:). Como variable de entorno se puede utilizar campos elementales, así como campos estructurados.

- Si el resultado de la sentencia SELECT es una tabla, se puede utilizar la cláusula PERFORMING. Por cada línea leída en la sentencia SELECT, se procesa la rutina especificada en PERFORMING.

Ejemplo:

```
EXEC SQL.
  CREATE TABLE AVERI_CLNT (
        CLIENT   CHAR(3)  NOT NULL,
        ARG1     CHAR(3)  NOT NULL,
        ARG2     CHAR(3)  NOT NULL,
        FUNCTION CHAR(10) NOT NULL,
        PRIMARY KEY (CLIENT, ARG1, ARG2)
                  )
ENDEXEC.
```

En este ejemplo se crea la tabla AVERI_CLNT. Y con el siguiente ejemplo leemos dicha tabla:

```
DATA: F1(3), F2(3), F3(3).

F3 = ' 1 '
EXEC SQL PERFORMING WRITE_AVERI_CLNT.
  SELECT CLIENT, ARG1 INTO :F1, :F2 FROM AVERI_CLNT
        WHERE ARG2 = :F3
ENDEXEC.

FORM WRITE_AVERI_CLNT.
  WRITE: / F1, F2.
ENDFORM.
```

Véase también: SELECT, INSERT, UPDATE, MODIFY.

EXIT

Definición
Termina un bucle de forma incondicional.
Sintaxis:
EXIT
Con esta sentencia abandonamos todos los bloques de proceso, a excepción de los que empiezan por AT, para ir directamente a la pantalla de salida. El abandono se realiza de forma incondicional. Si la sentencia se utiliza en un evento que empieza por AT (como AT SELECTION-SCREEN, etc..) se deja de procesar el evento tratado, pero se procesa el siguiente evento lógico.

A diferencia con la sentencia STOP, el evento END-OF-SELECTION no se ejecuta. Después de ejecutarse la sentencia EXITel sistema abandona el bucle inmediatamente para continuar el proceso en la sentencia siguiente al bucle.
Para terminar el proceso de una subrutina contamos con las sentencias EXIT y CHECK. Ambas sentencias se utilizan de la misma forma que en los bucles LOOP .. ENDLOOP. Después de parar el proceso de una subrutina el sistema continúa en el punto donde se realizo la llamada (sentencia PERFORM).
Véase también: STOP, CHECK.

EXIT FROM STEP-LOOP

Definición
Esta sentencia se utiliza para abandonar un bucle LOOP en el PBO o PAI.
Sintaxis:
EXIT FROM STEP-LOOP.
Hay que recordar que un bucle LOOP en un proceso lógico de un dynpro nos permite leer una tabla de pantalla. La línea actual tratara y las posteriores no son mostradas, si el LOOPs e encuentra en el proceso PBO, o no son tratadas, si el LOOP se encuentra en el proceso PAI.

EXIT FROM SQL

Definición
Abandona un bucle de lectura realizado dentro de un bloque EXEC SQL .. ENDEXEC.
Sintaxis:
EXIT FROM SQL.
Después de que el sistema procese esta sentencia, se abandona el bloque de proceso EXEC SQL .. ENDEXEC.

EXPORT

Definición
Exporta la descripción de una estructura generada. SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Sintaxis:
EXPORT <tabla> <h> <f> ID <id>.
Exporta la descripción de la estructura <tabla>. Esta sentencia sólo se utiliza en las herramientas del repositorio de ABAP/4. Se puede producir el error EXPORT_NAMETAB_WORK_IF si el nombre de la tabla es demasiado largo.

EXPORT DYNPRO

Definición
Sentencia que se utiliza para exportar un dynpro a la base de datos. SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Sintaxis:
EXPORT DYNPRO <h> <f> <e> <m> ID <id>.
Exporta el dynpro especificado en <id>. El campo <h> y las tablas internas <f>, <e> y <m> tienen la misma estructura y significado que en la sentencia IMPORT DYNPRO.
Véase también: IMPORT DYNPRO, DELETE DYNPRO, GENERATE DYNPRO.

EXPORT TO DATABASE

Definición
Para almacenar cluster en un fichero cluster de la base de datos se utiliza esta sentencia.
Sintaxis:
EXPORT <campo11> [FROM <campo12>] <campo21> [FROM <campo22>] ...
TO DATABASE <tabla>(<área>) [CLIENT <mandante>] ID <clave>.
Almacena los objetos de datos especificados (<campo11>, <campo21>, ..) bajo un cluster con clave <clave> en el fichero <tabla>, en el área <área>. <área> es un campo de dos posiciones que el sistema almacenará en el campo RELID. <tabla> indentifica el fichero cluster. La máxima longitud para <clave> está limitada por la longitud del campo clave del fichero cluster utilizado, por ejemplo, para el fichero INDX, esta longitud es de 22 caracteres.

- Sin la cláusula CLIENT, el sistema rellena de forma automática el campo MANDT con el mandante de conexión. Con la cláusula será el valor <mandante> el que actualice el campo MANDT.
- La cláusula CLIENT debe especificarse sólo a continuación de la especificación del fichero y área. Los campos de usuario definidos en el fichero serán transportados al fichero de forma automática, pero deben ser rellenados previamente. Esta sentencia siempre sobreescribe cualquier cluster existente con la misma clave y área para un mismo mandante.

Si utilizamos tablas internas con cabecera de línea, será el contenido de la tabla y no la cabecera de la tabla la que se guarde en el fichero cluster. La cabecera de línea en ningún caso se guarda en el fichero cluster.
Ejemplo:

```
TABLES INDX.
DATA: INDXKEY LIKE INDX-SRTFD VALUE 'VALORCLAVE',
     F1(4), F2 TYPE P,
     BEGIN OF ITAB3 OCCURS 2,
       CONT(4),
     END OF ITAB3.
* Antes de exportar los campos son
* rellenados
INDX-AEDAT = SY-DATUM.
INDX-USERA = SY-UNAME.
* Exportamos a la base de datos.
EXPORT F1 F2 ITAB3 TO
      DATABASE INDX(ST) ID INDXKEY.
```

Véase también: IMPORT FROM DATABASE.

EXPORT TO DATASET

Definición
Sirve para exportar objetos a un ficheros de datos que esta en el servidor de aplicación. Esta sentencia es obsoleta y en su lugar se utiliza la sentencia TRANSFER
Sintaxis:
EXPORT <objeto1> ... <objeton> TO DATASET <fichero>(área) ID <clave>.
Esta sentencia ha sido sustituida por TRANSFER.
Véase también: TRANSFER, IMPORT FROM DATASET.

EXPORT TO MEMORY

Definición
Se utiliza para guardar objetos de un programa ABAP/4 a la memoria ABAP/4.
Sintaxis:

EXPORT <campo11> [FROM <campo12>] <campo21> [FROM <campo22>] ...
TO MEMORY ID <clave>.
Se utiliza para almacenar los datos especificados en la lista <campo11>, <campo21>, ... como cluster en la memoria ABAP/4.

- Sin la cláusula FROM el objeto de datos <campo11> se guarda con su propio nombre, con la opción FROM el objeto de dato <campo12> se guarda con el nombre <campo11>.
- La clave puede tener hasta 32 caracteres e identifica el cluster en la memoria ABAP/4. La sentencia EXPORT siempre sobrescribe cualquier cluster que exista con la misma clave.

Si utilizamos tablas internas con cabecera de línea, será el contenido de la tabla y no la cabecera la que se guarde en el cluster.
Véase también: IMPORT FROM MEMORY.

EXPORT TO SHARED BUFFER

Definición
Exporta un cluster de datos al buffer *cross-transaction application*.
Sintaxis:
EXPORT <objeto1> [FROM <g1>]... <objeton> [FROM <gn>]
TO SHARED BUFFER <tabla>(<área>) ID <clave>
[CLIENT <mandante>]
Exporta los objetos <objeto1> ... <objeton> en forma de cluster al buffer. La tabla <tabla> debe tener estructura estándar. Para identificar el área se utiliza <área>. Con <clave> se especifica la clave del cluster de datos.
La cláusula FROM sirve para especificar el área de trabajo de donde debe obtener los datos la sentencia. Y la cláusula CLIENTE sirve para indicar el mandante.
SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Véase también: IMPORT SHARED BUFFER.

EXTRACT

Definición
Escribe un field-group sobre un extracto de datos.
Sintaxis:
EXTRACT <field-group>.
Escribe todos los campos del field-group en un registro de un extracto de datos. Si se ha definido un field-group con el nombre HEADER, los registros se añaden con la clave definida en dicho field-group.
Los errores en tiempo de ejecución que se pueden producir son los siguiente:

- EXTRACT_AFTER_SORT/LOOP -> Se ha realizado un EXTRACT después de la sentencia LOOP o SORT.
- EXTRACT_FIELD_TOO_LARGE -> La longitud de los datos a pasar es más grande que la longitud del campo donde queremos guardar dicho dato.
- EXTRACT_HEADER_NOT_UNIQUE -> El field-group HEADER se ha modificado después que los registros se hayan extraído con la sentencia EXTRACT.
- EXTRACT_TOO_LARGE -> La longitud total de los datos que se van a extraer (incluido los campos del HEADER) es demasido grande.

Véase también: FIELD-GROUPS, SORT, LOOP .. ENDLOOP.

FETCH

Definición

Para obtener la siguiente línea del resultado de la orden OPEN CURSOR utilizaremos la sentencia FETCH.

Sintaxis:

FETCH NEXT CURSOR <cursor> INTO <destino>.

Cursor identifica el cursor, previamente tiene que estar abierto. Las líneas leídas sobre el área de trabajo <destino>, que se especifica en la cláusula INTO. Si se puede leer la siguiente línea la variable SY-SUBRC valdrá 0, en caso contrarió valdrá 4.

Ejemplo:

```
TABLES SBOOK.
DATA   C TYPE CURSOR,
       WA LIKE SBOOK.
OPEN CURSOR C FOR SELECT * FROM SBOOK
  WHERE
    CARRID   = 'LH '      AND
    CONNID   = '0400'     AND
    FLDATE   = '19950228'
  ORDER BY PRIMARY KEY.

DO.
  FETCH NEXT CURSOR C INTO WA.
  IF SY-SUBRC <> 0.
    CLOSE CURSOR C. EXIT.
  ENDIF.
  WRITE: / WA-BOOKID, WA-CUSTOMID,   WA-CUSTTYPE,
           WA-SMOKER, WA-LUGGWEIGHT, WA-WUNIT,
           WA-INVOICE.
ENDDO.
```

Véase también: OPEN CURSOR, CLOSE CURSOR.

FIELD

Definición

Con esta sentencia se puede validar las entradas realizadas sobre un campo. Existen dos variantes, la primera realiza la validación en la *lógica de proceso* y la segunda sobre el *modulpool*.

Sintaxis:

FIELD <campo-pantalla> VALUES (<lista-valores>).

Con esta variante validamos la entrada realizada sobre el campo <campo-pantalla> directamente en la *lógica de proceso* de la pantalla a través de una lista de valores. Si el valor introducido no es igual a uno de los valores introducidos en la lista de valores <lista-valores> el sistema vuelve a mostrar la pantalla para realizar una nueva entrada. Los posibles formatos en la lista de valores pueden ser los siguientes:

Valor	**Sintaxis**
Valor individual	(<valor>)
Negación de un valor individual	(NOT <valor>)
Varios valores individuales con o sin negación	(<valor1>, <valor2>, NOT <valor3>)
Intervalo de valores	(BETWEEN <valor1> AND <valor2>)
Negación de un intervalo de valores	(NOT BETWEEN <valor1> AND <valor2>)

Para utilizar esta variante el campo de la pantalla ha de ser de tipo CHAR o NUMC. Además todos los valores de la lista de valores han de estar en mayúsculas. Esta sentencia solo tiene sentido utilizarla en el PAI.

FIELD <campo-pantalla> MODULE <módulo>

[ON INPUT \ ON REQUEST \ ON *-INPUT \ ON CHAIN-INPUT \ ON CHAIN-REQUEST]
Con esta variante los chequeos se realizan en el modulpool. Para el campo <campo-pantalla> se ejecuta el módulo <módulo>. Si durante la ejecución del módulo se activa un mensaje de error, el sistema vuelve a presentar la pantalla y sólo permite realizar cambios sobre el campo <campo-pantalla>.
ON INPUT -> El módulo se ejecuta si el valor del campo es distinto a su valor por defecto.
ON REQUEST -> El módulo se ejecuta si el usuario cambia el valor del campo.
ON *-INPUT -> El módulo se ejecuta si el usuario introduce un asterisco en la primera posición del campo, y si el campo tiene marcado el atributo "*-entry".
ON CHAIN-INPUT -> Tiene un significado parecido a ON INPUT. El módulo se procesa si algún campo definido en la cadena CHAIN .. ENDCHAIN tiene un valor distinto al valor por defecto del campo.
ON CHAIN-REQUEST -> Tiene un significado parecido a ON REQUEST. El módulo se procesa si algún campo definido en la cadena CHAIN.. ENDCHAIN el usuario realiza alguna entrada.
Véase también: CHAIN.

FIELD GROUPS

Definición
Se utiliza para declarar un extracto de datos, también llamado grupo de campos o *field-groups*.
Sintaxis:
FIELD-GROUPS <nombre>.
Un *field-group* se utiliza para agrupar un conjunto de campo con el mismo nombre. El nombre HEADER es especial, los campos que se definan en ese *field-groups* servirán como criterio de ordenación en la sentencia SORT.
Ejemplo:
FIELD-GROUPS: HEADER, ORDER, PRODUCT.
Véase también: INSERT.

FIELD-SYMBOLS

Definición
Declara un *field-symbol*. Las asignación se realiza con la sentencia ASSIGN en tiempo de ejecución.
Sintaxis:
FIELD-SYMBOLS <fs> [<tipo>].
En la sintaxis de los field-symbols, los símbolos "<" y ">" en <fs> forman parte de la sintaxis de la sentencia. Sirven para identificar un field-symbol en el código fuente de un programa. Con esta sentencia definimos el field-symbol en código del programa. Un field-symbol puede ser especificado con o sin identificación de tipo. Si no especificamos ningún tipo, se puede asignar cualquier objeto de dato, en tiempo de ejecución, al field-symbol. Durante el proceso de asignación, el field-symbol adquiere todos los atributos del objeto de dato.
En la opción <tipo> se puede utilizar la especificación directa (con TYPE), o la indirecta (con LIKE). Con la especificación de tipo, el sistema comprueba si los tipos del field-symbol y el dato son compatibles. En caso de no ser compatibles, se produce un error en tiempo de ejecución.
FIELD-SYMBOLS <fs> STRUCTURE <estructura> DEFAULT <campo>.
En esta sentencia definimos el field-symbol estructurado <fs>, el cual apunta por defecto al campo <campo>. El sistema obliga a un campo inicial, aunque se puede modificar a posteriori. El field-symbol <fs> adquiere la estructura <fs> que puede ser cualquier field-string o estructura definida en el diccionario de datos. Las estructuras del diccionario de datos referenciadas en esta sentencia no tienen que ser declaradas con la sentencia TABLES. La estructura se debe especificar sin encerrar entre comillas (""), sin ser posible su especificación en tiempo de ejecución. Si <estructura> no tiene componentes de tipo F o I, <campo> puede ser cualquier campo interno con la longitud de

<estructura>. Si <campo> es menor que <estructura> aparecerá un error de sintaxis cuando verifiquemos el programa. Si durante la ejecución asignamos un campo al field-string, el sistema verificará la longitud. Si la longitud es menor a la de la estructura entonces se producirá un error en tiempo de ejecución. Si la estructura <estructura> tiene campos de tipo F o I, debemos especificar un campo con la misma estructura.

Ejemplo 1:

```
FIELD-SYMBOLS <PT>.
TABLES SFLIGHT.
...
ASSIGN SFLIGHT-PLANETYPE TO <PT>.
WRITE <PT>.
```

Ejemplo 2:

```
DATA SBOOK_WA LIKE SBOOK.
FIELD-SYMBOLS <SB> STRUCTURE SBOOK
                DEFAULT   SBOOK_WA.
...
WRITE: <SB>-BOOKID, <SB>-FLDATE.
```

Véase también: ASSIGN, DATA.

FIELDS

Definición

Se utiliza para direccionar un campo.

Sintaxis:

FIELDS <campo>.

Se utiliza principalmente para direccionar campos de manera estática, que posteriormente serán utilizados dinámicamente.

FORM .. ENDFORM

Definición

Esta sentencia permite definir subrutinas.

Sintaxis:

FORM <subrutina> [TABLES <parámetros-formales>]
[USING <parámetros-formales>]
[CHANGING <parámetros-formales>]

<subrutina> define el nombre de la subrutina. Con TABLES, USING y CHANGING definimos los parámetros formales de la subrutina. Para las subrutina internas nos es definir parámetros ya que la subrutina tiene acceso a todos los objetos de datos declarados en el programa principal. Para las subrutinas externas se puede elegir pasar los objetos de datos mediante parámetros o utilizar memoria compartida entre el programa que llama a la subrutina y la subrutina (sentencia COMMON PART).

Ejemplo 1:

```
PERFORM WELCOME.

FORM WELCOME.
  WRITE / 'Hola mundo'.
ENDFORM.
```

Ejemplo 2:

```
DATA: BEGIN OF X.
      INCLUDE STRUCTURE SFLIGHT.
DATA:   ADDITION(8) TYPE C,
```

```
    END OF X.
...
PERFORM U USING X.
...
FORM U USING X STRUCTURE SFLIGHT.
  WRITE: X-FLDATE.
ENDFORM.
```

Ejemplo 3:

```
TYPES: BEGIN OF FLIGHT_STRUC,
         FLCARRID LIKE SFLIGHT-CARRID,
         PRICE    LIKE  SFLIGHT-FLDATE,
       END   OF FLIGHT_STRUC.
DATA: MY_FLIGHT TYPE FLIGHT_STRUC OCCURS 0 WITH HEADER LINE,
      IBOOK1    LIKE SBOOK       OCCURS 0 WITH HEADER LINE,
      IBOOK2    LIKE IBOOK1      OCCURS 0,
      STRUC     LIKE SBOOK.
PERFORM DISPLAY TABLES MY_FLIGHT IBOOK1 IBOOK2 USING STRUC.
FORM DISPLAY TABLES P_ITAB  LIKE      MY_FLIGHT[]
                    P_BOOK1 LIKE      IBOOK1[]
                    P_BOOK2 LIKE      IBOOK2[]
             USING  P_STRU  LIKE      STRUC.
  DATA L_CARRID  LIKE P_ITAB-FLCARRID.
  ...
  WRITE: / P_STRU-CARRID, P_STRU-CONNID.
  ...
  LOOP AT P_ITAB WHERE FLCARRID = L_CARRID.
    ...
  ENDLOOP.
  ...
ENDFORM.
```

Ejemplo 4:

```
DATA: NUMBER_1 TYPE I VALUE 1,
      NUMBER_2 TYPE I VALUE 2,
      TEXT_1(10)      VALUE 'one',
      TEXT_2(10)      VALUE 'two'.
PERFORM CONFUSE USING NUMBER_1
                      NUMBER_2
                      TEXT_1
                      NUMBER_1
                      TEXT_2.
FORM CONFUSE USING PAR_NUMBER_1 TYPE I
                   PAR_NUMBER_2 TYPE I
                   PAR_TEXT_1   TYPE C
                   VALUE(PAR_V_NUMBER_1) TYPE I
                   VALUE(PAR_V_TEXT_2) TYPE C.
  ADD 3 TO PAR_V_NUMBER_1.
  ADD 4 TO PAR_NUMBER_1.
  ADD NUMBER_1 TO PAR_NUMBER_2.
  TEXT_2 = 'three'.
  PAR_TEXT_1 = PAR_V_TEXT_2.
  PAR_V_TEXT_2 = 'four'.
ENDFORM.
```

El contenido de los campos después de la llamada al PERFORM Es:

```
NUMBER_1 = 5
NUMBER_2 = 7
TEXT_1 = 'two'
```

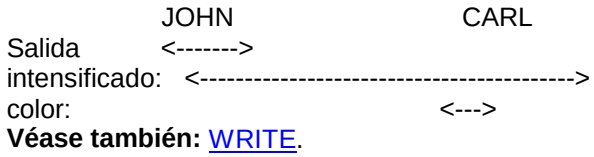
TEXT_2 = 'three'
Véase también: PERFORM

FORMAT

Definición
Se utiliza para dar formato a las sentencias de escritura en el dispositivo de salida.
Sintaxis:
FORMAT <opción1> [ON | OFF] <opción2> [ON | OFF] .....
Con esta opción variante podemos dar opciones de formato estáticamente. Las opciones de formato <opción1>, <opción2> ..., tienen efecto a partir de la ejecución de esta sentencia y hasta que nos encontremos una de desactivación (o sea, que ponga OFF). ON y OFF son opcionales, la opción por defecto es ON.
Algunas de las opciones más utilizadas son las siguientes:
Opción Significado
COLOR Color del listado.
INTENSIFIED Intensidad de salida.
INPUT Campos de entrada en el listado.
HOTSPOT Define campos sensibles al cursor (hotspot).
FORMAT <opción1> = <var1> <opción2> = <var2> ...
Con esta variante damos formato de manera dinámica. El sistema interpreta las variables <var1>, <var2>, ..., como numéricas (tipo I). Si el valor de la variable es 0 tiene el mismo efecto que OFF. Si el contenido de la variable es distinto de 0, el efecto es ON. Con la opción COLOR, el valor indica el color. Si a continuación de una sentencia FORMAT aparece una sentencia WRITE con opciones de formato, estas últimas sobreescriben las opciones de FORMAT. Las opciones de la sentencia FORMAT sólo tienen efecto sobre el evento (bloque de proceso) donde se ejecuten. Cuando se inicia un bloque de proceso los valores de todas las opciones de formato toman sus valores por defecto. Estos valores son OFF para todas las opciones menos para la opción INTENSIFIED.
FORMAT RESET.
Con esta variante ponemos todas las opciones de formato a sus valores por defecto.
FORMAT COLOR <n> [ON] INTENSIFIED [ON | OFF] INVERSE [ON | OFF].
Esta variante se utiliza para poder colores de manera estática.
FORMAT COLOR = <c> INTENSIFIED = <int> INVERSE = <inv>.
Con esta variante podemos asignar colores de manera dinámica. Los posibles valores de <n> y <c> son los siguientes:

<n>	**<c>**	**color**	**Uso recomendado por SAP**
OFF o COL_BACKGROUND	0	Depende del GUI	Color de fondo
1 o COL_HEADING	1	Azul	Cabeceras
2 o COL_NORMAL	2	Gris	Cuerpo del listado
3 o COL_TOTAL	3	Amarillo	Totales
4 o COL_KEY	4	Verde azulado	Columnas clave
5 o COL_POSITIVE	5	Verde	Significado positivo
6 o COL_NEGATIVE	6	Rojo	Significado negativo
7 o COL_GROUP	7	Violeta	Niveles

Ejemplo:
```
FORMAT INTENSIFIED INPUT.
WRITE 5 'JOHN'.
FORMAT INPUT OFF.
WRITE 40 'CARL'COLOR COL_GROUP.
```
La salida en pantalla es la siguiente:
```
....+....10...+....20...+....30...+....40...+
```



Véase también: WRITE.

FREE

Definición

Se utiliza para liberar el espacio de memoria que se reserva después de utilizar las sentencias REFRESH o CLEAR.

Sintaxis:

FREE <tabla>.

Véase también: CLEAR, REFRESH.

FREE MEMORY

Definición

Se utiliza para borrar cluster de la memoria ABAP/4.

Sintaxis:

FREE MEMORY [ID <clave>].

Sin la cláusula ID, el sistema borra todos los clusters creados en la memoria intermedia ABAP/4 (y con ello se puede producir inconsistencias debido a que hay cluster que son generador por programas de SAP),. Con la cláusula ID sólo se borra el cluster bajo la clave <clave>.

Véase también: EXPORT TO MEMORY, IMPORT FROM MEMORY.

FREE OBJECT

Definición

Libera el espacio utilizado en memoria por un objeto OLE.

Sintaxis:

FREE OBJECT <objeto>.

El objeto no podrá ser de nuevo utilizado. Esta sentencia pertenece a un conjunto de sentencias que permitan gestionar objetos externos al sistema. Actualmente solo se puede trabajar con objetos OLE2.

FUNCTION .. ENDFUNCTION

Definición

Define un módulo de función.

Sintaxis:

FUNCTION <función>

...

ENDFUNCTION.

Todas las sentencias incluidas entre FUNCTION y ENDFUNCTION serán ejecutadas cuando se realice una llamada a la función.

Para crear, modificar o visualizar un módulo tenemos que ir a (desde el menú principal) *Herramientas -> Abap/4 Workbench -> Módulo función*. O ir directamente a través de la transacción SE37.

Véase también: CALL FUNCTION.

FUNCTION-POOL

Definición
Esta sentencia es equivalente a la sentencia REPORT. Con ella introducimos un grupo de funciones.
Sintaxis:
FUNCTION-POOL <grupo>.
Un grupo de funciones contiene módulos de función que están encabezados con la sentencia FUNCTION.
Véase también: CALL FUNCTION, REPORT.

GENERATE DYNPRO

Definición
Genera una dynpro de la base de datos.
Sintaxis:
GENERATE DYNPRO <h> <f> <e> <m> ID <id> MESSAGE <c1> LINE <c2>
WORD <c3> [OFFSET <c4>] [TRACE-FILE <tabla>]
La información necesaria para generar un dynpro se toma del campo <h> y de las tablas internas <f>,<e> y <m>. El campo <h> y las tablas internas <f>, <e> y <m> tienen la misma estructura y significado que en la sentencia IMPORT DYNPRO. Si se produce un error al generar el mensaje de error se guarda en <c1>, la línea donde se produce el error se guarda en <c2> y la palabra con el error se guarda en <c3>.
La variable SY-SUBRC devuelve los siguientes valores:

- 0 -> No existen errores y la dynpro se ha generado correctamente.
- 4 -> El error se encuentra en la lógica de proceso.
- 8 -> El error se encuentra en los campos del dynpro.

El significado de las cláusulas es el siguiente:

- OFFSET -> Si ocurre un error el campo <c4> contiene la posición de la palabra incorrecta.
- TRACE-TABLE -> Cualquier salida del trazador se deposita en la tabla <tabla>.

SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Véase también: IMPORT DYNPRO, EXPORT DYNPRO, DELETE DYNPRO.

GENERATE REPORT

Definición
Genera un programa
Sintaxis:
GENERATE REPORT <programa> [MESSAGE <C1>] [INCLUDE <c2>] [LINE C3]
[WORD <c4>] [OFFSET <c5>] [TRACE-FILE <c6>] [DIRECTORY ENTRY <c7>]
[WITHOUT SELECTION-SCREEN]
Genera el programa especificado en <programa>. Si el programa es un report (programas del tipo I) también la pantalla de selección se genera. La variable SY-SUBRC puede tomar los siguiente valores:

- 0 -> El programa se genera correctamente.
- 4 -> Error de sintaxis y el report no se genera.
- 8 -> Error de generación y el report no se genera.

- 12 -> Error en la generación de la pantalla de selección y el report no se genera.

El significado de las cláusulas es el siguiente:

- MESAGE -> Cuando ocurre un error de sintaxis el error se almacena en la variable <c1>.
- INCLUDE -> Cuando ocurre un error de sintaxis el nombre del programa include relacionado se almacena en el campo <c2>.
- LINE -> Cuando ocurre un error de sintaxis el número de línea errónea se almacena en <c3>.
- WORD -> Cuando ocurre un error de sintaxis la palabra incorrecta se almacena en <c4>.
- OFFSET -> Cuando ocurre un error de sintaxis la posición de la palabra incorrecta se almacena en el campo <c5>.
- TRACE-FILE -> La traza de salida del programa se almacena en el fichero <c6>. Con esta cláusula se activa de forma automática el trazador de programas.
- DIRECTORY ENTRY -> Los atributos del programa requeridos para la comprobación sintáctica son tomados del campo <c7>. Este campo debe tener la estructura de la tabla TRDIR.
- WITHOUT SELECTION-SCREEN -> Con esta cláusula no se genera la pantalla de selección.

SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Véase también: SYNTAX-CHECK.

GENERATE SUBROUTINE POOL

Definición
Generamos en memoria un pool de subrutinas.
Sintaxis:
GENERATE SUBROUTINE POOL <tabla> NAME <nombre> [MESSAGE <c1>] [INCLUDE <c2>] [LINE <c3>]
[WORD <c4>] [OFFSET <c5>] [TRACE-FILE <c6>]
Genera un pool de subrutinas temporales en memoria. El código fuente de las subrutinas se encuentra en la tabla interna <tabla>. El campo <nombre> contiene el nombre bajo el cual las rutinas FORM puede ser direccionadas a través de la sentencia PERFORM. La variable SY-SUBRC puede tomar los siguiente valores:

- 0 -> Generación correcta.
- 4 -> Error de sintaxis.
- 8 -> Error de generación.

En contraste con la sentencia GENERATE REPORT, el código fuente se encuentra en una tabla interna y no en la base de datos. La versión generada sólo se encuentra en memoria principal. Las subrutinas generadas con esta sentencia sólo tiene validez para el programa que las ha generado. Hasta 36 pools de rutinas se puede generar en un solo programa.
Las cláusula MESSAGE, INCLUDE, LINE, WORD, OFFSET y TRACE-FILE tiene el mismo significado que en la sentencia GENERATE REPORT.
SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Véase también: SYNTAX-CHECK.

GET

Definición
Identifica un bucle de lectura.

Sintaxis:
GET <tabla> [FIELDS <lista>]
GET <tabla> LATE.
Este evento es el más importante en los programas de informes que utilicen bases de datos lógicas. Por cada ejecución de la sentencia GET se realiza una llamada al programa de bases de datos lógica (PBDL), a su correspondiente sentencia PUT. En el PBDL se realiza la lectura de la base de datos, para pasar la información al programa principal sobre el área de trabajo de la tabla referenciada <tabla>.
Sin la cláusula FIELDS, el PBDL devuelve todos los campos de la tabla referenciada. Con la cláusula FIELDS sólo serán devueltos los campos especificados en <lista>.
Ejemplo:

```
TABLES: SFLIGHT, SBOOK.
DATA SMOKERS TYPE I.
GET SFLIGHT.
  ULINE.
  WRITE: / SFLIGHT-SEATSMAX,
           SFLIGHT-SEATSOCC.
  SMOKERS = 0.
GET SBOOK FIELDS SMOKER.
  CHECK SBOOK-SMOKER <> SPACE.
  ADD 1 TO SMOKERS.
GET SFLIGHT LATE FIELDS SEATSMAX SEATSOCC.
  WRITE SMOKERS.
```

Véase también: PUT.

GET CURSOR

Definición
Recupera información a partir de la posición del cursor.
Sintaxis:
GET CURSOR FIELD <campo> [OFFSET <offset>] [LINE <línea>]
[VALUE <valor>] [LENGTH <longitud>]
Esta variante transfiere el nombre del campo sobre el cual se encuentra posicionando el cursor. Si el cursor se encuentra sobre algún campo, SY-SUBRC toma el valor 0, en caso contrario toma el valor 4. El sistema transporta el nombre de variables globales, campos simbólicos, constantes o parámetros referenciados de subrutinas. Para literales y campos locales, SY-SUBRC vale 0 pero el campo <campo> toma el valor SPACE. El significado de las cláusulas es el siguiente:

- OFFSET -> Contiene la posición relativa del cursor sobre el campo. La primera posición tiene el offset 0.
- LINE -> Contiene el número de la línea en el listado (variable SY-LILLI).
- VALUE -> Contiene el string de salida, incluyendo los formatos, del campo sobre el que está situado el cursor.
- LENGTH -> Contiene la longitud de salida del campo sobre el que está situado el cursor.

GET CURSOR LINE <línea> [OFFSET <offset>] [VALUE <valor>] [LENGTH <longitud>]
Transfiere a <línea> el número de la línea sobre la cual está situado el cursor. Si el cursor está sobre alguna línea del listado SY-SUBRC valdrá 0, en caso contrario valdrá 4. Esta variante se puede utilizar para validar que el usuario ha seleccionado una línea. El significado de las cláusulas es el siguiente:

- OFFSET -> Contiene la posición del cursor en la línea seleccionada.
- VALUE -> Contiene el string de salida de la línea donde se encuentra el cursor.
- LENGTH -> Contiene la longitud de salida donde se encuentra el cursor.

Ejemplo:

```
DATA: CURSORFIELD(20),
```

```
    GLOB_FIELD(20)    VALUE 'campo global',
    REF_PARAMETER(30) VALUE 'parámetro por referencia',
    VAL_PARAMETER(30) VALUE 'parámetro por valor',
    FIELD_SYMBOL(20)  VALUE 'field-symbol'.
FIELD-SYMBOLS: <F>.
PERFORM WRITE_LIST USING REF_PARAMETER VAL_PARAMETER.
ASSIGN GLOB_FIELD TO <F>.
AT LINE-SELECTION.
  GET CURSOR FIELD CURSORFIELD.
  WRITE: /   CURSORFIELD, SY-SUBRC.
FORM WRITE_LIST USING RP VALUE(VP).
  DATA: LOK_FIELD(20)  VALUE 'lokal field'.
  ASSIGN FIELD_SYMBOL TO <F>.
  WRITE: /  GLOB_FIELD,  /  LOK_FIELD,
         /  RP,          /  VP,
         /  'literal',   /  FIELD_SYMBOL.
ENDFORM.
```

Véase también: SET CURSOR.

GET LOCALE LANGUAGE

Definición

Recupera los valores actuales del entorno texto.

Sintaxis:

GET LOCALE LANGUAGE <lenguaje> COUNTRY <país> MODIFIER <modificador>.

Recupera los valores actuales del entorno. En el campo <lenguaje> recupera el valor actual de lenguaje, en el campo <país> recupera el país actual del lenguaje, y en el campo <versión> recupera la versión del país del lenguaje actual. Por ejemplo se puede tener distintas versiones en función del país: Estados Unidos, Gran Bretaña, etc.. y además se puede especificar, para un mismo país, distintas versiones de idiomas.

GET PARAMETER

Definición

Lee el valor del parámetro actualizando el contenido del campo.

Sintaxis:

GET PARAMETER ID '<parametro>' FIELD <campos>.

El sistema lee el valor del parámetro <parámetro> actualizando el contenido del campo <campo>. <parámetro> siempre es un literal alfanumérico de 3 posiciones y debe ir con las comillas (' '). El valor del parámetro de memoria no varia.

A partir de la versión 4.x se puede utilizar la variable SY-SUBRC. En esta versión si conseguimos leer el parámetro de memoria correctamente SY-SUBRC valdrá 0, en caso contrario valdrá 4. En versiones anteriores había que introducir un valor erróneo y una vez ejecutado la sentencia GET PARAMETER comprobar si el valor había cambiado.

Ejemplo:

```
DATA : REPID(8).
GET PARAMETER ID 'RID' FIELD REPID.
```

Véase también: SET PARAMETER.

GET PROPERTY

Definición
Recupera las propiedades de un objeto OLE.
Sintaxis:
GET PROPERTY OF <objeto> = <campo> [NO FLUSH].
Recupera la propiedad <propiedad> del objeto <objeto> sobre el campo <campo>.
Con la cláusula NO FLUSH el proceso OLE continúa aunque la siguiente sentencia a ejecutar no sea una sentencia OLE. Esta sentencia pertenece a un conjunto de sentencias que permitan gestionar objetos externos al sistema. Actualmente solo se puede trabajar con objetos OLE2.
Ejemplo:

```
INCLUDE OLE2INCL.
DATA: EXCEL   TYPE OLE2_OBJECT,
      VISIBLE TYPE I.
CREATE OBJECT EXCEL 'Excel.Application'.
GET PROPERTY OF EXCEL 'Visible' = VISIBLE.
```

Véase también: SET PROPERTY, CALL METHOD, CREATE OBJECT, FREE OBJECT.

GET RUN TIME

Definición
Se utiliza para obtener el tiempo de ejecución de un programa.
Sintaxis:
GET RUN TIME FIELD <campo>.
La primera vez que se ejecuta esta sentencia se inicializa el campo <campo> (que ha de ser de tipo I) a 0. Para las llamadas posteriores a la sentencia GET RUN TIME el sistema actualiza el campo <campo> con el tiempo transcurrido. Si el servidor de aplicación tiene 2 o más procesadores puede ocurrir que haya fluctuaciones el los tiempos devueltos. El tiempo que devuelve la sentencia esta en microsegundos.
Para realizar análisis más complejos utilizaremos la transacción SE30.
Ejemplo:

```
DATA: T1   TYPE I,
      T2   TYPE I,
      TMIN TYPE I.
DATA: F1(4000), F2 LIKE F1.
TMIN = 1000000.
DO 10 TIMES.
  GET RUN TIME FIELD T1.
    MOVE F1 TO F2.        "Medición la sentencia MOVE
  GET RUN TIME FIELD T2.
  T2 = T2 - T1. IF T2 < TMIN. TMIN = T2. ENDIF.
ENDDO.
WRITE: 'Mover 4000 bytes toma', TMIN, 'microsegundos'.
```

GET TIME

Definición
Se utiliza para obtener la hora del sistema.
Sintaxis:
GET TIME [FIELD <hora>].
Se actualiza la variable SY-UZEIT con la hora actual del sistema. También se inicia el campo SY-DATUM, SY-TIMLO, SY-DATLO, SY-TSTLO y SY-ZONLO.

Con la cláusula FIELD, actualiza el campo <hora> con la hora actual del sistema. Los campos anteriores no sufren cambios.

HIDE

Definición
La sentencia HIDE permite asociar a cada línea de un listado ciertos campos, que posteriormente podrán ser leídos. Estos campos puede no salir en el listado.
Sintaxis:
HIDE <campo>.
La orden HIDE almacena el contenido del campo <campo> en relación con la línea actual de salida. El campo <campo> puede o no puede aparecer en el listado.
Véase también: AT LINE SELECTION, AT PFx, AT USER-COMMAND, READ LINE, WRITE.

IF .. ENDIF

Definición
Con esta sentencia permite bifurcar el flujo de un programa en bloques de sentencias dependiendo de una condición.
Sintaxis:
IF <condición>.
  [<bloque-sentencias>]
[ELSEIF <condición>.]
  [<bloque-sentencias>]
[ELSEIF <condición>.]
  [<bloque-sentencias>]
...
[ELSE.]
  [<bloque-sentencias>]
ENDIF.
<condición> puede ser cualquier expresión lógica. <bloque-sentencias> puede ser cualquier combinación de sentencias. Un bloque de sentencias estará limitado por las palabras clave IF, ELSEIF, ELSE y ENDIF. La condiciones se evalúan de forma secuencial. Cuando el resultado de una condición es TRUE se ejecuta el bloque de sentencias de esa condición. Una vez se haya ejecutado el bloque de sentencias el proceso continua a partir de la cláusula ENDIF. Las cláusulas IF y ENDIF son obligatorias.
Se puede anidar tantas sentencias IF como queramos, y dentro del bloque se sentencias no puede haber ninguna sentencia de evento (START-OF-SELECTION).
Véase también: CHECK.

IMPORT DIRECTORY FROM DATABASE

Definición
Se utiliza para crear una tabla con el directorio de objetos de datos contenidos en los clusters existentes en un fichero cluster.
Sintaxis:
IMPORT DIRECTORY INTO <directorio> FROM DATABASE <tabla>(<área>) [CLIENT <mandante>]
    ID <clave>.
Esta sentencia crea una lista con los objetos de datos contenidos en todos los clusters existentes en el fichero <tabla>, para el área <área>. Esta información se almacena en la tabla interna <directorio>. La tabla <tabla> se debe declarar con la sentencia TABLES.

Esta sentencia también lee información contenida en los campos de usuario existentes en el fichero cluster. Si la tabla directorio puede ser creada SY-SUBRC valdrá 0, en caso contrario 4. La tabla interna debe tener la estructura que la tabla de diccionario CDIR.

```
TABLES INDX.
DATA: INDXKEY LIKE INDX-SRTFD,
      F1(4), F2 TYPE P,
      BEGIN OF TAB3 OCCURS 10,
        CONT(4),
      END OF TAB3,
      BEGIN OF DIRTAB OCCURS 10.
        INCLUDE STRUCTURE CDIR.
DATA  END OF DIRTAB.
INDXKEY = 'INDXKEY'.
EXPORT F1 F2 TAB3 TO
       DATABASE INDX(ST) ID INDXKEY.    " TAB3 tiene 17 entradas
...
IMPORT DIRECTORY INTO DIRTAB FROM DATABASE INDX(ST) ID INDXKEY.
```

La tabla DIRTAB contendrá lo siguiente:

```
NAME OTYPE FTYPE TFILL FLENG
-----------------------------------
F1 F C 0 4
F2 F P 0 8
TAB3 T C 17 4
```

Véase también: EXPORT, INCLUDE STRUCTURE.

IMPORT DYNPRO

Definición

Se utiliza para importar un dynpro.

Sintaxis:

IMPORT DYNPRO <h> <f> <e> <m> <id>.

Importa el dynpro especificado en el campo <id>. El nombre de un dynpro está compuesto de una campo de 8 caracteres, nombre del modulpool (pool de módulos de diálogo) y un campo de 4 caracteres que identifica el número de pantalla. En <h> se carga información de la cabecera de pantalla (estructura D020S). En la tabla interna <f> se carga la lista de campos (estructura D022S). En la tabla interna <e> se carga el proceso lógico (estructura D023S). La variable SY-SUBRC valdrá 0 si la dynpro se ha importado y tomara el valor 4 en caso contrario.

SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.

Véase también: EXPORT DYNPRO, DELETE DYNPRO, GENERATE DYNPRO, SYNTAX-CHECK FOR DYNPRO.

IMPORT FROM DATABASE

Definición

Se utiliza para leer objetos de un cluster almacenado en un fichero cluster.

Sintaxis:

IMPORT <campo11> [TO <campo12>] <campo21> [TO <campo22>] FROM DATABASE ...
<tabla>(<área>) [CLIENT <mandante>] {ID <clave> | MAJOR-ID <mayor>
[MINOR-ID <menor>] }.

Lee los objetos de datos especificados en la lista <campo11>, <campo21>, ... de un cluster de datos almacenados en fichero cluster <tabla> (esta tabla tiene que estar declaradas en la sentencia TABLES).

- Sin la cláusula TO el objeto de datos <campo11> de la base de datos se asigna sobre el objeto de datos con el mismo nombre definido en el programa. Con la cláusula TO el objeto de datos <campo11> leído de la base de datos se almacena sobre el campo <campo12> declarado en el programa.
- <área> es un campo o literal alfanumérico de dos posiciones que identifica el área del fichero cluster. Recordemos que el área se almacena en el campo RELID del fichero cluster.
- <clave> identifica la clave del fichero cluster y tendrá una longitud máxima del fichero. Por ejemplo, para el fichero INDX, la longitud máxima es de 22 caracteres.
- Se puede utiliza la cláusula MAJOR-ID en vez de la cláusula ID. De esta forma el cluster seleccionado será el primero en que coincida parte de la clave. del cluster con <mayor>. Si aparte especificamos la opción MINOR-ID, el cluster seleccionado será el que en la segunda parte del nombre del cluster (la primera parte es la de la cláusula MAJOR-ID) sea mayor o igual a <menor>.

No es necesario leer todos los objetos almacenados en el cluster sólo habrá que especificar en la lista de objetos aquellos que deseemos leer. Si no existe cluster para un fichero, área y clave especificado la variable SY-SUBRC valdrá 4 en caso contrario valdrá 0. Si algún objeto de dato especificado en la lista no existe en el cluster, éste permanece sin cambios. En tiempo de ejecución del sistema comprueba en el momento de ejecutar la sentencia que los tipos de los objetos de datos almacenados en el cluster y los provistos en la sentencia son compatibles.
Véase también: EXPORT TO DATABASE.

IMPORT FROM DATASET

Definición
Recupera registros de un fichero de datos. Se recomienda utilizar la sentencia READ DATASET debido a que esta sentencia ya es obsoleta y no se debe utilizar.
Sintaxis:
IMPORT <campo> <tabla> ... FROM DATASET <fichero>(<área>) ID <clave>.
Véase también: EXPORT TO DATASET, READ DATASET.

IMPORT FROM LOGFILE

Definición
Recupera objetos de datos del área de modificación de datos de la base de datos. SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Sintaxis:
IMPORT <campo> <tabla> FROM LOGFILE ID <clave>.
Recupera objetos de datos (variables, field-strings, tablas, etc.) del área de *updates* de la base de datos. Se recomienda no utilizar esta sentencia.

IMPORT FROM MEMORY

Definición
Se utiliza para leer objetos de un programa ABAP/4 de la memoria ABAP/4.
Sintaxis:
IMPORT <campo11> [TO <campo12>] <campo21> [TO <campo22>] ...
FROM MEMORY ID <clave>.
Lee los objetos especificados en la lista <campo11>, <campo21>, ... del cluster almacenado en la memoria ABAP/4.

Sin la cláusula TO el objeto de dato almacenado en el cluster se guarda en el objeto definido en el programa con el mismo nombre. Con la cláusula TO, el objeto de dato <campo11> almacenado en el cluster se guarda en el objeto de dato <campo12> que esta definido en el programa.
La clave <clave> puede tener hasta 32 caracteres e identifica el cluster de la memoria ABAP/4. No es necesario leer todos los objetos de datos almacenados en el cluster. Si intentamos leer un cluster bajo una clave que no existe SY-SUBRC vale 4, en caso contrario, o sea, que exista la clave aunque no indiquemos un objeto de datos la variable valdrá 0.
Si el objeto de dato especificado no existe el cluster no varia. Esta sentencia no comprueba si la estructura de los objetos en memoria coincide con los especificados en la lista. El transporte se realiza bit a bit, por lo tanto, si la estructuras no coinciden se puede generar inconsistencias.
Véase también: EXPORT TO MEMORY.

IMPORT SHARED BUFFER

Definición
SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Sintaxis:
IMPORT <campo> <tabla> FROM SHARED BUFFER <tabla>(<área>) ID <clave>.
Recupera un cluster de datos creado previamente en el buffer *cross-transaction application*, bajo la clave <clave>. Se recomienda no utilizar esta sentencia.

INCLUDE

Definición
Se utiliza para incluir parte código fuente a un programa.
Sintaxis:
INCLUDE <include>.
El programa <include> se include en el programa donde se ha utilizado la sentencia. Las sentencias del objeto INCLUDE se chequean en la comprobación sintáctica. Esta sentencia se utiliza para dividir programa grandes en unidades más pequeñas y manejables, también se suele utilizar para definir sólo una vez las áreas de memoria común o *common part*.
Esta sentencia no se puede expandir en más de una línea, ni puede compartir línea con otras sentencias. Si un programa es tipo I (include), sólo podrá ser utilizado como include en otros programas. Un ejemplo de como el sistema utiliza las includes es el programa estándar RSINCL00.
Ejemplo:
INCLUDE LSPRITOP.

INCLUDE STRUCTURE

Definición
La sentencia INCLUDE STRUCTURE se utiliza para incluir estructuras del diccionario de datos en nuestros programas.
Sintaxis:
INCLUDE STRUCTURE <estructura>.
Esta sentencia se utiliza en combinación de las sentencias DATA o TYPES para la definición de objetos de datos o tipos de datos respectivamente. SAP no recomienda su uso, en su lugar, se puede utilizar las cláusula TYPE en las sentencia DATA o TYPE.
Ejemplo:

```
DATA: BEGIN OF rec.
      INCLUDE STRUCTURE subRec.
DATA: END OF rec.
```

Es equivalente:

DATA rec LIKE subRec.
Véase también: INCLUDE TYPE.

INCLUDE TYPE

Definición
Se utiliza para incluir en una definición un tipo ya definido.
Sintaxis:
INCLUDE TYPE <estructura>.
Cuando definimos una estructura esta sentencia copia en dicha estructura los campos de la estructura <estructura>.
Véase también: INCLUDE STRUCTURE.

INFOTYPES

Definición
Declara un objeto de tipo infotipo, los infotipos son exclusivos del módulo de HR (Recursos humanos).
Sintaxis:
INFOTYPE <número> [NAME <nombre> [OCCURS <tabla>]]
[MODE <n>]
[VALID FROM <inicio> TO <final>].
<número> determina el número de infotipo, con la cláusula NAME declaramos una tabla interna con la misma estructura que el infotipo.
La cláusula OCCURS tiene el mismo significado que el de la sentencia DATA.
La cláusula VALID sólo se puede utilizar junto a la sentencia GET con la tabla PERNR. Recupera sólo aquellos registros del infotipo válidos entre las fecha <inicio> y <final>.
Ejemplo:

```
INFOTYPES: 0007 VALID FROM 19910101
                TO   19911231.
```

INITIALIZATION

Definición
Con este evento podemos inicializar los campos de una pantalla de selección (o cualquier otro campo u otra variable) si este programa la posee antes de que aparezca.
Sintaxis:
INITIALIZATION.
Si queremos que los campos que aparecen en pantalla de selección (o cualquier otra variable o campo) tengan ciertos valores o, en general, si queremos realizar cualquier acción antes de que aparezca la pantalla de selección.
Hay que recordar que en nuestro ABAP/4 aparece una pantalla de selección por dos motivos. El primero es porque hayamos puesto en nuestro programa alguna sentencia de criterio de selección y segundo en el programa accedemos a una base de datos lógica que tenga criterios de selección.
En un principio no sabemos que campos de selección aparecen en un programa que utiliza base de datos lógica. Para saber que campos podemos ir a la transacción SLDB (Base de datos lógica) o por menú (desde la pantalla principal) sería: *Herramientas, Workbench, Desarrollo, Entorno de programación, base de datos lógica*. Con esta transacción podemos ver los campos definidos en la pantalla de selección. Otra forma es nos posicionamos en el campo, que queremos saber el

nombre, y pulsamos F1 (la tecla de ayuda) y a continuación pulsaremos el botón que pone: *Información técnica*. En la pantalla resultante tendremos entre otras cosas el nombre del campo.
Ejemplo:

```
PARAMETERS QUAL_DAY TYPE D DEFAULT SY-DATUM.
INITIALIZATION.
  QUAL_DAY+6(2) = '01'.
  QUAL_DAY      = QUAL_DAY - 1.
```

Véase también: AT SELECTION-SCREEN, START-OF-SELECTION.

INSERT

Definición
Se utiliza para insertar líneas en una tabla interna antes de una posición determinada.
Sintaxis:
INSERT [<área-trabajo> INTO | INITIAL LINE INTO] <tabla> [INDEX <índice>].
Con <área-trabajo> INTO especificamos el área de trabajo que vamos a insertar. Si omitimos esta opción, el área de trabajo que se va a insertar será la asociada con la tabla interna. El significado de las cláusulas es el siguiente:

- INITIAL LINE TO -> inserta una línea con los valores iniciales de cada campo.
- INDEX -> Inserta una línea antes de la posición indicada en <índice> y la nueva tiene el índice <índice>.

Si la tabla tiene <índice> - 1 entradas la línea se insertara en la última posición. Si la tabla tiene menos de <índice> - 1 posiciones la línea no se insertara. Si se produce algún error al insertar la línea la variable SY-SUBRC valdrá 4, en caso contrario valdrá 0.
Si la sentencia se utiliza sin la cláusula INDEX sólo se podrá utilizar en un LOOP .. ENDLOOP. El sistema insertara la línea antes de la línea tratada en el bucle.
INSERT LINES OF <tabla1> [FROM <n1>] [TO <n2>] INTO <tabla2> [INDEX <índice>]

- Si no se especifican las cláusulas FROM <n1> y TO <n2> la tabla <tabla1> se inserta entera sobre la tabla <tabla2>. Especificando estas cláusulas se puede indicar la primera y la última línea que se va insertar. <n1> y <n2> son índice de la tabla interna.
- Si especificamos la cláusula INDEX <índice>, la línea se insertará antes de la línea con el índice <índice>. Si la sentencia INSERT se utiliza sin la cláusula INDEX sólo se podrá ser utilizada en un bucle LOOP .. ENDLOOP. El sistema inserta la línea tratada en el bucle. Después de ejecutarse la sentencia la variable del sistema SY-TABIX tiene el índice de la última línea añadida. Dependiendo del tamaño de la tabla interna que hay que copiar, con este método el proceso en 20 veces más rápido que insertarlas línea a línea.

La sentencia INSERT puede ser utilizada para insertar una o varias líneas (a partir de una tabla interna) en una tabla de la base de datos. Si no sabemos si la clave primaria existe, se debe utilizar la sentencia MODIFY. Se puede utilizar una vista como especificación de tabla, siempre y cuando la vista haga referencia a una sola tabla.
INSERT INTO { <tabla> | (<campo>) } [CLIENT SPECIFIED] VALUES <área>.
Esta sentencia se utiliza para insertar una línea a una tabla de diccionario. El contenido del área de trabajo <área> será escrito en la tabla de la base de datos <tabla>. La tabla tiene que estar declarada en la sentencia TABLES y la área de trabajo <área> tiene que tener la misma estructura que <tabla> (se recomienda utilizar el operador LIKE en la sentencia DATA para declararla de la misma estructura).
Si la tabla no contiene una línea con la misma clave primaria que la especificada en el área de trabajo <área>, la variable SY-SUBRC valdrá 0, en caso contrario, la línea no se insertara y la variable SY-SUBRC valdrá 4.
Para especificar el nombre de la tabla en tiempo de ejecución utilizaremos la opción (<campo>). En tiempo de ejecución se debe rellenar con el nombre de la tabla donde realizaremos la operación de inserción.
INSERT { <tabla> | (<campo>) } [CLIENT SPECIFIED] [FROM <field-string>].
Esta variante tiene el mismo efecto que la variante vista anteriormente.

- Se trata de una versión *simplificada* donde la cláusula INTO no se especifica y en lugar de VALUES se pone FROM.
- Si no se especifica la cláusula FROM, los datos que hay que insertar serán los del área de trabajo de la tabla <tabla>. Si especificamos el nombre de la tabla en tiempo de ejecución, la cláusula FROM es obligatoria.

INSERT { <tabla> | (<campo>) } [CLIENT SPECIFIED] FROM TABLE <tabla-interna>
[ACCEPTING DUPLICATE KEYS].
En esta variante se puede insertar más de una línea en la tabla de diccionario. La estructura de la tabla interna debe ser la misma que la estructura de la tabla en la que se van a insertar líneas. Las líneas serán ejecutadas en una sola operación. Si la sentencia se ejecuta correctamente SY-SUBRC valdrá 0, en caso contrario valdrá 4.
Si alguna de las líneas de la tabla interna no puede ser insertada, el sistema activa un error en tiempo de ejecución (suele ser por duplicación de claves). Para evitar este error en tiempo de ejecución, se utiliza la cláusula ACCEPTING DUPLICATE KEYS. Con estos forzamos al sistema a saltar las líneas que no puede insertar. La variable SY-DBCNT tendremos las líneas insertadas y la variable SY-SUBRC valdrá 4 cuando se salte alguna línea.
Es más eficiente utilizar esta última sentencia para insertar un conjunto de líneas, que utilizar las otras variantes que las insertaría de en una en una.
Ejemplo 1:

```
DATA: VALUE TYPE I,
      ITAB  TYPE I OCCURS 100 WITH HEADER LINE.
ITAB  = 5.
VALUE = 36.
INSERT ITAB INDEX 1.
INSERT VALUE INTO ITAB INDEX 2.
INSERT INITIAL LINE INTO ITAB INDEX 2.
```

Ejemplo 2:

```
TYPES NAME(10) TYPE C.
DATA: NAME_TAB_1 TYPE NAME OCCURS 5,
      NAME_TAB_2 TYPE NAME OCCURS 5.
APPEND 'Alice'  TO NAME_TAB_1.
APPEND 'Martha' TO NAME_TAB_1.
APPEND 'Ruth'   TO NAME_TAB_1.
APPEND 'Harry'  TO NAME_TAB_2.
APPEND 'Walter' TO NAME_TAB_2.
INSERT LINES OF NAME_TAB_1 FROM 2 INTO NAME_TAB_2 INDEX 2.
```

Véase también: APPEND, COLLECT, MODIFY, LOOP, READ TABLE.

INSERT .. INTO

Definición
Declara los campos que formaran parte de de una agrupación de campos o *field-group*.
Sintaxis:
INSERT <campo1> ... <campon> INTO <field-group>.
Los campos <campo1>, ..., <campon> deben estar declarados y la agrupación de campos <field-group> debe estar declarada con la sentencia FIELD-GROUPS.
Véase también: FIELD-GROUPS.

INSERT REPORT

Definición
Se utiliza para insertar un programa en el sistema.
Sintaxis:
INSERT REPORT <programa> FROM <tabla>.
El código fuente del programa esta en una tabla interna <tabla>. Las líneas de la tabla interna no puede tener más de 72 caracteres de longitud. Los atributos del programa (tipo, fecha, ...) son asignados automáticamente por el sistema. Los atributos se pueden modificar desde el editor de programas o a través de la tabla TRDIR. Los errores en tiempo de ejecución son los siguientes:

- INSERT_PROGRAM_INTERNAL_NAME -> El nombre del programa esta reservado para el sistema (%_T).
- INSERT_PROGRAM_NAME_BLANK -> El nombre del programa no puede contener espacios en blanco.
- INSERT_PROGRAM_NAME_TOO_LONG -> El nombre del programa no puede superar los 8 caracteres.
- INSERT_REPORT_LINE_TOO_LONG -> Una línea o más de una no puede superar los 72 caracteres.

Véase también: DELETE REPORT, READ REPORT, GENERATE REPORT.

INSERT TEXTPOOL

Definición
Se utiliza para añadir elementos de texto en la base de datos.
Sintaxis:
INSERT TEXTPOOL <programa> FROM <tabla> LANGUAGE <lenguaje>.
Añade a la base de datos los elementos contenidos en la tabla interna <tabla> al programa <programa> en el lenguaje <lenguaje>. La tabla interna debe tener la estructura TEXTPOOL.
Ejemplo:

```
DATA: PROGRAM(8) VALUE 'PROGNAME',
      TAB LIKE TEXTPOOL OCCURS 50 WITH HEADER LINE.
TAB-ID = 'T'. TAB-KEY = SPACE.  TAB-ENTRY = 'Ventas'.
APPEND TAB.
TAB-ID = 'I'. TAB-KEY = '200'.  TAB-ENTRY = 'Impuestos'.
APPEND TAB.
TAB-ID = 'H'. TAB-KEY = '001'.  TAB-ENTRY = 'Nombre   Edad'.
APPEND TAB.
TAB-ID = 'S'. TAB-KEY = 'CUST'. TAB-ENTRY = 'Vendedor'.
APPEND TAB.
TAB-ID = 'R'. TAB-KEY = SPACE.  TAB-ENTRY = 'Test programa'.
APPEND TAB.
SORT TAB BY ID KEY.
INSERT TEXTPOOL PROGRAM FROM TAB LANGUAGE SY-LANGU.
```

Véase también: DELETE REPORT, READ REPORT, GENERATE REPORT.

LEAVE

Definición
Abandona un modo *CALL*.
Sintaxis:
LEAVE.

Abandona un modo introducido con las sentencias CALL TRANSACTION, CALL DIALOG o SUBMIT ... AND RETURN. El control vuelve al punto donde se realizó la llamada.
Véase también: LEAVE PROGRAM, LEAVE SCREEN, LEAVE LIST-PROCESSING.

LEAVE PROGRAM

Definición
Se utiliza para abandonar el programa actual.
Sintaxis:
LEAVE.
Abandona el programa actual y continúa el proceso después de la sentencia CALL TRANSACTION, CALL DIALOG o SUBMIT ... AND RETURN. Esta sentencia termina la ejecución del programa sin ejecutar ninguna sentencia más.
Véase también: LEAVE, LEAVE SCREEN, LEAVE LIST-PROCESSING.

LEAVE SCREEN

Definición
Se utiliza para abandonar el proceso de la pantalla actual.
Sintaxis:
LEAVE SCREEN.
Con esta sentencia abandonamos el proceso del dynpro actual. El siguiente dynpro puede haber sido definido estáticamente en los atributos del dynpro o dinámicamente con la sentencia SET SCREEN. Si el siguiente dynpro que se va a procesar es el 0, el sistema devuelve control al programa que realizó la llamada al dynpro.
Véase también: LEAVE TO SCREEN, SET SCREEN.

LEAVE TO LIST-PROCESSING

Definición
Sentencia utilizada para abandonar el modo de ejecución de listado para volver al modo de diálogo.
Sintaxis:
LEAVE LIST-PROCESSING [AND RETURN TO SCREEN <dynpro>].
No es necesario utilizar esta sentencia cuando en un programa interactivo pulsamos una tecla de función. Cuando se ejecuta la sentencia el sistema devuelve control al proceso PBO del dynpro donde realizamos la llamada.
Si utilizamos la cláusula AND RETURN TO SCREEN <dynpro> el programa irá a la dynpro del programa que le indiquemos <dynpro>.
Véase también: LEAVE PROGRAM, LEAVE SCREEN, LEAVE.

LEAVE TO SCREEN

Definición
Permite abandonar la dynpro actual para pasar a otra dynpro.
Sintaxis:
LEAVE TO SCREEN <dynpro>.
<dynpro> identifica el número del dynpro que se va a procesar. Esta sentencia es una mezcla de la sentencia SET SCREEN y LEAVE SCREEN. Si <dynpro> es 0 se devuelve el control al programa que realizó la llamada al módulo de función.

El ejemplo de la sentencia SET SCREEN nos sirve para ver un ejemplo de la sentencia LEAVE SCREEN.
Véase también: SET SCREEN, LEAVE SCREEN.

LEAVE TO TRANSACTION

Definición
Permite abandonar la dynpro actual para pasar a otra dynpro.
Sintaxis:
LEAVE TO TRANSACTION <transacción> [AND SKIP FIRST SCREEN].
La transacción llamada se especifica en <transacción>.
La cláusula AND SKIP FIRST SCREEN tiene el mismo significado que en la sentencia CALL TRANSACTION.
Véase también: CALL TRANSACTION.

LOAD REPORT

Definición
Esta sentencia proporciona una herramienta para analizar distintas partes de un programa.
Sintaxis:
LOAD REPORT <programa> PART { 'HEAD' | 'TRIG' | 'CONT' | 'DATA' | 'DDNM' |
'DATV' | 'SELC' | 'STOR' | 'LITL' | 'SYMB' | 'LREF' | 'SSCR' | 'BASE' |
'INIT' | 'DATP' | 'TXID' | 'COMP' } INTO <tabla>.
Carga la parte específica en la cláusula PART de la versión generada del programa sobre la tabla interna con el único propósito de análisis. Los posibles valores en la variable SY-SUBRC son los siguientes:

- 0 -> La parte del programa se carga satisfactoriamente.
- 4 -> La parte del programa a cargar no existe.
- 8 -> El programa existe pero no tiene versión generada.

Las posibles partes a cargar de un programa es:

PARTE ...	**Se carga sobre la tabla interna ...**	**Estructura de <tabla>**
HEAD	La cabecera del programa	RHEAD
TRIG	Los bloques de control de evento	RTRIG
CONT	Los bloques de control de proceso	RCONT
DATA	La descripción de los datos estáticos	RDATA
DDNM	El nombre de las estructuras del diccionario de datos utilizadas en el programa.	RDDNM
DATV	La descripción de las variables	RDATA
SELC	La descripción de las variables de selección	RSELC
STOR	Los valores iniciales de los datos globales	Un campo de tipo X
LITL	La tabla de literales	Un campo de tipo X
SYMB	La tabla de símbolos	RSYMB
LREF	La línea de referencia	RLREF
SSCR	La descripción de la pantalla de selección	RSSCR
BASE	La tabla de segmento	RBASE
INIT	Los valores iniciales de los datos locales	Un campo de tipo X
DATP	Las descripciones de los parámetros	RDATA
TXID	El índice de los elementos de texto	RTXID
COMP	La descripción de los componentes internos.	RDATA

Los errores en tiempo de ejecución pueden ser los siguiente:

- LOAD_REPORT_PART_NOT_FOUND -> Se especifico una parte no válida.
- LOAD_ REPORT_PROGRAM_NOT_FOUND -> El programa especificado no existe.
- LOAD_REPORT_TABLE_TOO_SHORT -> La tabla especificada es demasiado pequeño.

SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Véase también: CALL TRANSACTION.

LOCAL

Definición
Declara un objeto local dentro de una subrutina.
Sintaxis:
LOCAL <objeto de dato>.
<objeto de dato> puede ser cualquier objeto de dato, incluidos los *field-symbols* y los parámetros formales de una subrutina. Las modificaciones realizadas sobre el objeto de dato sólo serán efectivas dentro de la subrutina.

LOOP .. ENDLOOP

Definición
Esta sentencia lee una tabla interna línea a línea a través de un bucle de lectura. Permite realizar un bucle en una tabla de pantalla.
Sintaxis:
LOOP AT <tabla> [INTO <área-trabajo>] [FROM <n1>] [TO <n2>]
[WHERE <condición>].
...
ENDLOOP.

- Con la cláusula INTO <área-trabajo> especificamos un área de trabajo, la cual se convierte en el área de destino. En el caso de tablas con cabeceras de línea, esta cláusula es opcional. La tabla interna <tabla> se lee línea a línea sobre <área-trabajo> (si se especifica) o sobre el área de trabajo de la tabla <tabla>. Por cada línea leída el sistema procesa el bloque de sentencias del bucle LOOP .. ENDLOOP. Con la sentencia AT se puede controlar el flujo de bloque de sentencias. Mientras se procesa el bloque de sentencias, la variable SY-TABIX contiene el índice de la actual línea tratada. El bucle termina en cuanto se han leído todas las líneas de la tabla interna. Después de salir del bucle, la variable SY-SUBRC será 0 si al menos una línea ha sido leída, si no, tendrá el valor 4.
- Se puede restringir el número de líneas para leer, con las cláusulas FROM, TO y WHERE. Con la cláusula FROM, en <n1> especificamos el índice desde donde comenzaremos la lectura. Con la cláusula WHERE especificamos cualquier expresión lógica en <condición>. El primer operando de la expresión lógica debe ser un componente de la estructura de la línea. No se puede utilizar la cláusula WHERE si en el bloque de proceso se utiliza la sentencia AT. Con las cláusulas FROM y TO limitamos las lecturas de la tabla interna, mientras que con la cláusula WHERE leemos todas las entradas para poder aplicar la expresión lógica. Por razones de optimización, y siempre que podamos, se debe utilizar las cláusulas FROM y TO.

LOOP [WITH CONTROL <tabla-control>]
...
ENDLOOP.
Esta variante permite realizar un bucle sobre una tabla de pantalla. La tabla de pantalla puede ser una tabla de control, y si es así, se debe utilizar la cláusula WITH CONTROL.
LOOP AT <tabla> CURSOR <cursor> [WITH CONTROL <tabla-control>]
[FROM <línea1>] [TO <línea2>].

...
ENDLOOP.
Esta variante nos permite rellenar los campos de la tabla de pantalla a través de campos de una tabla interna. El significado de las cláusulas es el siguiente:

- CURSOR define el cursor de la tabla interna.
- WITH CONTROL define una tabla de control.
- FROM define la línea desde donde realizamos el traspaso de datos.
- TO define la línea límite de asociación con la tabla de la pantalla.

Ejemplo 1:

```
DATA: BEGIN OF T OCCURS 100,
       BAREA(5), BLNCE(5),
      END OF T.
LOOP AT T FROM 7 TO 8.
  WRITE: / T-BAREA, T-BLNCE.
ENDLOOP.
```

Ejemplo 2:

```
DATA: BEGIN OF T OCCURS 100,
       BAREA(5), BLNCE(5),
      END OF T.
DATA CMP_BAREA LIKE T-BAREA.
CMP_BAREA = '01'.
LOOP AT T WHERE BAREA = CMP_BAREA.
  WRITE: / T-BAREA, T-BLNCE.
ENDLOOP.
```

Véase también: APPEND, INSERT, COLLECT, MODIFY, DELETE, SORT, READ TABLE.

LOOP AT SCREEN .. ENDLOOP

Definición

Esta sentencia nos permite acceder a los atributos de todos los campos de un dynpro.

Sintaxis:

```
LOOP AT SCREEN.
...
ENDLOOP.
```

Todos los campos de un dynpro se almacenan en la tabla del sistema SCREEN con sus atributos. La sentencia LOOP AT SCREEN sitúa la información de cada campo sobre la línea de cabecera de la tabla del sistema. Esta sentencia se puede utilizar en combinación con la sentencia MODIFY SCREEN para modificar los atributos de cualquier campo de la dynpro, aunque esta acción solo se puede realizar en el PBO (Porcess Before Output) del dynpro. Si esta sentencia se utiliza en combinación de una *step-loop* (bucle sobre una tabla de pantalla) sólo veremos los atributos de la línea de la tabla de pantalla que se está tratando en ese momento. Los campos de un *step-loop* de pantalla no deben ser modificados fuera del bucle *step-loop*.

Los campos de la tabla *screen* son los siguientes:

Campo	Tipo	lg.	Significado
NAME	C	30	Nombre del campo.
GROUP1	C	3	Grupo de evaluación de modificación 1.
GROUP2	C	3	Grupo de evaluación de modificación 2.
GROUP3	C	3	Grupo de evaluación de modificación 3.
GROUP4	C	3	Grupo de evaluación de modificación 4.
REQUIRED	C	1	Campo obligatorio.
INPUT	C	1	Campo de entrada.
OUTPUT	C	1	Campo de salida.
INTENSIFIED	C	1	Campo con intensidad.

INVISIBLE	C	1	Campo invisible.
LENGTH	X	1	Longitud del campo.
ACTIVE	C	1	Campo activo.
DISPLAY_3D	C	1	Atributo formato 3-D
VALUE_HELP	C	1	Atributo de campo de ayuda.

Los atributos se activan con el valor '1' y se desactivan con el valor '0'.

Ejemplo:

```
CONSTANTS OFF VALUE '0'.
LOOP AT SCREEN.
  SCREEN-INPUT = OFF.
  MODIFY SCREEN.
ENDLOOP.
```

Con esta bucle indicamos que todos los campos solo son de salida.

Véase también: MODIFY SCREEN

MESSAGE

Definición

El sistema permite reaccionar a acciones incorrectas por parte del usuario mostrando un mensaje, el cual afectará al flujo del programa.

Sintaxis:

REPORT <programa> ... MESSAGE-ID <clase>.

Los mensajes se almacenan en la tabla T100. Los mensajes están ordenados por el lenguaje, la identificación de la clase de mensaje (2 caracteres) y por el número de mensaje (3 caracteres). En un programa se puede utilizar varias clases de mensaje, pero sólo una de ellas podrá ser definida estáticamente, que se define en la sentencia REPORT.

En la sentencia REPORT existe la cláusula MESSAGE-ID que nos permite asociar al programa, de manera estática, una clase de mensaje.

MESSAGE <tipo><num> [WITH <c1> <c2> <c3> <c4>].

<tipo> define el tipo de mensaje. <num> indentifica el número de mensaje. En esta sentencia se mostraran los mensajes de la clase de mensaje definidos en la sentencia REPORT....

- La cláusula WITH permite añadir variables en el mensaje. Existe la limitación de cuatro variables por mensaje. En el mensaje alamacenado en la tabla T100, las variables se definen con un *ampersand* (&) acompañado de un número del 1 al 4.

MESSAGE ID <clase> TYPE <tipo> NUMBER <num> [WITH <c1> <c2> <c3> <c4>].

Con esta variante especificamos la clase del mensaje de forma dinámica. <clase> determina la clase de mensaje.

- La cláusula TYPE se utiliza para determinar el tipo de mensaje. La cláusula NUMBER se utiliza para especificar el número de mensaje.
- La cláusula WITH tiene el mismo significado que en la anterior variante.

Un mensaje puede tener cinco diferentes tipos:

- Tipo A (Abend) -> El sistema muestra este tipo de mensaje en una ventana de diálogo. Después de que el usuario confirme el mensaje pulsando la tecla INTRO, el sistema abandona la transacción donde este.
- Tipo E (Error) o tipo W(Warning) -> El sistema muestra el mensaje en la línea de status. Después de que el usuario pulsa INTRO el sistema actúa en función del punto de proceso donde nos encontremos. Si nos encontramos en la creación del listado básico, SAP termina el programa. Si nos encontramos en la creación de un listado secundario, el sistema termina el actual bloque de proceso y presenta el listado del nivel anterior.
- Tipo I (Información) -> El sistema muestra el mensaje en una ventana de diálogo. Después de que el usuario pulse INTRO, el sistema continúa con el proceso.
- Tipo S (Success) -> El sistema muestra el mensaje en la línea de status del listado creado.

Ejemplo 1:

MESSAGE ID 'XX' TYPE 'E' NUMBER '001'
WITH 'Text'.

Ejemplo 2:

MESSAGE E010 WITH 'Example' SY-UNAME.

MODIFY

Definición

Se utiliza para sustituir líneas de una tabla interna.

Sintaxis:

MODIFY <tabla> [FROM <área-trabajo>] [INDEX <índice>].

El área de trabajo <área-trabajo> sustituye a la línea de la tabla interna.

- En el caso de tablas internas con líneas de cabecera, la cláusula FROM es opcional.
- Si utilizamos la cláusula INDEX, la línea que hay que reemplazar será la existente con el índice <índice>. Si el sistema puede realizar la modificación, la variable del sistema SY-SUBRC es 0. Si la tabla interna tiene menos líneas que las indicadas en <índice>, la modificación no se realiza y SY-SUBRC vale 4. Si no utilizamos la cláusula INDEX, la sentencia solo puede ser procesada en un bucle LOOP .. ENDLOOP. En este caso la línea que se modificara será la que se este tratando.

MODIFY { <tabla> | (<tabla>) }[CLIENT SPECIFIED] [FROM <área>].

Sentencia utilizada para modificar o insertar registros de una tabla de la base de datos. Con esta sentencia tenemos dos posibilidades. Si la tabla de base de datos no tiene ninguna línea con la misma clave primera se inserta. En caso contrario se modifica. Por cuestiones de rendimiento se debe limitar su uso al máximo.

Con <tabla> se especifica el nombre de la tabla de forma estática. Para especificar la tabla de forma dinámica se utiliza la opción (<tabla>).

- Con la especificación dinámica, la cláusula FROM es obligatoria.

MODIFY { <tabla> | (<tabla>) }[CLIENT SPECIFIED] [FROM TABLE <área>].

Esta sentencia modifica o inserta el contenido de área de trabajo <área> sobre la tabla de la base de datos.

Las líneas de la tabla interna <tabla-interna> modifican las líneas de la tabla de la base de datos si la clave primera existe. Para el resto de entradas de la tabla interna, se añaden a la tabla. SY-SUBRC es siempre 0. SY-DBCNT toma el valor del número de entradas de la tabla interna. Cuando se añaden entradas, la sentencia funciona como la sentencia INSERT y cuando se modifican entradas, las sentencias funciona como la sentencia MODIFY.

Véase también: COLLECT, APPEND, INSERT.

MODIFY CURRENT LINE

Definición

Esta sentencia modifica la última línea leída.

Sintaxis:

MODIFY CURRENT LINE [LINE FORMAT <opción-1> ... <opción-n>]
[LINE FORMAT <opción-1> ... <opción-n>]
[FIELD VALUE <c11> [FROM <c12>] ... <cn1> [FROM <cn2>]] [FIELD FORMAT <c1> <opciones-1> ... <cn> [<opciones-n>].

Si no utilizamos ninguna opción, la última línea leída se modifica con el contenido actual del campo SY-LISEL. El área HIDE de la línea se sustituye por los valores actuales de los campo correspondientes, sin embargo esto no influye en los valores mostrados. Si el sistema puede modificar la línea, el campo del sistema SY-SUBRC toma el valor 0. En caso contrario toma un valor distinto de 0.

Ejemplo 1:

```
DATA I TYPE I VALUE 2.
WRITE: / 'Intensified'     INTENSIFIED,
       'Input'          INPUT,
       'color 1'        COLOR 1,
       'intensified off' INTENSIFIED OFF.
* Line selection
AT LINE-SELECTION.
  MODIFY CURRENT LINE
    LINE FORMAT INVERSE
              INPUT OFF
              COLOR = I.
```

Ejemplo 2:

```
DATA: FLAG     VALUE 'X',
      TEXT(20) VALUE 'Australia',
      I TYPE I VALUE 7.
FORMAT INTENSIFIED OFF.
WRITE: / FLAG AS CHECKBOX, TEXT COLOR COL_NEGATIVE.
AT LINE-SELECTION.
  MODIFY CURRENT LINE
    LINE  FORMAT INTENSIFIED
    FIELD VALUE  FLAG FROM SPACE
    FIELD FORMAT FLAG INPUT OFF
              TEXT COLOR = I.
```

Véase también: MODIFY LINE.

MODIFY LINE

Definición

Esta sentencia modifica cualquier línea de un listado.

Sintaxis:

MODIFY LINE <n> [INDEX <índice> | OF CURRENT PAGE | OF PAGE <página>]
[LINE FORMAT <opción-1> ... <opción-n>]
[LINE FORMAT <opción-1> ... <opción-n>]
[FIELD VALUE <c11> [FROM <c12>] ... <cn1> [FROM <cn2>]]
[FIELD FORMAT <c1> <opciones-1> ... <cn> [<opciones-n>].

Sin las cláusulas opcionales de la primera línea de la sentencia, el sistema modifica la línea <n> del listado en el cual ocurra el evento (índice SY-LISTI). Con las cláusula de la primera línea se puede modificar lo siguiente:

- INDEX -> Se modifica la línea <n> del listado de nivel definido en <índice>.
- OF CURRENT PAGE -> Se utiliza para modificar la línea <n> de la página <página>.
- OF PAGE -> Se utiliza para modificar la línea <n> de la página <página>.

El significado del resto de cláusulas es el siguiente:

- LINE FORMAT -> Actualiza el formato de salida de la línea a modificar de acuerdo con las especificaciones realizadas en <opción-1> ... <opción-n>.
- FIELD VALUE -> Modifica ciertos campos de la línea que modificamos. El campo <c11> se modifica con el valor actual del campo <c1> o del campo <c12> si se especifica la cláusula FROM. Si el campo especificado se utiliza más de una vez en la línea que se quiere modificar, el sistema modifica, sólo el primer campo. Si en el campo no aparece en la línea que hay que modificar sólo el primer campo. Si en el campo no aparece en la línea que hay que modificar, la opción se ignora. El sistema modifica el contenido del campo de la línea que se debe modificar, independientemente del contenido actual del campo del sistema

SY-LISEL. Además de modificar un campo con esta cláusula se puede modificar directamente la variable del sistema SY-LISEL.

- FIELD FORMAT -> Modifica el formato de salida del campo especificado en <c1>, con las opciones <opciones-1>. En opciones <opciones-1> se puede especificar una o más de una opción de la cláusula FORMAT. Esta cláusula sobreescribe las opciones utilizadas con la cláusula LINE FORMAT. Si el campo especificado se utiliza más de una vez en la línea que se va a modificar, el sistema modifica sólo el primer campo. Si el campo no aparece en la línea que se va a modificar, la opción se ignora.

Véase también: MODIFY CURRENT LINE.

MODIFY SCREEN

Definición
Modifica ciertos atributos de campos de un dynpro. Sentencia utilizada para modificar los atributos de los campos de pantalla.
Sintaxis:
MODIFY SCREEN.
Las modificaciones realizadas en los atributos de campos de un dynpro se deben realizar en un bucle LOOP AT SCREEN ... ENDLOOP. En el sistema existe una tabla denominada SCREEN donde se guardan los atributos de los campos de un dynpro. Dicha tabla, y dentro de un bucle, se puede actualizar los valores de la tabla para luego modificarla con esta sentencia. Esta sentencia solo se puede utilizar en el proceso PBO de un dynpro o de la pantalla de selección de un dynpro.
Esta sentencia solo tiene sentido en combinación con la sentencia LOOP AT SCREEN.
Véase también: LOOP AT SCREEN.

MODULE

Definición
Se utiliza para llamar a un módulo. El módulo debe encontrarse en el *modulpool* de la transacción.
Sintaxis:
MODULE <módulo> [AT EXIT-COMMAND].
La cláusula AT EXIT-COMMAND tiene un efecto especial. El módulo se ejecuta si el usuario activa una función de tipo *E* (exit).
Véase también: LOOP AT SCREEN.

MODULE .. ENDMODULE

Definición
Declaración de un módulo.
Sintaxis:
MODULE <módulo> [INPUT | OUTPUT].
...
ENDMODULE.
Al bloque de proceso comprendido entre MODULE y ENDMODULE se conoce como *módulo*. Desde el proceso lógico de un dynpro llamamos al módulo con la sentencia del proceso lógico MODULE <módulo>.
No se debe confundir la sentencia ABAP/4 MODULE (que siempre termina con la sentencia ENDMODULE), utilizada en un programa o modulpool, con la sentencia del proceso lógico MODULE.

- Los módulos que serán llamados desde el bloque PBO (Process Before Output), es decir, antes del procesamiento de la pantalla deben llevar la cláusula OUT-PUT.
- La cláusula INPUT define un módulo que será procesado en el bloque PAI (Process After Input). La cláusula INPUT es el valor po defecto de la sentencia y no es necesario incluirla. La cláusula INPUT y OUTPUT no se pueden especificar a la vez. Un mensaje de tipo E (error) cancela el proceso del módulo. Un mensaje de tipo W (warning) hace repetir el módulo actual (o la de módulos, sentencia CHAIN). Si presionamos INTRO después del mensaje W el proceso continúa después de la sentencia MESSAGE.

MODULE <módulo> [INPUT \ OUTPUT].
...
ENDMODULE.
Con la cláusula INPUT y OUTPUT definimos si el módulo pertenece al proceso PAI o PBO respectivamente. Si el módulo pertenece al proceso PAI, la cláusula INPUT es obligatoria.

MOVE

Definición
Asigna un literal o el contenido de un campo fuente sobre un campo objeto se utiliza la sentencia MOVE.
Sintaxis:
MOVE <c1> [+ <o1>] [(<l1>)] TO <c2> [+ <o2>] [(<l2>)].
Esta sentencia transporta el contenido del campo <c1>, el cual puede se cualquier objeto de dato, sobre el campo <c2>, el cual debe ser una variable. <c2> no puede ser una constante o un literal. El contenido de <c1> permanece sin cambios.
MOVE <campo1> TO <campo2> PERCENTAGE <p> [RIGHT].
Copia el porcentaje <p> izquierdo del campo <campo1> sobre <campo2>. Sin indicamos la cláusula RIGHT será la parte derecha del campo <campo1> la que será movida al campo <campo2>.
MOVE <campo1> TO <campo2> PERCENTAGE <p> [RIGHT].
<tabla1> y <tabla2> son las áreas de trabajo de las tablas internas. Si una tabla interna tiene área de trabajo, área y tabla tienen el mismo nombre, para poder distinguirlos se utilizan los corchetes ([ ]). Con los corchetes será el contenido de la tabla interna el que se copiará, en lugar del área de trabajo. La sentencia MOVE copia el contenido de la primera tabla interna (o área de trabajo) especificada sobre la segunda tabla interna (o área de trabajo). Las tablas utilizadas pueden tener como componentes a su vez tablas internas. El contenido de <tabla2> se sobrescribe.
Sólo se pueden copiar tablas internas sobre otras tablas internas, no sobre field-strings o campos elementales. Para poder copiar tablas internas, las líneas de la tablas deben ser convertibles (aunque no sean compatibles).
Véase también: MOVE-CORRESPONDING.

MOVE-CORRESPONDING

Definición
Se utiliza para copiar los datos de un componente field-string sobre otro componente field-string.
Sintaxis:
MOVE-CORRESPONDING <string1> TO <string2>.
Asigna el contenido de los componentes del registro <string1> sobre los componentes del registro <string2> que se llamen igual. Por cada para de nombres coincidentes el sistema ejecuta una sentencia MOVE, además realiza todos los tipo de conversión necesarios de forma separada.
Ejemplo:

```
DATA: BEGIN OF INT_TABLE OCCURS 10,
      WORD(10),
      NUMBER TYPE I,
```

```
      INDEX  LIKE SY-INDEX,
    END   OF INT_TABLE,
    BEGIN OF RECORD,
      NAME(10) VALUE 'not WORD',
      NUMBER TYPE I,
      INDEX(20),
    END   OF RECORD.
...
MOVE-CORRESPONDING INT_TABLE TO RECORD.
```
Véase también: MOVE.

MULTIPLY

Definición
Multiplica el contenido de campos.
Sintaxis:
MULTIPLY <m> BY <n>.
Multiplica el contenido de los campos <m> y <n>, dejando el resultado sobre el campo <m>.
Ejemplo:
```
DATA: DAYS_PER_YEAR    TYPE P VALUE 365,
    HOURS_PER_DAY    TYPE F VALUE '24.0',
    MINUTES_PER_YEAR TYPE I VALUE 1.
MULTIPLY MINUTES_PER_YEAR BY DAYS_PER_YEAR.
MULTIPLY MINUTES_PER_YEAR BY HOURS_PER_DAY.
MULTIPLY MINUTES_PER_YEAR BY 60.
MINUTES_PER_YEAR contiene 525600.
```
Véase también: MULTIPLY-CORRESPONDING.

MULTIPLY-CORRESPONDING

Definición
Multiplica el contenido de los componentes del registro <m> y del registro <n>.
Sintaxis:
MULTIPLY-CORRESPONDING <m> BY <n>.
Multiplica el contenido de los campos <m> y <n>, para aquellos componentes que se llamen igual. El resultado actualiza los componentes del registro <m>.
Ejemplo:
```
DATA: BEGIN OF MONEY,
      VALUE_IN(20) VALUE 'Marcos germanos'.
      USA TYPE I VALUE 100,
      FRG TYPE I VALUE 200,
      AUT TYPE I VALUE 300,
    END   OF MONEY,
    BEGIN OF CHANGE,
      DESCRIPTION(30)
         VALUE 'DM a moneda del país'.
      USA TYPE F VALUE '0.6667',
      FRG TYPE F VALUE '1.0',
      AUT TYPE F VALUE '7.0',
    END   OF CHANGE.
MULTIPLY-CORRESPONDING MONEY BY CHANGE.
MONEY-VALUE_IN = 'Moneda del país'.
```
Véase también: MULTIPLY.

NEW-LINE

Definición

Para provocar un salto de línea se puede utilizar la cláusula AT o "/" de la sentencia WRITE, la sentencia ULINE, o la sentencia NEW-LINE.

Sintaxis:

NEW-LINE [NO-SCROLLING | SCROLLING].

Esta sentencia posiciona la salida en una nueva línea, SY-COLNO toma el valor 1 y SY-LINNO incrementa su valor en 1. Esta sentencia, a diferencia con la sentencia SKIP, no genera líneas en blanco. Si después de esta sentencia no hay más sentencias de escritura, ésta no tiene ningún efecto.

Con la opción NO-SCROLLING la siguiente línea no se desplaza con el scrolling horizontal.

Con la opción SCROLLING desactivada. La otra opción tiene efecto si anteriormente se ha utilizado la opción NO-SCROLLING y no se ha realizado ninguna salida en la nueva línea.

NEW-PAGE

Definición

Si en un informe escribimos más líneas en el dispositivo de salida que las definidas en la cláusula LINE-COUNT de la sentencia REPORT, el sistema crea automáticamente una nueva página. Aparte de esta creación automática de páginas, se puede forzar al sistema a crear una nueva página con las sentencias NEW-PAGE y RESERVE.

Sintaxis:

NEW-PAGE.

Esta sentencia termina con las líneas de salida en la página actual. Cualquier nueva línea de salida saldrá en una nueva página. La nueva página será creada si a partir de la sentencia NEW-PAGE aparecen sentencias de escritura en el dispositivo de salida (WRITE, SKIP, ...). Destacar que el evento END-OF-PAGE no se activa después de la sentencia NEW-PAGE.

Véase también: RESERVE.

ON CHANGE OF .. ENDON

Definición

Ejecuta un bloque de proceso si cambia el valor de un campo.

Sintaxis:

ON CHANGE OF <campo1> [OR <campo2>]
...
ENDON.

El bloque de proceso marcado por la sentencia ON CHANGE OF y ENDON se ejecuta si se produce algún cambio en el campo <campo1>.

Con la cláusula OR se puede especificar más campo para considerar posibles cambios.

Ejemplo:

```
* Base de datos lógica F1S
TABLES: SPFLI, SFLIGHT, SBOOK.
GET SBOOK.
  ON CHANGE OF SPFLI-CARRID   OR
               SPFLI-CONNID   OR
               SFLIGHT-FLDATE.
    ULINE.
    WRITE: /5 SPFLI-CARRID, SPFLI-CONNID,
           5 SFLIGHT-FLDATE, SPFLI-FLTIME,
           5 SFLIGHT-SEATSMAX, SFLIGHT-SEATSOCC.
  ENDON.
  WRITE: / SBOOK-CUSTOMID.
```

Véase también: RESERVE.

OPEN CURSOR

Definición
Se puede leer líneas de una tabla de la base de datos utilizando un cursor. Antes de poder utilizar un cursor (con la sentencia FETCH) es necesario abrirlo y, para ello, contamos con la sentencia OPEN CURSOR.
Sintaxis:
OPEN CURSOR [WITH HOLD] <cursor> FOR SELECT ... [WHERE <condición>].
La sentencia lleva implícita una sentencia SELECT que nos sirve para seleccionar las líneas que serán recorridas por el cursor. El cursor <cursor> debe estar definido con la sentencia DATA y ser del tipo CURSOR. Si utilizamos la opción WITH HOLD, el cursor permanece abierto después de un COMMIT de la base de datos con una sentencia *Native SQL*.
En la cláusula SELECT no se puede utilizar la opción SINGLE y funciones de agregación. El resto de opciones están permitidas.
Véase también: CLOSE CURSOR, GET CURSOR.

OPEN DATASET

Definición
Abre un fichero en el servidor de aplicación.
Sintaxis:
OPEN DATASET <fichero> [MESSAGE <mensaje>]
[FOR INPUT | FOR OUPUT | FOR APPENDING]
[IN BINARY | IN TEXT MODE] [AT POSITION <posición>].
Abre el fichero <fichero>. Si no se especifica la opción para el modo de apertura, el fichero se abre para lectura en modo binario. Si el sistema puede abrir el fichero, SY-SUBRC vale 0. En caso contrario vale 8. <fichero> puede ser un literal o un campo que contenga el nombre del fichero. Si no se especifica el camino de acceso del fichero, el sistema abre el fichero en el directorio donde el sistema SAP se esté ejecutnado, en el servidor de aplicación. Para abrir un fichero, el usuario bajo el que se esté ejecutando el sistema SAP debe tener las apropiadas autorizaciones del sistema operativo.

- MESSAGE -> Sobre el campo <mensaje> recibimos mensajes del sistema operativo de cómo ha ido la operación de apertura del fichero. <mensaje> y la variable del sistema SY-SUBRC nos pueden servir para controlar los posibles errores que puedan surgir.
- FOR INPUT -> El fichero se abre para lectura. Si el fichero no existe SY-SUBRC vale 8. Si el fichero ya se ecnuentra abierto (para cualquier opción de apertura: lectura, escritura, añadir), el sistema inicia el posicionamiento sobre el fichero al principio de éste. Con esto el sistema no da error, pero se recomienda utilizar la sentencia CLOSE DATASET antes de utilizar de nuevo un fichero.
- FOR OUPUT -> Abre un fichero para escritura. Si el fichero no existe se crea, si existe y además esta abierto en el programa, el posicionamiento sobre el fichero se inicia a la primera posición de éste. Si el fichero no se puede abrir, SY-SUBRC valdrá 8.
- FOR APPENDING -> El fichero se abre para escritura para añadir registros al final del fichero. Si el fichero no existe se crea. Si el fichero existe y está cerrado, el sistema abre el fichero y posiciona el cursor del fichero al final de éste. Si el fichero existe y está abierto en el programa, la posición del cursor del fichero se posiciona al final de éste. La variable del sistema siempre toma el valor 0. Es aconsejable cerrar el fichero antes de abrirlo en el mismo programa.
- IN BINARY MODE -> El fichero abierto, tanto para lectura como para escritura, los datos se transmiten byte a byte. El contenido del fichero no es interpretado durante la transmisión. Cuando escribimos el contenido de un campo sobre un fichero, el sistema transmite todos

los bytes del campo fuente. Cuando leemos datos de un fichero de un campo, el número de bytes transferidos depende de la longitud del campo receptor. Cuando después de la lectura, referenciamos el campo destino en otra sentencia, el sistema interpreta el contenido del campo en función de su tipo.

- IN TEXT MODE -> El fichero se abre en modo texto. Los datos leídos del fichero o los datos escritos sobre el fichero se realizan de línea en línea. Por cada sentencia TRANSFER se escriben todos los bytes pertenecientes a una línea y se pone una marca de fin de línea. Si el campo destino es más pequeño que la línea del fichero, los datos se truncan. Si el campo destino es mayor que la línea leída, se rellena con caracteres en blanco. Se debe utilizar el modo texto si queremos escribir string sobre el fichero, o si conocemos el formato de la línea del fichero.
- AT POSITION -> El fichero se abre en una posición determinada. El cursor del fichero se posiciona en la posición <posición> es contada en bytes desde el principio del fichero. No se puede especificar una posición anterior al principio del fichero.

Véase también: CLOSE DATASET, READ DATASET.

OPERADOR DE ASIGNACION "="

Definición
Con este operador de asignación conseguimos lo mismo que con la sentencia MOVE.
Sintaxis:
<tabla1> = <tabla2>. <tabla1> = <tabla2>[].
<tabla1> = ... = <tablan>. <tabla1> = ... = <tablan>[].
Todo lo comentado para la sentencia MOVE es válido para este operador de asignación.
Véase también: MOVE, MOVE TO.

OVERLAY

Definición
Se utiliza para sobrescribir un campo alfanumérico con otro campo.
Sintaxis:
OVERLAY <campo1> WITH <campo2> [ONLY <string>].
Esta sentencia sobrescribe todas las posiciones del campo <campo1> que contengan las letras de <string> con el contenido del campo <campo2>. <campo2> permanece sin cambios.
Si omitimos la cláusula ONLY, sólo las posiciones con espacios en blanco serán sobrescritas. Si al menos una posición del campo <campo1> es sustituida, SY-SUBRC será 0. En caso contrario será 4. Si <campo1> es superior a la longitud que <campo2>, sólo los caracteres correspondientes a la longitud de <campo2> podrán ser actualizados en <campo1>.
Ejemplo 1:

```
DATA: WORK(20) VALUE 'Th t h s ch ng d.',
      HELP(20) VALUE 'Grab  a   pattern'.
OVERLAY WORK WITH HELP.
```

La variable WORK vale 'That has changed.'
Ejemplo 2:

```
DATA: ONE(16), TWO(16).
ONE = '----****++++....'.
TWO = '-*+.-*+.-*+.-*+.'.
OVERLAY ONE WITH TWO ONLY '.'.
OVERLAY TWO WITH ONE ONLY '.+'.
OVERLAY ONE WITH TWO ONLY '+*'.
```

El campo ONE contiene '-----***-*++-*+.' y el campo TWO contiene '-*---***-*++-*+.' .
Véase también: REPLACE, SHIFT, SEARCH, TRANSLATE.

PACK

Definición
Empaqueta el contenido de un campo sobre otro.
Sintaxis:
PACK <campo1> TO <campo2>.
Empaqueta el campo <campo1> en el campo <campo2>. La operación contraria ser haría con la sentencia UNPACK.
Ejemplo:

```
DATA C_FIELD(4) TYPE C VALUE '0103',
    P_FIELD(2) TYPE P.
PACK C_FIELD TO P_FIELD.
C_FIELD: C'0103' --> P_FIELD: P'103C'
```

Véase también: UNPACK.

PARAMETERS

Definición
Si queremos dar la posibilidad al usuario de introducir valores en variables en la pantalla de selección, debemos definir dichas variables con la sentencia PARAMETERS. La declaración de variables con la sentencia PARAMETERS es muy parecida a la sentencia DATA.
Sintaxis:

```
PARAMETERS <parámetro> [ (<longitud)>].
[ TYPE <tipo> | LIKE <campo> ]
[ DECIMALS <decimales> ] [ DEFAULT <valor> ]
[ MEMORY ID <memoria>] [ MODIF ID <grupo> ]
[ MATCHCODE OBJECT <objeto> ] [ AS MATCHCODE STRUCTURE]
[ NO-DISPLAY ] [ LOWER CASE ] [ OBLIGATORY ] [ FOR TABLE <tabla> ]
[ AS CHECKBOX ] [ RADIOBUTTON GROUP <grupo> ]
[ VALUE-REQUEST ] [ HELP REQUEST ].
```

Con esta sentencia definimos el parámetro <parámetro>. Las cláusulas <longitud>, <tipo> y <decimales> sin iguales a las de la sentencia DATA.
Cuando el usuario arranque el programa aparece una pantalla de selección con el parámetro <parámetro>, con longitud la especificada en <longitud>, o la longitud por defecto del campo si la cláusula no se define. Por defecto la descripción que aparece a la izquierda del campo es el nombre del campo. El objeto parcial de programa *Elementos de texto*, la opción *Textos de selección* nos permite asociar al parámetro un texto significativo. Otra posibilidad es la utilización de la sentencia SELECTION-SCREEN como veremos en el apartado correspondiente a esta sentencia.
Se puede utilizar los parámetros de selección, por ejemplo, para que el usuario limite los valores desde y hasta para una posterior lectura de la base de datos. Si las condiciones de selección son más complejas que indicar un valor desde y hasta, es preferible utilizar la sentencia SELECT-OPTIONS.

- DEFAULT -> <valor> puede ser un literal o el nombre de un campo. Si especificamos el nombre de un campo el valor por defecto en el parámetro será el valor del campo en el momento de presentar la pantalla de selección. Recordemos que el evento INITIALIZATION se ejecuta antes de la presentación de la pantalla de selección, en él se puede actualizar el valor del campo especificado. El usuario puede modificar el valor que aparece por defecto.
- NO-DISPLAY -> El parámetro no aparece en la pantalla de selección. Se le puede dar un valor con la cláusula DEFAULT, en el evento INITILIZATION, o con una llamada al programa con la sentencia SUBMIT. Si queremos mostrar el parámetro bajo ciertas circunstancias no se debe utilizar la cláusula NO-DISPLAY ya que con ella no se podrá hacer visible el parámetro. Se debe definir visible y si no queremos que aparezca el campo utilizamos la sentencia MODIFY SCREEN.

- LOWER-CASE -> El sistema no convierte el valor del parámetro a mayúsculas, es decir, respeta la entrada del usuario, sea con caracteres en mayúsculas y/o minúsculas. Si definimos el tipo de parámetro con la opción LIKE y referenciamos un campo del diccionario de datos, la cláusula LOWER CASE no se puede utilizar.
- OBLIGATORY -> Obliga a introducir un valor al usuario en el parámetro. Para significar el hecho de la obligatoriedad, el sistema presenta el campo con el signo de interrogación (?).
- AS CHECKBOX -> Un checkbox es un campo de entrada de longitud 1 sobre el que se puede activar o desactivar un *vise*. Para definirlo utiliza la cláusula AS CHECKBOX. El valor interno del parámetro será "X" o " " si está activo o desactivo respectivamente. El usuario para activar o desactivar el parámetro debe hacer un click sobre el parámetro. Si utilizamos la opción LIKE en la sentencia PARAMETERS haciendo referencia a un campo del diccionario de tipo C, longitud 1 y valores permitidos "X" y " " (a través del dominio), el parámetro aparece automáticamente como checkbox.
- RADIOBUTTON -> Es un conjunto de campo de entrada de longitud 1 sobre el que se puede activar o desactivar uno y sólo uno de ellos. Para definir un grupo de campos con el formato de radiobutton se utiliza la cláusula RADIOBUTTON GROUP. Cada parámetro definido como radiobutton es de tipo C y longitud 1, y se asigna al grupo <grupo>. La máxima longitud del campo <grupo> es de 4. La cláusula LIKE se puede utilizar pero debe referenciar un campo de tipo C y longitud 1. Se deben asignar al menos dos parámetros por cada grupo. Sólo un parámetro por grupo podrá tener un valor por defecto con la cláusula DEFAULT, y este valor sólo podrá ser "X". Cuando el usuario hace un click sobre el radiobutton, el sistema activa dicho parámetro (valor interno: "X") y desactiva (valor interno: " ") el parámetro del grupo que estuviera activo. Siempre un parámetro del grupo está activo. Si ningún parámetro se activa por definición, o en el evento INITIALIZATION, el sistema activa el primer parámetro del grupo.
- MEMORY ID -> El sistema presentará en el parámetro de la pantalla el valor almacenado en un parámetro de memoria SPA/GPA. <memoria> identifica el nombre del parámetro y debe tener hasta tres caracteres de longitud.
- MODIF ID -> Se utiliza para asignar un parámetro a un grupo de modificación. Ese grupo de modificación podrá ser utilizada por la sentencia MODIFY SCREEN para modificar los atributos de la pantalla de selección. <grupo> es el nombre del grupo de modificación y debe ser un string de tres caracteres sin las comillas ('). <grupo> se asigna a la columna SCREEN-GROUP1 de la tabla interna SCREEN. En el evento AT SELECTION-SCREEN se puede manipular la pantalla de selección gracias a las sentencias LOOP AT SCREEN y MODIFY SCREEN.
- FOR TABLE y AS MATCHCODE STRUCTURE -> Asignamos el parámetro a la tabla de la base de datos <tabla>. Esto sólo tiene sentido en programas de acceso a bases de datos lógicas. A diferencia de un campo de un programa de diálogo (transacción), el istema no valida si el valor introducido existe en el matchcode.
- Las opciones VALUE-REQUEST y HELP-REQUEST -> Sólo se pueden ser utilizadas en programas de definición de bases de datos.

PERFORM

Definición

Nos permite realizar una llamada a una subrutina. Se puede llamar a subrutinas que se encuentran en el mismo programa donde se encuentra la llamada (subrutinas internas) o en otros programas (subrutinas externas). Se puede especificar el nombre de la subrutina estática o dinámicamente en tiempo de ejecución. Se puede realizar llamadas a subrutinas desde un subrutina, incluso es posible que una subrutina se llame así misma (llamada recursiva).

Sintaxis:

PERFORM <subrutina> [TABLES <parámetros_actuales>]
[USING <parámetro_actuales>]

[CHANGING <parámetro_actuales>].

Esta variante primera llama a la subrutina interna <subrutina>. En la cláusula <parámetros> se especifican los parámetros que hay que pasar a la subrutina (parámetros actuales). La subrutina puede acceder a todos los objetos de datos definidos en el programa, por lo tanto, el uso de parámetros es opcional. Se suelen utilizar por claridad en la codificación y por la posibilidad de ser variables.

PERFORM <subrutina>(<programa>) [TABLES <parámetros_actuales>]
[USING <parámetro_actuales>]
[CHANGING <parámetro_actuales>]
[IF FOUND].

Esta variante permite realizar llamadas a subrutinas que se encuentran en otro programa. La única forma de pasar datos es a través de los parámetros o utilizando un área común de memoria (COMMON PARTS).

- Si utilizamos la cláusula IF FOUND, el programa continuará sin producir ningún error si la subrutina no existe.

Cada vez que llamamos a una subrutina externa, el programa que la contiene se carga en memoria. Este hecho hay que tenerlo muy en cuenta a la hora de diseñar una aplicación para evitar cargas excesivas de información.

PERFORM <subrutina>(<programa>) IN PROGRAM <programa>
[TABLES <parámetros_actuales>]
[USING <parámetro_actuales>]
[CHANGING <parámetro_actuales>]
[IF FOUND].

Aquí se permite realizar una llamada a una subrutina que especificamos en tiempo de ejecución. Con la cláusula IN PROGRAM se puede determinar que la subrutina es externa, también de forma dinámica.

PERFORM <índice> OF <subrutina-1> <subrutina-2> ... <subrutina-n>.

Esta variante nos permite llamar a una subrutina en función del valor de un campo índice y la posición que guarda la subrutina en la sentencia. La subrutinas sólo pueden ser internas.

El concepto de parámetro formal corresponde con la definición del parámetro en la subrutina (sentencia FORM). El concepto de parámetro actual corresponde con el parámetro utilizado en la llamada a la subrutina (sentencia PERFORM). Un parámetro (formal o actual) puede ser definido con la cláusula TABLES, USING o CHANGING. El significado de cada una de estas cláusulas:

- TABLES -> Los parámetros definidos con esta cláusula sólo pueden ser tablas internas, con o sin línea de cabecera.
- USING -> Los parámetros definidos con esta cláusula pueden ser de cualquier tipo de dato, incluidas las tablas internas. Se suelen utilizar como parámetros de entrada a la subrutina.
- CHANGING -> Los parámetros definidos con esta cláusula pueden ser de cualquier tipo de dato, incluidas las tablas internas. Se suelen utilizar como parámetro de salida de la subrutina.

Los parámetros formales con USING o CHANGING pueden ser especificados en el parámetro actual utilizando los siguientes métodos:

- Llamada por referencia -> Es el método por defecto. No se especifica nada, tanto en los parámetros USING como en los parámetros CHANGING. En la llamada transferimos la dirección del objeto de dato. Cualquier modificación realizada sobre el objeto de dato tendrá efecto después de la llamada.
- Llamada por valor -> Se antepone a los parámetros la opción VALUE (el parámetro entre paréntesis) con la cláusula USING. En la llamada se crea una copia del parámetro, que será la utilizada por la subrutina. Cualquier modificación realizada en la subrutina no afectará al objeto de dato fuera de ésta.
- Llamada por valor y resultado -> Se antepone a los parámetros la opción VALUE (el parámetro entre paréntesis) con la cláusula CHANGING. En la llamada se crea una copia del parámetro, que será la utilizada por la subrutina. Cualquier modificación realizada en la

subrutina (copia de parámetro actual) afectará al objeto de dato (parámetro actual) sólo si la subrutina termina correctamente, es decir, si no se ha ejecutado ninguna sentencia MESSAGE. Este tipo de método sólo tiene sentido en programación de transacciones.

En los parámetros formales (sentencia FORM) se puede especificar un tipo de dato para asegurarnos el tipo de éste. Las conversiones de tipo realizadas entre los parámetros actuales y los parámetros formales cumplen las siguientes reglas:

Especificación de tipo Conversión

Sin especificación / TYPE ANY El sistema acepta cualquier tipo del parámetro actual. Todos los atributos del parámetro actual se traspasan al parámetro formal.

TYPE TABLE El sistema comprueba si el parámetro actual es una tabla interna. Todos los atributos y estructura del parámetro actual se transporta al parámetro formal.

TYPE C, N, P o X El sistema comprueba si el parámetro actual es del tipo especificado. La longitud del parámetro y las especificamos de decimales, para el tipo P se transportan del parámetro actual al parámetro formal.

TYPE D, F, I o T El tipo de dato del parámetro actual debe coincidir con el tipo de dato del parámetro formal.

LIKE <campo> / TYPE <tipo-usuario>

Ejemplo 1:

```
DATA: RNAME(30) VALUE 'WRITE_STATISTIC',
     PNAME(8)  VALUE 'ZYX_STAT'.
PERFORM WRITE_STATISTIC(ZYX_STAT).
PERFORM (RNAME)        IN PROGRAM ZYX_STAT.
PERFORM WRITE_STATISTIC IN PROGRAM (PNAME).
PERFORM (RNAME)        IN PROGRAM (PNAME).
```

Ejemplo 2:

```
DATA: BEGIN OF ITAB OCCURS 100,
       TEXT(50),
       NUMBER TYPE I,
     END   OF ITAB.
     STRUC LIKE T005T.
...
PERFORM DISPLAY TABLES ITAB
             USING  STRUC.
FORM DISPLAY TABLES PAR_ITAB STRUCTURE ITAB
           USING  PAR STRUCTURE T005T.
  DATA LOC_COMPARE LIKE PAR_ITAB-TEXT.
  WRITE: / PAR-LAND1, PAR-LANDX.
  ...
  LOOP AT PAR_ITAB WHERE TEXT = LOC_COMPARE.
    ...
  ENDLOOP.
  ...
ENDFORM.
```

Ejemplo 3:

```
DATA: NUMBER_I TYPE I VALUE 5,
     NUMBER_P TYPE P VALUE 4,
     BEGIN OF PERSON,
       NAME(10)      VALUE 'Paul',
       AGE TYPE I    VALUE 28,
     END   OF PERSON,
     ALPHA(10)       VALUE 'abcdefghij'.
FIELD-SYMBOLS .
ASSIGN NUMBER_P TO .
PERFORM CHANGE USING 1
                 NUMBER_I
                 NUMBER_P
```

```
                PERSON
                ALPHA+NUMBER_I().
FORM CHANGE USING VALUE(PAR_1)
                PAR_NUMBER_I
                PAR_NUMBER_P
                PAR_POINTER
                PAR_PERSON STRUCTURE PERSON
                PAR_PART_OF_ALPHA.
  ADD PAR_1 TO PAR_NUMBER_I.
  PAR_NUMBER_P = 0.
  PAR_PERSON-NAME+4(1) = ALPHA.
  PAR_PERSON-AGE = NUMBER_P + 25.
  ADD NUMBER_I TO PAR_POINTER.
  PAR_PART_OF_ALPHA = SPACE.
ENDFORM.
```

El contenido de los campos después del PERFORM es:
NUMBER_I = 6
NUMBER_P = 6
PERSON-NAME = 'Paula'
PERSON-AGE = 25
ALPHA = 'abcde j'
Véase también: FORM.

POSITION

Definición
Para especificar una posición de salida horizontal, SAP ofrece dos posibilidades. La primera con la cláusula AT en las sentencias WRITE y ULINE, y la segunda con la sentencia POSITION.
Sintaxis:
POSITION <columna>.
El sistema, tras ejecutar la sentencia, pone la posición de salida horizontal y la variable del sistema SY-COLNO con el valor <columna>. Si <columna> sobrepasa los límites del informe, las subsiguientes sentencias de escritura se ignoran.
Las variables del sistema SY-COLNO y SY-LINNO contienen la posición de la columna actual y la línea actual respectivamente. Aunque estas variables se pueden actualizar es recomendable no hacerlo y sólo utilizarlas de lectura. Si las actualizamos SAP no garantiza que el resultado del programa sea correcto.
Ejemplo:

```
FORM LINEOUTPUT USING COLUMN LENGTH CHAR.
  DATA LINEPOS TYPE P.
  LINEPOS = COLUMN.
  DO LENGTH TIMES.
    POSITION LINEPOS. WRITE CHAR.
    ADD 1 TO LINEPOS.
  ENDDO.
ENDFORM.
```

Si llamamos al form de esta forma:
PERFORM LINEOUTPUT USING 5 10 '='.
El resultado sería:
==========
Véase también: WRITE...AT.

PRINT-CONTROL

Definición

Para definir el formatop de impresión se utiliza la sentencia PRINT-CONTROL. El formato de la sentencia es el siguiente:

Sintaxis:

PRINT-CONTROL <formato> [LINE <línea>] [POSITION <columna>].

Sin la cláusulas LINE y POSITION, esta sentencia pone el formato de impresión especificado en <formato> a partir de la posición actual de salida (campos del sistema SY-COLNO y SY-LINNO). Con la cláusula LINE el formato de impresión se aplica a partir de la línea <línea>. Con la cláusula POSITION el formato de impresión se aplica a partir de la posición <posición>. Esta sentencia sólo tiene efecto si el listado es enviado directamente al spool, con o sin impresión inmediata. Si la impresión se realiza en pantalla no tiene efecto.

En <formato> se puede especificar varias opciones de impresión. El sistema convierte el valor de la sentencia mediante un código interno. El sistema convierte el código, junto con la impresora especificada, en una secuencia de escape que la impresora interpreta. Los opciones para <formato> son las siguientes:

Formato	**Código**	**Significado**
CPI <cpi>	CI<cpi>	Caracteres por pulgada
LPI <lpi>	LI<lpi>	Líneas por pulgada
COLOR BLACK	CO001	Color negro
COLOR RED	CO002	Color rojo
COLOR BLUE	CO003	Color azul
COLOR GREEN	CO004	Color verde
COLOR YELLOW	CO005	Color amarillo
COLOR PINK	CO006	Color rosa
LEFT MARGIN <m>	LM<m>	Margen izquierdo
FONT <font>	FO<font>	Font de impresión
FUNCTION <código>	<código>	Especificación directa de código

Con la opción FUNCTION se puede especificar cualquier código que exista en el sistema. Los print-control se definen a través de la transacción SAPD (Herramientas -> Gestión; Spool -> Administración del Spool).

También la sentencia PRINT-CONTROL nos permite incluir líneas en el spool que luego no serán impresas.

PRINT-CONTROL INDEX-LINE <campo>.

Esta sentencia escribe el contenido del campo <campo> en una línea determinada *línea-indice* a partir de la línea actual de salida. Esta línea se almacenará en el spool, y será visible desde el spool pero no será impresa. Con esta sentencia se puede incluir información en los informes, por ejemplo, para la administración del spool.

PROCESS

Definición

Sentencia de evento que indica el inicio del proceso PBO. Sentencia de evento que indica el inicio del proceso PAI. Sentencia de evento que marca el inicio del proceso donde se definen las ayudas no estándar de campos. Sentencia de evento que marca el inicio del proceso donde se definen los posibles valores para un campo.

Sintaxis:

PROCESS BEFORE OUTPUT.

El proceso PBO se suele utilizar para iniciar los campo de la pantalla, posicionar el cursor, mostrar o esconder campo o cambiar los atributos de los campos dinámicamente.

PROCESS AFTER INPUT.

El proceso PAI se suele utilizar para validar las entradas realizadas por el usuario, procesar la posición del cursor o procesar las funciones activadas.

PROCESS ON HELP-RESQUEST.

Este evento se procesa si el usuario pulsa la tecla F1 estando el cursor sobre algún campo de la pantalla. Si el evento se define el campo en cuestión (sentencia FIELD), la ayuda que aparece será la programa en la transacción.
PROCESS ON HELP-RESQUEST.
Este evento se procesa si el usuario pulsa la tecla F4 estando el cursor sobre algún campo de la pantalla. Si el evento se define el campo en cuestión (sentencia FIELD), la ayuda que aparece será la programa en la transacción.

PROGRAM

Definición
Se definen ciertas características de un programa.
Sintaxis:
En las versiones actuales de la aplicación SAP R/3, la sentencia PROGRAM tiene el mismo efecto que la sentencia REPORT.
Véase también: REPORT.

PROVIDE .. ENDPROVIDE

Definición
Recupera objetos de datos de tablas internas.
Sintaxis:

```
PROVIDE <campo11> <campo12> ...  FROM <tabla1>
       <campo21> <campo22> ...  FROM <tabla2>
...
       *                  FROM <tablan>
...
BETWEEN <valor1> AND <valor2>.
```

Ejemplo:

```
DATA: BEGIN OF SE OCCURS 3,
       FROM     TYPE D,
       TO       TYPE D,
       NAME(15) TYPE C,
       AGE TYPE I,
      END OF SE,
      BEGIN OF PR OCCURS 4,
       START    TYPE D,
       END      TYPE D,
       PRICE    TYPE I,
       NAME(10) TYPE C,
      END OF PR,
      BEGIN OF SH OCCURS 2,
       CLOSED   TYPE D,
       STR(20)  TYPE C,
       OPENED   TYPE D,
      END OF SH VALID BETWEEN OPENED AND CLOSED,
      BEGIN TYPE D VALUE '19910701',
      END   TYPE D VALUE '19921001'.
SE-FROM = '19910801'. SE-TO  = '19910930'.
SE-NAME = 'Shorty'.   SE-AGE = 19. APPEND SE.
SE-FROM = '19911005'. SE-TO  = '19920315'.
SE-NAME = 'Snowman'.  SE-AGE = 35. APPEND SE.
SE-FROM = '19920318'. SE-TO  = '19921231'.
SE-NAME = 'Tom'.      SE-AGE = 25. APPEND SE.
PR-START = '19910901'. PR-END = '19911130'.
```

```
PR-NAME  = 'Car'.   PR-PRICE = 30000. APPEND PR.
PR-START = '19911201'. PR-END = '19920315'.
PR-NAME  = 'Wood'.  PR-PRICE = 10.    APPEND PR.
PR-START = '19920318'. PR-END = '19920801'.
PR-NAME  = 'TV'.    PR-PRICE = 1000.  APPEND PR.
PR-START = '19920802'. PR-END = '19921031'.
PR-NAME  = 'Medal'. PR-PRICE = 5000.  APPEND PR.
SH-CLOSED = '19920315'. SH-STR = 'Gold Avenue'.
SH-OPENED = '19910801'. APPEND SH.
SH-CLOSED = '19921031'. SH-STR = 'Wall Street'.
SH-OPENED = '19920318'. APPEND SH.
PROVIDE NAME AGE FROM SE
      NAME     FROM PR
      *        FROM SH
      BETWEEN BEGIN AND END.
  ...
ENDPROVIDE.
```

PUT

Definición

Es una sentencia nueva que aparece en el programa de las BDL. Esta sentencia permite interactuar entre los dos programas.

Sintaxis:

PUT <tabla_BD>.

Esta setencia sólo puede utilizarse en programas de bases de datos lógicas. Esta sentencia activa el evento GET para la tabla especificada en <tabla_BD> en el programa asociado a la BDL. Las áreas de trabajo de las tablas definidas en la BDL y el programa se comparten.

Véase también: GET, CHECK, REJECT.

RAISE

Definición

Activa una excepción dentro de un módulo de función.

Sintaxis:

RAISE EXCEPCIÓN.

Activa una excepción dentro de un módulo de función. Esta sentencia se tiene que ejecutar dentro de un módulo de función.

Ejemplo:

```
    FUNCTION-POOL CSTR.
    FUNCTION STRING_SPLIT.
      ...
      IF STRING NA DELIMITER.
        RAISE NOT_FOUND.
      ENDIF.
      ...
    ENDFUNCTION.
```

El programa que llama a la función contiene el siguiente código:

```
    PROGRAM EXAMPLE.
    ...
    CALL FUNCTION 'STRING_SPLIT'
    *    ...
         EXCEPTIONS
```

```
        NOT_FOUND = 7.
  IF SY-SUBRC = 7.
    WRITE / 'Hay un problema.'.
  ELSE.
    ...
  ENDIF.
```

Véase también: CALL FUNCTION.

RANGES

Definición

Con esta sentencia declaramos una tabla interna con el mismo formato que el utilizado en los criterios de selección. El objeto declarado con esta sentencia podrá ser utilizado en una expresión lógica con el operador IN.

Sintaxis:

RANGES <criterio> FOR <campo>.

Se declara el criterio de selección <criterio>, el cual referencia al campo <campo>. <campo> puede ser un campo de una tabla de la base de datos, o cualquier otro tipo de campo. Se puede declarar una tabla con los campos de un criterio de selección: SIGN, OPTION, LOW y HIGH.

Véase también: SELECT-OPTIONS.

READ CALENDAR

Definición

Sentencia que recupera información del calendario. Sólo es válida en el entorno R/2 de SAP. En el entorno R/3 ha sido sustituida por módulos de función.

READ CURRENT LINE

Definición

Vuelve a una línea ya tratada, por ejemplo, por el evento de la tecla de función F2, o por la sentencia READ LINE.

Sintaxis:

READ CURRENT LINE [FIELD VALUE <c11> [INTO <c12>] ... <cn1> [INTO <cn2>]].

La cláusula FIELD VALUE tiene el mismo efecto que el visto en la sentencia READ LINE.

Véase también: MODIFY CURRENT LINE.

READ DATASET

Definición

Lee un fichero secuencial en el servidor de aplicación.

Sintaxis:

READ DATASET <fichero> INTO <campo> [LENGTH <longitud>].

Lee datos del fichero secuencial <fichero> sobre el campo <campo>. Para decidir el formato de <campo> es necesario conocer la estructura del fichero. El modo de transferencia se indica en la sentencia OPEN DATASET. Si el fichero no esta abierto para lectura, el sistema intenta abrir el fichero en modo binario, o con las opciones de la última sentencia OPEN DATASET sobre ese fichero. No obstante, es recomendable abrir siempre el fichero con la sentencia OPEN DATASET. Si la operación de lectura ha sido satisfactoria la variable SY-SUBRC vale 0, si vale 4 quiere decir que hemos llegado al final del fichero, y si vale 8 es que el fichero no ha podido ser abierto.

Trabajando en el modo binario, se puede utilizar la cláusula LENGTH para que el sistema actualice el campo <longitud> con la longitud del campo leído.
Ejemplo:

```
DO.
  READ DATASET '/usr/test' INTO REC.
  IF SY-SUBRC <> 0.
    EXIT.
  ENDIF.
  WRITE: / REC-TEXT, REC-NUMBER.
ENDDO.
```

Véase también: OPEN DATASET, CLOSE DATASET, TRANSFER.

READ LINE

Definición
Permite leer líneas de un listado.
Sintaxis:

```
READ LINE <línea> [ INDEX <índice> ]
[ FIELD VALUE <c11> [ INTO <c12> ] ... <cn1> [ INTO <cn2> ] ]
[ OF CURRENT PAGE | OF PAGE <página> ].
```

Esta sentencia sin utilizar ninguna opción lee la línea con índice <línea> en el listado actual, actualizando la variable SY-LISEL con el contenido de la línea, y actualizando todos los campos asociados a la línea con la sentencia HIDE. Si la línea indica existe, SY-SUBRC toma el valor 0, en caso contrario toma el valoe 4. Las distintas opciones de la sentencia tienen los siguientes efectos:

- INDEX -> La línea que hay que leer será del listado con el nivel determinado en <índice>. podrá ser una línea del listado básico (valor 0) o de un listado secundario (valor 1, 2, ...)
- FIELD VALUE -> El sistema interpreta la salida del campo <c11> sobre la línea <línea> para almacenarlo sobre el mismo campo o sobre el campo alternativo <c12> si utilizamos la opción INTO.
- OF CURRENT PAGE -> Con está cláusula el valor de <línea> no se refiere al listado completo, sino a la posición en la página actual.
- OF PAGE -> Con esta cláusula el valor de <línea> no se refiere al listado completo, sino a la posición relativa de la página <página>.

Véase también: READ CURRENT LINE.

READ REPORT

Definición
Lee un programa de la base de datos.
Sintaxis:

```
READ REPORT <programa> INTO <tabla>.
```

Lee el programa <programa> de la base de datos sobre la tabla interna <tabla>. La longitud de la línea de la tabla interna debe ser, al menos, de 72 caracteres. Si el programa lee, SY-SUBRC vale 0, en caso contrario vale distinto de 0.
Ejemplo:

```
DATA: BEGIN OF INT_TABLE  OCCURS 100,
       COMP1,
       COMP2,
       COMP3,
     END OF INT_TABLE.
FORM PUT_ENTRY USING ENTRY LIKE LINE OF INT_TABLE.
  READ TABLE INT_TABLE WITH KEY COMP2 = ENTRY-COMP2
```

```
                BINARY SEARCH
                TRANSPORTING NO FIELDS.
  IF SY-SUBRC <> 0.
    INSERT ENTRY INTO INT_TABLE INDEX SY-TABIX.
  ENDIF.
ENDFORM.
```
Véase también: LOOP .. AT, INSERT, MODIFY.
.

READ TABLE

Definición
Se utiliza para leer una sola línea de una tabla interna.
Sintaxis:
READ TABLE <tabla> [INTO <área-trabajo>] INDEX <índice>.
Con la cláusula INTO especificamos un área de trabajo distinto a la línea de cabecera de la tabla. La cláusula INTO es opcional si la tabla especificada tiene línea de cabecera. El sistema lee la línea correspondiente al índice <índice>. Si el sistema encuentra la línea con el índice indicado la variable SY-SUBRC valdrá 0 y SY-TABIX contiene el índice de la línea 0; sino, SY-SUBRC es distinto de 0. Si <índice> es menor o igual a 0 se produce un error en tiempo de ejecución. Si el índice excede del tamaño de la tabla, SY-SUBRC es 4.
READ TABLE <tabla> [INTO <área-trabajo>] WITH KEY <clave> [BINAY SEARCH].
Con esta variante se puede leer de una tabla interna a través de una clave particular o una tabla estándar. Con esta variante también se puede ejecutar una búsqueda binaria en lugar de la secuencial. <clave> puede tener las siguientes opciones:

- WITH KEY <campo1> = <valor1> ... <campon> = <valorn> -> Los campos <campo1> son campos de la tabla. Los campos <valor1> son los valores que deben coincidir con los campos de la tabla.
- WITH KEY = <valor> -> <valor> es una línea completa de la tabla.
- WITH KEY <valor> -> Con esta opción el sistema compara la parte izquierda de la tabla con el campo <valor>. La longitud que hay que comparar viene definida por la longitud del campo <valor>.

READ TABLE <tabla> [INTO <área-trabajo>] [BINAY SEARCH].
Con esta variante leemos una línea con la clave estándar de la tabla interna. Esta variante sólo puede ser utilizada con tablas con cabecera de línea.
Ejemplo:

```
DATA: BEGIN OF INT_TABLE  OCCURS 100,
      COMP1,
      COMP2,
      COMP3,
    END OF INT_TABLE.
FORM PUT_ENTRY USING ENTRY LIKE LINE OF INT_TABLE.
  READ TABLE INT_TABLE WITH KEY COMP2 = ENTRY-COMP2
                BINARY SEARCH
                TRANSPORTING NO FIELDS.
  IF SY-SUBRC <> 0.
    INSERT ENTRY INTO INT_TABLE INDEX SY-TABIX.
  ENDIF.
ENDFORM.
```
Véase también: LOOP .. AT, INSERT, MODIFY.
.

READ TEXTPOOL

Definición
Lee los elementos de textos de un programa.
Sintaxis:
READ TEXTPOOL <programa> INTO <tabla> LANGUAJE <lenguaje>.
Lee los elementos de texto del programa <programa> en el lenguaje <lenguaje> añadiéndolos a la tabla interna <tabla>. La tabla interna debe tener la estructura TEXTPOOL. Si los datos son leídos correctamente SY-SUBRC vale 0, en caso contrario vale 4.
La estructura TEXTPOOL es la siguiente:

Campo	**Significado**
ID	Identifica el tipo de elementos de texto: R -> Título del programa T -> Cabecera del listado H -> Cabecera de columna S -> Textos de selección T -> Textos simbólicos
KEY	Campo clave. En función del tipo de elemento de texto tiene distintos significados: H -> Número de línea en la cabecera de columna S -> Nombre del parámetro o criterio de selección (8 caracteres) I -> Número del texto simbólico
ENTRY	Texto asociado al elemento de texto
LENGTH	Longitud del texto

Ejemplo:

```
DATA: PROGRAM(8) VALUE 'PROGNAME',
     TAB LIKE TEXTPOOL OCCURS 50 WITH HEADER LINE.
READ TEXTPOOL PROGRAM INTO TAB LANGUAGE SY-LANGU.
```

Véase también: INSERT TEXTPOOL, DELETE TEXTPOOL.
.

RECEIVE RESULTS FROM FUNCTION

Definición
Recupera los resultados de una función que ha sido ejecutada de forma asíncrona.
Sintaxis:
RECEIVED RESULTS FROM FUNCTION <función>
[IMPORTING <parámetro1> = <campo1> ... <parámetron> = <campon>]
[TABLES <parámetro1> = <campo1> ... <parámetron> = <campon>]
[EXCEPTIONS <excepción1> = <campo1> ... <excepciónn> = <campon>].
Recupera todos los parámetros de una función que ha sido ejecutada asincrónicamente, como por ejemplo con la sentencia CALL FUNCTION .. STARTING NEW TASK.
Ejemplo:

```
DATA: INFO LIKE RFCSI,
* Result of RFC_SYSTEM_INFO function
     SYSTEM_MSG(80) VALUE SPACE.
* Exception handling
CALL FUNCTION 'RFC_SYSTEM_INFO'
    STARTING NEW TASK 'INFO'
    PERFORMING 'RETURN_INFO' ON END OF TASK.
WRITE: 'Wait for reply'.
...
AT USER-COMMAND.
* Return from FORM routine RETURN_INFO
  IF SYSTEM_MSG = SPACE.
    WRITE: 'Destination =', INFO-RFCDEST.
```

```
  ELSE.
    WRITE SYSTEM_MSG.
  ENDIF.
...
FORM RETURN_INFO USING TASKNAME.
  RECEIVE RESULTS FROM FUNCTION 'RFC_SYSTEM_INFO'
     IMPORTING  RFCSI_EXPORT = INFO
     EXCEPTIONS SYSTEM_FAILURE MESSAGE SYSTEM_MSG.
  REFRESH SCREEN. "Simula el campo de comando = return key
ENDFORM.
```

Véase también: CALL FUNCTION .. STARTING NEW TASK.

REFRESH

Definición
Se utiliza para inicializar una tabla, con o sin línea de cabecera.
Sintaxis:
REFRESH <tabla>.
Después de procesar el sistema esta sentencia, la tabla interna no contiene ninguna línea.
Véase también: CLEAR, FREE.

REFRESH CONTROL

Definición
Inicializa un control de pantalla.
Sintaxis:
REFRESH CONTROL <tabla> FROM SCREEN <pantalla>.
Inicializa el control de pantalla <control> de la pantalla <pantalla>..
Véase también: CONTROLS.

REFRESH SCREEN

Definición
Refresca la pantalla de la interfaz gráfica de usuario (SAPGUI).
Sintaxis:
REFRESH SCREEN.
Inicializa el control de pantalla <control> de la pantalla <pantalla>.
Véase también: RECEIVE RESULTS FROM FUNCTION.

REJECT

Definición
Termina con el bloque de proceso del evento GET para ir a la siguiente iteración del mismo evento u otro de evento GET.
Sintaxis:
REJECT [<tabla>].
La sentencia sin parámetros es el de abandonar el actual bloque de proceso y continuar con la siguiente iteración del mismo evento GET. Con <tabla> en lugar de irnos al mismo evento GET, el sistema ejecuta el evento GET para la tabla indicada. <tabla> deber ser una tabla con jerarquía

superior (estructura jerárquica de tablas de una base de datos lógica) a la tabla tratada en el actual evento GET.
Ejemplo:
Base de datos lógica F1S
Jerarquía: SPFLI -> SFLIGHT -> SBOOK
TABLES: SFLIGHT,
SBOOK.
GET SFLIGHT.
...
GET SBOOK.
...
REJECT 'SFLIGHT'.
...
REJECT cancela el proceso del evento 'GET SBOOK' y continua con el procesamiento del evento 'GET SFLIGHT' .
Véase también: RECEIVE RESULTS FROM FUNCTION.

REPLACE

Definición
Reemplaza cierta parte de un string por otro string.
Sintaxis:
REPLACE <string1> WITH <string2> INTO <campo> [LENGTH <longitud>].
El sistema busca en el campo <campo> la primera ocurrencia del string <string1> para sustituirla por el string <string2>.
Con la cláusula LENGTH sólo se sustituirá los caracteres indicados en <longitud>. Si no se especifica la cláusula LENGTH la sustituición del string <string1> será completa. Si el string <string1> se encuentra en el campo <campo> es sustituido por <string2> y la variable del sistema SY-SUBRC toma el valor 0. En caso contrario, no se reemplaza nada y SY-SUBRC toma el valor 4. <string1>, <string2> y <longitud> pueden ser variables.
Ejemplo:
DATA FIELD(10).
MOVE 'ABCB' TO FIELD.
REPLACE 'B' WITH 'string' INTO FIELD.
Devuelve:
FIELD = 'AstringCB', SY-SUBRC = 0
Véase también: SEARCH, TRANSLATE, OVERLAY.

REPORT

Definición
Define ciertas características de un programa.
Sintaxis:
REPORT <nombre> [NO STANDARD PAGE HEADING]
[LINE-SIZE <ancho>]
[LINE-COUNT <líneas>]
[MESSAGE-ID <mm>].
Por defecto el sistema ofrece una cabecera estándar compuesta de una primera línea con el título del programa (sacado de los atributos del programa) y un número, y una segunda línea compuesta de una línea horizontal. El significado de las cláusulas es la siguiente:

- NO STANDARD PAGE HEADING -> Excluye la cabecera estándar.
- LINE-SIZE indicamos el número de columnas del informe, <ancho> es un literal numérico que indica ese valor.

- LINE-COUNT -> Indicamos el número de línea por página. El número de líneas está indicado por <líneas>. Si utilizamos el evento END-OF-PAGE habrá que indicar entre paréntesis el número de líneas en ese evento a continuación de las líneas por página.
- MESSAGE-ID -> Indica la clase de mensaje que utilizaremos en el programa. Esta cláusula no tiene ningún efecto en el formato de la página.

```
REPORT <programa> [ NO STANDARD PAGE HEADING ]
                  [ LINE-SIZE <ancho> ]
                [ LINE-COUNT <líneas> ]
                  [ MESSAGE-ID <mm> ]
                 [ DEFINING DATABASE <base-de-datos> ].
```

<programa> puede ser cualquier combinación de hasta 8 caracteres de longitud. El significado de las cláusulas puede ser el siguiente:

- NO STANDARD PAGE HEADING -> Suprimimos la cabecera estándar del sistema.
- LINE-SIZE -> Definimos el número de columnas por página. El máximo de columnas es de 255 caracteres. Si no especificamos la cláusula, el programa tendrá el número de columnas correspondiente a la pantalla actual. Este valor se guarda en la variable del sistema SY-LINSZ. El campo <columna> no deber ser escrito entre comillas. Si la salida va a ser por impresora, hay que tener en cuenta que ciertas impresoras no admiten más de 132 caracteres.
- LINE-COUNT -> Indica el número de filas por páginas, <filas> indica el número de líneas y <n> el número de líneas para el pie de página (evento END-OF-PAGE). Si omitimos <n> el evento END-OF-PAGE no tendrá efecto. La variable del sistema SY-LINCT contiene el número actual de líneas por página en un listado. Si la cláusula no se especifica, el número de líneas por página se obtiene a partir de la variable del sistema SY-LINCT. El campo <fila> no debe ser escrito entre comillas.
- MESSAGE-ID -> Definimos la clase de mensaje que utilizará el programa. <clase> define la clase de mensaje y debe existir en la tabla de mensajes T100. <clase> no debe ir encerrada entre comillas.
- DEFINING DATABASE se utiliza exclusivamente en los programas de definición de bases de datos lógicas. Estas cláusula se incluye automáticamente si utilizamos las herramientas estándar de creación de bases de datos lógica.

Es aconsejable seguir las reglas de nomenclatura de SAP para objetos de cliente, sobretodo para los nombres de los programas. Un programa que no siga la norma puede ser borrado accidentalmente con un upgrade del sistema. Los programas de usuario deben comenzar por "Z" o por "Y". El resto de caracteres es de libre uso.

RESERVE

Definición

Para forzar un salto de página en función de cierta condición se utiliza la sentencia RESERVE.

Sintaxis:

```
RESERVE <n> LINES.
```

Forzamos un salto de página si en la página actual quedan menos de <n> líneas entre la línea actual de salida y el pie pagina. Las líneas del pie de página no se incluyen en el cálculo. <n> puede ser una variable. Antes de comenzar una nueva página el sistema procesa el evento END-OF-PAGE. La sentencia RESERVE sólo tiene efecto si a continuación hay sentencias de escritura. También puede ser utilizada en combinación con la sentencia BACK marcando un bloque de líneas.

Véase también: NEW-PAGE, BACK.

ROLLBACK WORK

Definición
Deshace los cambios realizados en la base de datos.
Sintaxis:
ROLLBACK WORK.
Si la operación se realiza satisfactoriamente, SY-SUBRC vale 0, en caso contrario, SY-SUBRC toma un valor distinto de 0..
Véase también: COMMIT WORK.

SCAN

Definición
Analiza un programa.
Sintaxis:
SCAN ABAP-SOURCE <tabla1> TOKENS INTO <tabla2>
[STATEMENTS INTO <tabla3>] [FROM <n1> TO <n2>] [KEYWORDS FROM <tabla4>]
[LEVELS INTO <tabla5>] [OVERFLOW INTO <campo1>] [WITH ANALYSIS]
[WITH COMMENTS] [WITH INCLUDES] [WITHOUT TRMAC] [PROGRAM FROM <campo2>]
[INCLUDED INTO <campo3>] [MESSAGE INTO <campo4>]
[WORD INTO <campo5>] [LINE INTO <n3>] [OFFSET INTO <n4>].
Desglosa el código fuente de un programa contenido en la tabla <tabla1> en elementos sobre la tabla <tabla2> (estructura STOKEN o STOKEX si se especifica la cláusula WITH ANALYSIS). Los comentarios son eliminados del análisis a no ser que utilicemos la cláusula WITH COMMENTS.
Con la cláusula STATEMENTS INTO cada sentencia del programa que hay que analizar se incluye en la tabla <tabla3> (estructura SSTMNT). Las sentencias anidadas se dividen en sentencias elementales. Los códigos de retorno devueltos por el sistema (se almacenan en la variable SY-SUBRC) son los siguientes:

Valor	Significado
0	La tabla con el código fuente no está vacía, no contiene errores sintácticos y se ha podido subdividir en elementos.
1	La tabla con el código fuente no está vacía y se ha podido subdividir en elementos pero al menos un programa include no existe. Este error sólo puede ocurrir si se pone la cláusula WITH INCLUDES.
2	La tabla con el código fuente está vacía (o el rango seleccionado con las cláusulas FROM y TO).
4	La sentencia detecta errores en el código fuente.
8	Otro tipo de error.

El resto de las sentencias es la siguiente:

- FROM y TO -> Con ellas delimitamos las líneas de la tabla interna que hay que analizar.
- KEYWORDS FROM -> Solo tratamos las sentencias contenidas en la tabla <tabla4>. Si la tabla esta vacía se analizan todas las sentencias.
- LEVELS INTO -> Expande sobre la tabla <tabla5> un mayor detalle de ciertas sentencias de la tabla <tabla2>, como por ejemplo la sentencia INCLUDE. <tabla5> tiene la estructura SLEVEL.

Véase también: SYNTAX-CHECK.

SCROLL LIST

Definición
Se utiliza para hacer un desplazamiento (scrolling) del informe de salida, por ejemplo, como respuesta a una entrada por parte del usuario. El scrolling puede ser horizontal o vertical.
Sintaxis:

SCROLL LIST { FORWARD | BACKWARD } [INDEX <índice>] [<n> PAGES].
Con la cláusula FORWARD el scrolling se realiza hacía adelante. Con BACKWARD se realiza hacía atrás. Sin la opción INDEX, el scrolling se realiza desde la posición actual en listado actual, una página adelante oatrás en función de la primera cláusula. Con la opción INDEX el scrolling se realiza en el nivel marcado por <índice>. Sin la cláusula PAGE se avanza o retrocede una página.
En todas las variantes que veremos de la sentencia la cláusula INDEX tiene el mismo significado que el visto anteriormente.
La sentencia SCROLL tiene efecto sobre el listado completo. La sentencia no tiene efecto si se utiliza antes de la primera sentencia de salida del listado. Si se utiliza después de la primera sentencia de salida, el efecto de la sentencia es para todo el listado, incluyendo posibles sentencias de salida posteriores.
SCROLL LIST TO { FIRST PAGE | LAST PAGE | PAGE <página> }
[INDEX <índice>] [LINE <línea>].
El informe se desplazará hasta la primera página con la opción FIRST PAGE, a la última página con la opción LAST PAGE, o a una página determinada con la opción PAGE <página>. Con la opción LINE el sistema muestra la página en la cual se encuentra la línea <línea>. Las líneas de la cabecera y pie de página no se consideran a la hora de contar las líneas del informe.
SCROLL LIST { LEFT | RIGHT } [BY <n> PLACES] [INDEX <índice>].
El scrolling se realiza horizontalmente, con LEFT hasta el margen izquierdo, con RIGHT hasta el margen izquierdo. Con la cláusula BY especificamos el número de columnas que queramos desplazar.
SCROLL LIST { LEFT | RIGHT } [BY <n> PLACES] [INDEX <índice>].
Desplaza el informe horizontalmente hasta la columna <col>. El desplazamiento, por lo tanto, podrá ser hacia la izquierda o hasta la derecha en función de la columna actual de listado.

SEARCH

Definición
Se utiliza para buscar una cadena de caracteres en un campo alfanúmerico.
Sintaxis:
SEARCH <campo> FOR <str> { ABREVIATED | STARTING AT <n1> | ENDING AT <n2> | AND MARK }.
Esta sentencia busca en el campo <campo> la secuenca de caracteres <str>. Si se encuentra, la variable del sistema SY-SUBRC vale 0 y SY-FDPOS toma el valor del offset del string encontrado. En caso contrario, SY-SUBRC vale 4. El string <str> puede contener caracteres con cierto significado particular.

Operador	Significado
<string>	Cualquier sentencia de caracteres. Los espacios en blanco se ignoran.
.<string>.	Cualquier sentencia de caracteres. Los espacios en blanco no se ignoran.
*<string>	Cualquier palabra terminada con el string especificado.
<string>*	Cualquier palabra que empiece con el string especificado.

- ABREVIATED -> Los caracteres de <string> a buscar en <campo> pueden estar separados por otros caracteres, pero el conjunto de caracteres deben formar un sola palabra.
- STARTING AT -> La búsqueda en el campo <campo> se realiza a partir de la posición <n1>. El resultado en SY-FDPOS es relativo a la posición <n1> y no al principio del campo <campo>.
- ENDING AT -> Limitamos la búsqueda hasta la posición <n2>.
- AND MARK -> Si la búsqueda es satisfactoria, todos los caracteres del string de búsqueda se convierten en mayúsculas.

Véase también: REPLACE, OVERLAY, SHIFT, SPLIT, TRANSLATE.

SELECT .. ENDSELECT

Definición

Se utiliza para leer un/os registro/s de una tabla de la base de datos. Lectura de un único registro de una tabla de la base de datos.

Sintaxis:

```
SELECT [ SINGLE ] <opciones-select>
    FROM <opciones-from>
    [ INTO <opciones-into  | APPENDING <opciones-appending> ]
    [ WHERE <opciones-where> ]
    [ GROUP BY <opciones-group> ]
    [ ORDER BY <opciones-order> ].
    ...
    [ ENDSELECT. ]
```

- Con la cláusula SELECT identificamos los campos que queremos leer, si se usa con la cláusula FROM es obligatoria e identifica la tabla que hay que leer.
- INTO o APPENDING -> Son opcionales e identifican los campos destino de la lectura.
- WHERE -> Es opcional e identifica las condiciones de selección.
- GROUP BY -> Es opcional y sirve para la agrupación de campos.
- ORDER BY -> Es opcional y sirve para marcar el criterio de ordenación.
- ENDSELECT -> Marca el final de un bloque si la sentencia SELECT no se ha puesto la opción SINGLE.

La sentencia *Open SQL* utilizan de forma automática el campo mandante. Las sentencias acceden a tablas dependientes de mandante leen y procresan sólo los datos del mandante actual (mandante de conexión). Sin la cláusula CLIENT SPECIFIED no es posible utilizar el campo mandante en la cláusula WHERE (provocan un error de sintaxis). Si rellenamos el campo mandante en las sentencias INSERT, UPDATE o DELETE no se produce ningún error en tiempo de ejecución, el sistema sobrescribe el campo mandante del área de trabajo con el mandante actual antes de procesar la sentencia Open SQL. Si deseamos especificar un mandante distinto al actual debemos utilizar la cláusula CLIENT SPECIFIED en las sentencias SELECT, INSERT, UPDATE, MODIFY o DELETE.

```
... CLIENT SPECIFIED.
```

Esta opción debe ir siempre a continuación del nombre de la tabla. Con esta opción desconectamos el automatismo del manejo del campo mandante, por lo tanto, habrá que rellenarlo en la sentencias de inserción o modificación.

La cláusula SELECT define la selección sencilla o múltiple de fila, define también si las filas duplicadas serán excluidas, así como las columnas que seleccionar.

```
SELECT [ SINGLE [ FOR UPDATE ]  ] [ DISTINCT ]
    { * | <campos> | MAX(<campo> ) AS <m> | MIN(<campo> ) AS <m> |
      AVG(<campo> ) AS <m> | SUM(<campo> ) AS <m> | COUNT(* ) as <m> | (lista)
      COUNT([ DISTINCT ] <campo> ) AS <m> }
...
[  ENDSELECT ].
```

Para leer todas las columnas de una tabla se utiliza el asterisco (*).

Con la cláusula SINGLE es necesario especificar todos los campos de la clave primaria en la cláusula WHERE para identificar de forma única un registro. Si la línea no se encuentra SY-SUBRC vale 4. Si el sistema consigue leer hasta una línea SY-SUBRC valdrá 0. Con la cláusula FOR UPDATE el sistema bloquea la línea leída. Sólo puede ser utilizada en combinación con la cláusula SINGLE. El programa espera hasta que recibe por parte del sistema confirmación del bloqueo. Si no se pudiera realizar el bloqueo el programa termina con un error. La cláusula no se pudiera realizar el bloqueo el programa termina con un error. La cláusula FOR UPDATE debe ser utilizada en combinación con los mecanismos de bloque y desbloqueo. Con la cláusula SINGLE leemos una sola línea de la tabla y sin la cláusula la selección es múltiple. Cada vez que el sistema nos deja una línea sobre el área de trabajo especificado, se ejecuta el bloque de sentencias hasta la sentencia ENDSELECT.

Con la opción DISTINCT automáticamente se excluyen las línea duplicadas. Si al menos se lee una línea de la tabla, SY-SUBRC vale 0, en caso contrario vale 4. La variable del sistema SY-DBCNT se incrementa en uno por cada paso del bucle. Al final de éste conserva el número de líneas leídas.
En lugar de leer todas las columnas de la tabla se puede leer ciertas columnas o aplicar ciertas funciones a dichas columnas. Veamos todas las posibilidades de la cláusula:

- * -> Son todos los campos de la tabla.
- <campo> -> Es una lista de campos de la tabla.
- MAX(<campo>) -> Devuelve el valor máximo de la columna <campo>.
- MIN(<campo>) -> Devuelve el valor mínimo de la columna <campo>.
- AVG(<campo>) -> Devuelve la media de valores de la columna <campo>.
- SUM(<campo>) -> Devuelve la suma de valores de la columna <campo>.
- COUNT([DISTINCT]<campo>) -> Devuelve el número de registros seleccionados. Con DISTINCT sólo se cuentan los registros distinto
- (<tabla>) -> Es una tabla interna donde se especifica cualquier opción de las vistas anteriormente.

Se deben dejar espacios en blanco entre los paréntesis y el argumento de la función. Con la opción AS se utiliza un nombre alternativo <m>. Si se especifican campos a leer o funciones, la cláusula INTO es obligatoria.

Opciones de la cláusula FROM

Esta cláusula especifica la tabla o vista de la base de datos que va a ser leída.
FROM { <tabla> | <vista> | <tabla> }
[CLIENT SPECIFIED] [BYPASSING BUFFER] [UP TO <n> ROWS]
La tabla o vista <tabla> debe estar definida en el diccionario de datos, y debe estar declarada en el programa en la sentencia TABLES. Con la opción BYPASSING BUFFER el sistema lee los datos de la tabla directamente de la base de datos, sin utilizar el buffer de la tabla. Con ellos nos aseguramos de que los datos son los más recientes. Cuando se define un tabla en el diccionario de datos se puede especificar que el sistema utilice un buffer local de la tabla. Este buffer se actualiza de forma asíncrona. Normalmente la sentencia SELECT utiliza este buffer, y no tiene por qué ser la versión más reciente. Para asegurar la lectura de la última versión, se utiliza esta opción. Si queremos leer un número determinado de líneas se utiliza la opción UP TO <n> ROWS. <n> determina el número de líneas a leer. Si <n> es igual a 0 se leen todas las líneas, si <n> es menor que 0, se produce un error en tiempo de ejecución. Si combinamos la opción UP TO <n> ROWS con la cláusula ORDER BY, el sistema primero ordena la tabla y posteriormente procesa las <n> primeras líneas.
Se puede especificar el nombre de la tabla en tiempo de ejecución, para ello, se utiliza la cláusula (<campo>). <campo> contiene el nombre de la tabla, y no tiene por qué declararse con la sentencia TABLES. Con esta opción la cláusula INTO es obligatoria.

Opciones de la cláusula INTO

Para especificar un área de destino de los datos seleccionados se utiliza la cláusula INTO de la sentencia SELECT.
INTO [CORRESPONDING FIELD OF]
{ <área> | TABLE <tabla> | [PACKAGE SIZE <n>] }
Esta cláusula es necesaria si queremos utilizar un área de trabajo distinto al área de trabajo de la tabla leída. Las áreas de trabajo de las tablas se generan automáticamente en el momento de declarar la tabla con la sentencia TABLES. La cláusula INTO también aparece en otras sentencias *Open SQL* como FETCH. El campo <área> debe estar declarado como objeto de datos y debe tener, al menos, la longitud de los campos leídos. Con la opción CORRESPONDING FIELD OF el sistema deja los campos sobre la estructura especificada, sobre los campos con el mismo nombre que los leídos. Con la opción TABLE la lectura se realiza en una sola operación (no es necesario ENDSELECT) y cada línea leída es insertada en la tabla interna. Si se especifica la cláusula PACKAGE SIZE la lectura se realiza en paquetes de <n> líneas (si es necesario ENDSELECT).

Opciones de la cláusula APPENDING

Con la cláusula APPENDING las líneas añadidas en la tabla interna se añaden al final-

APPENDING [CORRESPONDING FIELD OF]
TABLE <tabla> [PACKAGE SIZE <n>]

Las opciones CORRESPONDING FIELDS OF, TABLE y PACKAGE SIZE tienen el mismo significado que en la cláusula INTO.

Opciones de la cláusula WHERE

la cláusula WHERE nos permite indicar los criterios de selección. Esta cláusula también se utiliza en las sentencias UPDATE, DELETE y OPEN CURSOR.

WHERE { <campo_bd> <operador> <campo> |
<campo_bd> [NOT] BETWEEN <campo1> AND <campo2> |
<campo_bd> [NOT] LIKE <campo> [ESCAPE <e>] |
<campo_bd> [NOT] IN <campo1> (campo1>, ... , <campon>) |
<campo_bd> IS [NOT] NULL |
<campo_bd> [NOT] IN <criterio_seleccion> |
(<tabla_condiciones>) }
[{ NOT, AND, OR, (,) } ...]

El significado de las opciones de esta cláusula es el siguiente:

- <campo_bd> <operador> <campo> -> Los operadores válidos para <operador> son los siguientes:

Operador	Equivalente	Significado
EQ	=	Igual a
NE	<> o ><	No igual
LT	<	Menor que
LE	<=	Menor o igual a
GT	>	Mayor que
GE	>=	Mayor o igual que

- <campo_bd> [NOT] BETWEEN <campo1> AND <campo2> -> El valor del campo de la base de datos <campo_bd> debe estar comprendido (o no, si se utiliza NOT) entre los valores de los campos <campo1> y <campo2>. <campo1> y <campo2> pueden ser literales.
- <campo_bd> [NOT] LIKE <campo> [ESCAPE <e>] -> Esta opción sólo puede ser utilizada con campos alfanuméricos. El campo de la base de datos debe, o no, corresponder con el patrón <campo>. Patrón significa que el campo puede tener caracteres con un significado especial. El guión bajo (_) representa cualquier carácter individual, el carácter porcentaje (%) representa cualquier número de caracteres. Si queremos utilizar estos caracteres especiales como carácter significativo hay que anteponerle el carácter de escape <e>.
- <campo_bd> [NOT] IN <criterio_seleccion> <campo1> (campo1>, ... , <campon>) -> El valor de la base de datos debe, o no, ser el valor nulo.
- <campo_bd> IS [NOT] NULL -> El valor del campo de la base de datos debe, o no, ser el valor nulo.
- <campo_bd> [NOT] IN <criterio_seleccion> -> El campo de la base de datos debe, o no, cumplir las condiciones del criterio.
- (<tabla_condiciones>) -> Se puede especificar la expresión en tiempo de ejecución a través de la tabla interna <tabla_condiciones>. La tabla debe contener sólo un campo alfanúmerico de longitud 72. La tabla debe estar especificada entre paréntesis sin espacios en blanco entre los paréntesis y el nombre de la tabla. Puede estar combinada con expresiones estáticas gracias a los operadores AND y OR.

Se puede crear una expresión combinada gracias a los operadores AND, OR y NOT. NOT no tiene prioridad sobre AND y a su vez OR. Se pueden utilizar los paréntesis para marcar la prioridad en una expresión.

Hay un formato especial de la sentencia WHERE que es el siguiente:

FOR ALL ENTRIES IN <tabla> WHERE <condición>.

Con esta variante se puede especificar condiciones en tiempo de ejecución. En <condición> se puede especificar campos de la tabla interna y literales. Con esta variante no se puede utilizar en <condición> las opciones LIKE, BETWEEN e IN.

Opciones de la cláusula GROUP BY

Para combinar en contenido de un grupo de líneas de una tabla de la base de datos en una sola línea se utiliza la cláusula GROUP BY.

GROUP BY { <campo1> ... <campon> | (<tabla>) }

Con campo <campo1> ... <campon> especificamos los campos de agrupamiento. Con (<tabla>) se puede especificar los campos de agrupamiento dinámicamente, es decir, en tiempo de ejecución.

Opciones de la cláusula ORDER BY

Con dicha cláusula fijamos un criterio de ordenación en los datos.

```
ORDER BY { PRIMARY KEY   |
       <campo1> [ ASCENDING | DESCENDING ] ...
       <campon> [ ASCENDING | DESCENDING ]   |
       (<tabla>) }
```

Con la opción PRIMARY KEY se ordenan por los campo de la clave primaria ascendentemente. <campo1> puede ser cualquier campo de la tabla. Se puede especificar si el criterio de ordenación, en cada campo, es ascendente (por defecto) o descendente. Si especificamos más de un campo se ordenarán en la secuencia especificada. Con la opción (<tabla>) se puede indicar los campos de ordenación dinámicamente. La línea de la tabla debe ser del tipo C y longitud 72.

Ejemplo:

```
TABLES SBOOK.
SELECT * FROM SBOOK
  WHERE
    CARRID   = 'LH '      AND
    CONNID   = '0400'     AND
    FLDATE   = '19950228'
  ORDER BY PRIMARY KEY.
  WRITE: / SBOOK-BOOKID, SBOOK-CUSTOMID,   SBOOK-CUSTTYPE,
           SBOOK-SMOKER, SBOOK-LUGGWEIGHT, SBOOK-WUNIT,
           SBOOK-INVOICE.
ENDSELECT.
```

SELECTION-OPTIONS

Definición

Se utiliza para definir un criterio de selección. Algunas de las variantes de esta sentencia sólo se pueden utilizar en programas de definición de bases de datos lógicas.

Sintaxis:

```
SELECT-OPTIONS <criterio> FOR <campo>
      [ DEFAULT <g> [ TO <h> ] [ OPTION <opción> SIGN <signo> ]
      [ MEMORY ID <memoria> ] [ MODIF ID <clave> ]
      [ MATCHCODE OBJECT <objeto> ]
      [ NO-DISPLAY ] [ LOWER CASE ] [ OBLIGATORY ]
      [ NO-EXTENSION ] [ NO INTERVALS ] [ NO DATABASE SELECTION ]
      [ VALUE REQUEST ][ FOR LOW/HIGH ] ]
      [ VALUE-REQUEST [ FOR LOW/HIGH ] ]
      [ HELP-REQUESTM [ FOR LOW/HIGH ] ].
```

Esta sentencia crea el criterio de selección <criterio> para el campo <campo>. <campo> puede ser un campo de una tabla de la base de datos o un campo interno al programa. <criterio> puede ser como máximo de 8 caracteres de longitud. El criterio de selección lo rellena el usuario en la pantalla de selección. Los textos descriptivos que aparecen a la izquierda del criterio de selección se pueden cambiar utilizando (igual que con los parámetros definidos con PARAMETERS) con el objeto parcial a un programa, elementos de texto. El significado de las cláusulas es el siguiente:

- DEFAULT -> Se puede incluir una línea en el criterio de selección con valores por defecto. <g> actualiza el campo LOW del criterio de selección. Con TO <h> incluimos un valor por defecto en el campo HIGH del criterio de selección. El campo o literal <opción>, utilizado en OPTION, incluye un valor en el campo SIGN del criterio de selección. Todos los campos vistos en esta cláusula, <g>, <h>, <opción> y <signo> pueden ser variables o literales.
- NO-EXTENSION -> Restringe que el usuario sólo pueda introducir una línea en el criterio de selección. El sistema no presenta el campo de extensión que aparece a la derecha de los criterios de selección, por lo tanto, el usuario no puede realizar extensiones.
- NO-INTERVALS -> Restringe que el usuario no puede introducir intervalos y se limite a introducir únicamente valores sencillos. El sistema elimina la columna de valores *hasta*.
- NO DATABASE SELECTION -> Si el criterio de selección definido tiene otro objetivo que el de la selección de datos se puede utilizar esta cláusula para que no sea transportado al programa de la base de datos. Cuando declaramos un criterio de selección para un campo de una tabla de la base de datos y utilizamos un programa de bases de datos lógica (sentencia GET), el sistema, por defecto, transporta el criterio de selección al programa de la base de datos para limitar la lectura de registros.
- NO-DISPLAY, LOWER CASE, OBLIGATORY, MEMORY ID, MODIF ID y MATCHCODE OBJECT -> Se utilizan de la misma forma que en la sentencia PARAMETERS.
- VALUE-REQUEST y HELP-REQUEST -> Se utilizan exclusivamente en los programa de definición de bases de datos lógica.

Ejemplo 1:

```
TABLES SAPLANE.
...
SELECT-OPTIONS S_PTYPE FOR SAPLANE-PLANETYPE MODIF ID ABC.
...
AT SELECTION-SCREEN OUTPUT.
LOOP AT SCREEN.
  IF SCREEN-GROUP1 = 'ABC'.
    SCREEN-INTENSIFIED = '1'.
    MODIFY SCREEN.
  ENDIF.
ENDLOOP.
```

Ejemplo 2:

```
SELECT-OPTIONS DATE FOR SY-DATUM DEFAULT SY-DATUM.
```

Véase también: PARAMETERS.

SELECTION-SCREEN

Definición

En la pantalla de selección los parámetros y criterios de selección aparecen uno detrás de otro en filas distintas. Si este formato de pantalla no es suficiente para nuestras necesidades se puede utilizar la sentencia SELECTION-SCREEN para formatear la pantalla de selección.

Sintaxis:

```
SELECTION-SCREEN BEGIN OF LINE...
....
SELECTION-SCREEN END OF LINE.
SELECTION-SCREEN SKIP [ <n> ].
SELECTION-SCREEN ULINE [ [ / ] <posición> ( <longitud>) ] [ / ]
SELECTION-SCREEN POSITION <posición>.
SELECTION-SCREEN COMMENT [ / ]  <posición> (<longitud>) <campo>
[  FOR FIELD <campo2> ]  [ MODIF ID <clave>  ] .
```

```
SELECTION-SCREEN BEGIN OF BLOCK <bloque> [ WITH FRAME [ TITLE <título> ]
] [ NO INTERVALS ]
...
SELECTION-SCREEN END OF BLOCK <bloque>.
SELECTION-SCREEN FUNCTION KEY <i>.
SELECTION-SCREEN BEGIN OF VERSION <versión> TEXT-xxx.
SELECTION-SCREEN END OF VERSION <versión>.
SELECTION-SCREEN EXCLUDE
SELECTION-SCREEN DYNAMIC SELECTIONS FOR TABLE <tabla>.
SELECTION-SCREEN FIELD SELECTION FOR TABLE <tabla>.
```

El significado de las cláusulas es el siguiente:

- SKIP -> Provoca líneas en blanco en la pantalla de selección. <n> es opcional se utiliza para saltar más de una línea.
- ULINE -> Subraya una línea o parte de ella en la pantalla de selección. Si no utilizamos la opción <posición> (<longitud>) una nueva lína se crea. Si utilizamos la opción <posición>(<longitud>) la nueva línea comienza en la posición <posición> con una longitud de <longitud> caracteres. Con varios elementos sobre una línea se puede especificar <longitud> sin especificar <posición>.
- MODIF ID -> Tiene el mismo significado que en la sentencia PARAMETERS.
- COMMENT -> Se utiliza para escribir texto sobre la pantalla de selección. Con la barra (/) saltamos de línea y con <posición> y <longitud> definimos la posición y la longitud del texto en la pantalla. <campo1> puede ser un elemento de texto o un campo con una longitud máxima de 8 caracteres. Para asignar un texto a un parámetro o a un criterio de selección se utiliza la opción FOR FIELD, <campo2> identifica el nombre del parámetro o criterio de selección. La opción MODIF ID tiene el mismo significado que en la cláusula PARAMETERS.
- BEGIN OF LINE ... END OF LINE -> Se utiliza para situar en una misma línea varios parámetros y/o comentarios en la pantalla de selección. Cuando utilizamos esta cláusula el posible elemento de texto habrá que utilizar la cláusula COMMENT de la sentencia SELECTION-SCREEN. Los criterios de selección no se pueden incluir. La opción barra (/) no se puede especificar, y la opción <posición> se puede omitir.
- POSITION -> Sitúa un parámetro o un comentario en una posición determinada. Para <posición> se puede especificar un número, POS_LOW y POS_HIGH. POS_LOW y POS_HIGH son las posiciones del campo *desde* y campo *hasta* de un criterio de selección sobre la pantalla. Esta cláusula sólo puede ser utilizada en un bloque BEGIN OF LINE ... END OF LINE.
- BEGIN OF BLOCK ... END OF BLOCK -> Crea un bloque lógico sobre la pantalla de selección. Con la opción WITH FRAME el bloque se engloba con un cuadreo. Con la opción TITLE aparecerá el título <título> en la primera línea del cuadro. <título> puede ser un elemento de texto o un literal. Con la opción NO INTERVALS todas las sentencias SELECT-OPTIONS se procesan como si tuvieran la sentencia NO INTERVALS. Con esta opción, si utilizamos además WITH FRAME, el cuadro será menos ancho.

Se puede crear hasta cinco botones en la barra de botones sobre la pantalla de selección. Estos botones se conectan automáticamente a teclas de función.<i> debe estar comprendido entre 1 y 5. El texto que aparece sobre el botón se especifica en tiempo de ejecución moviendo un valor sobre el campo SSCRFIELDS-FUNCTXT_0<i>. La estructura SSCRFIELDS debe estar declarada con la sentencia TABLES. Cuando el usuario activa un botón, el valor FC0<i> es introducido sobre el campo SSCRFIELDS-UCOMM, el cual puede ser chequeado en el evento AT SELECTION-SCREEN.

Existe también la posibilidad de situar botones sobre la pantalla de selección con la opción PUSHBUTTON. Las opciones *barra* (/), posición>, <longitud> y MODIF ID <clave> son las mismas que las explicadas en la cláusula COMMENT. <campo> es el texto que aparece sobre el botón. Para <cmd> se debe especificar un código de hasta cuatro caracteres. Cuando el usuario *presiona* el botón, <cmd> es introducido en el campo SSCRFIELDS-UCOMM. Como sucedía en la cláusula anterior, SCCRFIELDS debe declararse con la sentencia TABLES. El campo SSCRFIELDS-UCOMM puede ser utilizado en el evento AT SELECTION-SCREEN.

El resto de cláusulas se utilizan exclusivamente en los programas de definición de bases de datos.
Ejemplo 1:
```
SELECTION-SCREEN BEGIN OF LINE.
  SELECTION-SCREEN COMMENT 1(10) TEXT-001.
  PARAMETERS: P1(3), P2(5), P3(1).
SELECTION-SCREEN END OF LINE.
```
Ejemplo 2:
```
SELECT-OPTIONS DATE FOR SY-DATUM DEFAULT SY-DATUM.
```
Ejemplo 3:
```
SELECTION-SCREEN BEGIN OF LINE.
  SELECTION-SCREEN COMMENT 10(20) TEXT-001
                   FOR FIELD PARM.
  SELECTION-SCREEN POSITION POS_LOW.
  PARAMETERS PARM LIKE SAPLANE-PLANETYPE.
SELECTION-SCREEN END OF LINE.
```
Ejemplo 4:
```
TABLES SSCRFIELDS.
  ...
SELECTION-SCREEN PUSHBUTTON /10(20) CHARLY USER-COMMAND ABCD.
  ...
INITIALIZATION.
  MOVE 'My text' TO CHARLY.
  ...
AT SELECTION-SCREEN.
  IF SSCRFIELDS-UCOMM = 'ABCD'.
    ...
  ENDIF.
```
Ejemplo 5:
```
TABLES SAPLANE.
SELECTION-SCREEN BEGIN OF BLOCK CHARLY
                 WITH FRAME TITLE TEXT-001.
  PARAMETERS PARM(5).
  SELECT-OPTIONS SEL FOR SAPLANE-PLANETYPE.
SELECTION-SCREEN END   OF BLOCK CHARLY.
```
Véase también: PARAMETERS, SELECT-OPTIONS, INITIALIZATION.

SET BLANK LINES

Definición
Se utiliza para escribir líneas en blanco con la cláusula AT de la sentencia WRITE.
Sintaxis:
SET BLANK LINES.
Por defecto el sistema no crea línea en blanco en el dispositivo de salida con la sentencia WRITE ... AT.
Una línea en blanco es una línea únicamente compuesta del carácter espacio en blanco. Con la opción ON el sistema no elimina las líneas en blanco creadas con la sentencia WRITE. Con la opción OFF dejamos la opción por defecto en el sistema.
Ejemplo:
```
DATA: BEGIN OF TEXTTAB OCCURS 100,
      LINE(72),
    END   OF TEXTTAB.
SET BLANK LINES ON.
LOOP AT TEXTTAB.
  WRITE / TEXTTAB-LINE.
```

```
ENDLOOP.
```

SET COUNTRY

Definición

Define el formato del punto decimal y el formato de la fecha para todas las sentencias de escritura (WRITE) que se utilicen a continuación, de acuerdo con la parametrización que se haya realizado sobre la tabla T005X.

Sintaxis:

SET COUNTRY <país>.

Con <país> identificamos el código de país a utilizar en la sentencia. Si el país existe en la tabla de países T005X, el código de retorno de la sentencia es 0, en caso contrario, es 4. Si país tiene el valor SPACE se formatea la salida al formato especificado en el registro maestro del usuario. El efecto de esta sentencia no se limita a la ejecución de un programa, sino que su efecto se extiende a toda el área de trabajo del usuario.

Ejemplo:

```
DATA: RECEIVER_COUNTRY LIKE T005X-LAND,
      DATE           LIKE SY-DATUM,
      AMOUNT           TYPE P DECIMALS 2.
...
SET COUNTRY RECEIVER_COUNTRY.
IF SY-SUBRC = 4.
  WRITE: / RECEIVER COUNTRY, ' es desconocido'.
ENDIF.
WRITE: / DATE, AMOUNT.
```

SET CURSOR

Definición

Sitúa el cursor en una posición determinada.

Sintaxis:

SET CURSOR <columna> <fila>.

Esta variante permite situar el cursor sobre la columna <columna> y fila <fila> respecto de la ventana actual de salida.

SET CURSOR [FIELD <campo>]LINE <línea> [OFFSET <offset>].

Esta variante permite situar el cursor sobre una posición del listado actual. El significado de las cláusulas es el siguiente:

- FIELD -> Es opcional. Si se utiliza sitúa el cursor sobre el campo especificado en <campo>. Si se ha utilizado la cláusula FIELD el desplazamiento será relativo al campo. Si no se ha utilizado la cláusula FIELD el desplazamiento será relativo a la línea.
- LINE -> Es obligatoria y determina el número de línea absoluto del listado.
- OFFSET -> Es opcional y determina el offset o desplazamiento utilizado.

Ejemplo:

```
MOVE 'MODUS' TO F.
MOVE '1234567890' TO MODUS.
DO 10 TIMES.
  NEW-LINE.   POSITION SY-INDEX   WRITE MODUS.
ENDDO.
AT LINE-SELECTION.
  SET CURSOR FIELD F LINE SY-LILLI.
```

SET EXTENDED CHECK

Definición
Esta sentencia activa o desactiva la comprobación sintáctica de un programa.
Sintaxis:
SET EXTENDED CHECK { ON | OFF }.
Con esta sentencia se puede marcar un conjunto de sentencias para que no sean consideradas en el verificador de sintaxis. Con ello se puede avanzar en la comprobación sintáctica. Estas sentencias no son consideradas en tiempo de ejecución.

SET LANGUAGE

Definición
Inicialización de los elementos de texto.
Sintaxis:
SET LANGUAGE <lenguaje>.
Inicializa todos los elementos de texto numerados y todos los literales de texto especificados en el programa al lenguaje especificado en la variable <lenguaje>.

SET LEFT SCROLL-BOUNDARY

Definición
Se utiliza para fijar una zona fija a la hora de realizar el scroll horizontal.
Sintaxis:
SET LEFT SCROLL-BOUNDARY [COLUMN <col>].
Sin la cláusula COLUMN la zona fija para el scroll horizontal es la marcada por la posición actual de la salida. Con la opción COLUMN, la zona fija viene marcada por la columna <col>. Sólo la zona de la derecha se desplaza con el scroll horizontal. Esta sentencia también afecta a la cabecera definida con la página de cabecera estándar. La sentencia sólo se aplica en la página actual, para que tenga efecto en todo un informe habrá que aplicarla en cada página, un buen sitio puede ser en la sentencia TOP-OF-PAGE.
Ejemplo:

```
DATA: NUMBER     TYPE I,
      SUB_NUMBER TYPE I.
NEW-PAGE NO-TITLE NO-HEADING.
DO 10 TIMES.
  NUMBER = SY-INDEX.
  DO 10 TIMES.
    SUB_NUMBER = SY-INDEX.
    WRITE: /(10) NUMBER,     '|',
           (10) SUB_NUMBER, '|'.
    SET LEFT SCROLL-BOUNDARY.
No haría falta ponerlo aquí porque ya esta declarado en el evento TOP-OF-PAGE .
    WRITE: 'Data 1', '|', 'Data 2', '|',  'Data 3', '|'. " ... 'Data n'
  ENDDO.
ENDDO.
ULINE.
TOP-OF-PAGE.
  ULINE.
  WRITE: /(10) 'No',     '|',
         (10) 'Sub_No', '|'.
  SET LEFT SCROLL-BOUNDARY.
  WRITE: 'DATA 1', '|', 'DATA 2', '|',  'DATA 3', '|'. " ... 'DATA n'
  ULINE.
```

SET LOCALE LANGUAGE

Definición
Las variables de entorno de texto se almacenan en el registro maestro del usuario, pero se puede modificar dichas variables, sólo para la ejecución del programa, a través de la sentencia SET LOCALE.
Sintaxis:
SET LOCALE LANGUAGE <lenguaje> [COUNTRY <país>][MODIFIER <mod>].
Esta sentencia pone la variable de entorno de texto en el lenguaje especificado en <lenguaje>. El significado de las cláusula es el siguientes:

- COUNTRY -> Se utiliza para especificar adicionalmente en país del lenguaje. Hay ciertos lenguajes que dependiendo del país, los criterios de ordenación varian.
- MODIFIER -> Especificamos el criterio de selección.

<país> y <mod> deben ser del tipo C y deben ser de la misma longitud que el campo clave de la tabla TCP0C. Las variables del entorno afectan a todas las operaciones que dependan del juego de caracteres.

SET MARGIN

Definición
Determina el margen superior-izquierdo en la impresión directa de un informe.
Sintaxis:
SET MARGIN <x> [<y>].
<x> determina el margen izquierdo. <y> es opcional y determina el margen superior de impresión. Esta sentencia sólo tiene efecto si el listado es enviado directamente al spool, con o sin impresión inmediata. Si la impresión se realiza en pantalla no tiene efecto.

SET PARAMETER

Definición
Actualización de un parámetro SPA/GPA.
Sintaxis:
SET PARAMETER ID '<parámetro>' FIELD <campo>.
El sistema actualiza el valor del parámetro <parámetro> con el contenido del campo <campo>. <parámetro> siempre es un literal alfanumérico de tres posiciones y debe ir con las comillas (' '). El valor anterior del parámetro se sobrescribe.
Véase también: GET PARAMETER.

SET PF-STATUS

Definición
Una interfaz de usuario está compuesta de un estatus y un título. Para poder utilizar funciones de usuario es necesario crear interfaz particulares. Estas interfaces se definen con la transacción *Menu Painter* (SE41). La sentencia PF-STATUS define un estatus para el listado actual (básico o secundario).
Sintaxis:
SET PF-STATUS { <status> | SPACE } [EXCLUDING { <función> | <tabla> }][INMEDIATELY].
<status> especifica el estatus que hay que activar y puede tener hasta ocho caracteres. El estará activo hasta que se active un nuevo estatus. Desde el propio editor ABAP/4 se puede ver la definición de un estatus si realizamos un doble click sobre el nombre del estatus. Si el estatus

existe directamente se visualiza, si no existe, el sistema nos pregunta si deseamos crear el estatus. El significado de las cláusulas es el siguiente:

- SPACE -> Indicamos al sistema que ha de utilizar el estatus estándar.
- EXCLUDING -> Desactiva funciones especificas. En un programa que necesite varios estatus que difieren muy poco, se puede crear un único estatus y eliminar funciones con esta cláusula. Para indicar una única función utilizamos la opción <función>. <función> es un literal o una variable que indica el código de la función a desactivar. También se puede utilizar una tabla interna para desactivar todas las funciones almacenadas en la tabla.
- INMEDIATILY -> Modificamos el estatus que en estos momentos esté activo. Si esta cláusula modificamos el estatus del siguiente listado a mostrar.

Ejemplo:

```
DATA: BEGIN OF TAB OCCURS 10,
      FCODE(4),
    END OF TAB.
REFRESH TAB.
MOVE 'DELE' TO TAB-FCODE.
APPEND TAB.
MOVE 'AUSW' TO TAB-FCODE.
APPEND TAB.
SET PF-STATUS 'STA3' EXCLUDING TAB.
```

Véase también: SET TITLEBAR.

SET PROPERTY

Definición

Actualiza una propiedad de un objeto.

Sintaxis:

SET PROPERTY OF <objeto> <propiedad> = <campo> [FLUSH].

Actualiza la propiedad <propiedad> del objeto <objeto> con el valor del campo <campo>. <objeto> debe ser del tipo OLE2_OBJECT. Con la cláusula FLUSH la sentencia no se ejecuta hasta la próxima sentencia OLE2 que no lleve la cláusula FLUSH. De esta forma se pueden agrupar las sentencias en una sola operación de transporte.

Ejemplo:

```
INCLUDE OLE2INCL.
DATA EXCEL TYPE OLE2_OBJECT.
CREATE OBJECT EXCEL 'Excel.Application'.
SET PROPERTY OF EXCEL 'Visible' = 1.
```

Véase también: GET PROPERTY, CALL METHOD, CREATE OBJECT.

SET RUN TIME ANALIZER

Definición

Sentencia que activa o desactiva la grabación de información en el fichero de análisis..

Sintaxis:

SET RUN TIME ANALYZER { ON [MODE <modo>] | OFF }.

Con la cláusula ON activa la grabación de la información en el fichero de análisis. Con la cláusula OFF se cierra el fichero de análisis y se desactiva la grabación. Con la opción MODE se especifica que tipo de información se graba en el fichero de análisis. <modo> es un literal numérico. Su representación binaria determina que información se almacena. Un 0 activa la grabación, un 1 la desactiva. Cada posición tiene un significado determinado:

- Byte 1: con un "1" la información referente a tablas internas (APPEND, COLLECT...) no se graba en el fichero de análisis.

- Byte 2: con un "1" cierta información técnica de la base de datos (OPEN, FETCH..) no se graba en el fichero de análisis.
- Byte 3: con un "1" las subrutinas (PERFORM) no se graban en el fichero de análisis.

Después de crear el fichero de análisis se puede ir a la transacción de análisis para estudiar dicho fichero.
SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Ejemplo:

```
DO 2 TIMES.
  IF SY-UNAME = 'SMITH'.
    CALL FUNCTION 'S_ABAP_TRACE_OPEN_FILE'.
    SET RUN TIME ANALYZER ON.
  ENDIF.
* >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
* The modularization unit to be measured is called
* here.
* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
  IF SY-UNAME = 'SMITH'.
    SET RUN TIME ANALYZER OFF.
  ENDIF.
ENDDO.
```

Véase también: GET PROPERTY, CALL METHOD, CREATE OBJECT.

SET SCREEN

Definición
Determina cual va ser la dynpro siguiente en procesarse.
Sintaxis:
SET SCREEN <dynpro>.
<dynpro> determina el número del dynpro que será procesado cuando en dynpro actual se haya procesado completamente. Para termina el proceso de un dynpro se puede utilizar la sentencia SET SCREEN con el valor 0 o la sentencia LEAVE SCREEN. Un proceso también termina cuando se han ejecutado todos los módulos del proceso lógico del dynpro.
Véase también: LEAVE SCREEN, LEAVE TO SCEEN.

SET TITLEBAR

Definición
Define un título para la pantalla de salida.
Sintaxis:
SET TITLEBAR <título> [WITH <var1> <var2> <var3> <var4>].
Al utilizar esta sentencia sustituimos el título estándar por el título definido en <título>. El título permanece activo en todas las pantallas de salida (listado básico y listado secundarios) mientras no volvamos a utilizar esta sentencia. <título> es un campo de tres posiciones que hace referencia al título definido en la transacción *Menu painter*. El significado de las cláusulas es el siguiente:
WITH -> Se puede añadir al título de la pantalla hasta cuatro variables. Las variables se definen en el texto del título, en la transacción *Menu painter*, con un *ampersand* (&) y un número, del 1 al 4. Las variables definidas en el texto del título pero no especificadas en la sentencia SET TITLEBAR se sustituyen por espacios en blanco. El texto del título puede tener hasta setenta caracteres.

SET UPDATE TASK LOCAL

Definición
Le dice al sistema que las actualizaciones se deben realizar localmente.
Sintaxis:
SET UPDATE TASK LOCAL.

SET USER-COMMAND

Definición
Permite activar un evento desde el programa.
Sintaxis:
SET USER-COMMAND <comando>.
Esta sentencia tiene efecto después de que el actual listado, básico o secundario, haya sido completado. Si utilizamos sentencias SET USER-COMMAND durante la generación de un mismo listado, sólo tendrá efecto el último listado.
Ejemplo:
WRITE: 'List'... "Create a list
SET CURSOR LINE 7.
SET USER-COMMAND 'PICK'.
El evento AT LINE-SELECTION es procesado para la línea siete.

SHIFT

Definición
Se utiliza para desplazar el contenido de un campo alfanumérico. Disponemos de distintas variantes en función del modo de desplazamiento.
Sintaxis:
SHIFT <campo> [BY <n> PLACES] [{ LEFT | RIGHT | CIRCULAR }].
Desplaza el contenido del campo <campo> <n> posiciones.El significado de las cláusulas es el siguiente:

- BY -> Si se omite, el desplazamiento es de una posición. Si <n> es cero o negativo no se realiza ningún desplazamiento. Si <n> excede de la longitud del campo <campo>, el desplazamiento se rellena con espacios en blanco. <n> puede ser una variable.
- LEFT -> el desplazamiento se realiza hacia la izquierda, con relleno de espacios en blanco por la derecha.
- RIGHT -> El desplazamiento se realiza hacia la derecha, con relleno de espacios en blanco por la izquierda.
- CIRCULAR -> El desplazamiento se realiza hacía la izquierda. Los caracteres que desaparecen por la izquierda aparecen por la derecha.

SHIFT <campo> UP TO <string> [{ LEFT | RIGHT | CIRCULAR }].
Realiza un desplazamiento del contenido de un campo hasta un string dado. El sistema busca el string <string> en el campo <campo>. Si el string está contenido en el campo <campo> se realizará el desplazamiento hasta alcanzar el string. De no contener <campo> el string <string> no se realizará ningún desplazamiento. <string> puede ser una variable. El modo de desplazamiento es el mismo que en la variante anterior. Si el string se encuentra en el campo <campo>, SY-SUBRC es 0, en caso contrario es 4.
SHIFT <campo> LEFT DELETING LEADING <string>.
SHIFT <campo> RIGHT DELETING LEADING <string>.
Realiza un desplazamiento hasta que el primer o último carácter cumpla cierto criterio. Esta sentencia desplaza el contenido del campo <campo> hasta que el primer carácter de la izquierda (primera sentencia) o el último carácter de la derecha (segunda sentencia) satisface cierta condición. El desplazamiento se rellena con espacios en blanco. <string> puede ser una variable.

Ejemplo 1:
DATA ALPHABET(10) VALUE 'ABCDEFGHIJ'.
SHIFT ALPHABET CIRCULAR.
ALPHABET contendría 'BCDEFGHIJA' .
Ejemplo 2:
DATA ALPHABET(10) VALUE 'ABCDEFGHIJ',
FIVE TYPE I VALUE 5.
SHIFT ALPHABET BY FIVE PLACES.
ALPHABET contendría 'FGHIJ' .
Ejemplo 3:
DATA ALPHABET(10) VALUE 'ABCDEFGHIJ',
THREE(3) VALUE 'DEF',
FOUR(4) VALUE 'DEF '.
SHIFT ALPHABET UP TO THREE CIRCULAR.
ALPHABET contendría 'DEFGHIJABC' .
Véase también: CONCATENATE, SHIFT, SPLIT.

SKIP

Definición
Genera líneas en blanco en el dispositivo de salida.
Sintaxis:
SKIP { [<n>] | TO LINE <n> }.
Con la primera opción el sistema salta <n> líneas en blanco. Si <n> no se especifica sólo se salta una línea.
Con la segunda opción situamos la posición de salida en la línea indicada con <n>. Con esta opción se puede ir a líneas anteriores como posteriores, además se actualiza la variable del sistema SY-LINNO. Si <n> no está comprendido entre 1 y la longitud de la página, el sistema ignora la sentencia. También se tienen en cuenta la cabecera y el pie de página (no hay que olvidar este detalle).
Ejemplo:
REPORT TEST NO STANDARD PAGE HEADING.
DATA: ROW TYPE I VALUE 3.
WRITE 'Line 1'.
SKIP TO LINE 5.
WRITE 'Line 5'.
SKIP TO LINE ROW
WRITE 'Line 3'.
Véase también: NEW-LINE.

SORT

Definición
Ordena una tabla interna.
Sintaxis:
SORT <tabla> [<orden>][AS TEXT]
[BY <campo1> [<orden>][AS TEXT] ... <campon> [<orden>][AS TEXT]].
El significado de las cláusulas es el siguiente:

- BY -> Si no la utilizamos la tabla interna es ordenada por la clave estándar. Para definir un criterio de ordenación distinto a la clave estándar se utiliza la cláusula BY- Con la cláusula BY los campos se ordenarán de acuerdo con los componentes especificados <campo1> ... <campon>. Estos componentes puede ser de cualquier tipo (incluido tablas). El número de campos que se pueden ordenar está restringido a 250. El sistema utiliza las opciones

utilizadas antes de BY (<orden> y AS TEXT) como valores por defecto para los campos especificados en BY. Se pueden especificar el criterio de ordenación en tiempo de ejecución, utilizando (<nombre>) en lugar de <campo>. La variable <nombre> contiene el nombre del componente de la tabla interna. Si la variable <nombre> está vacía, el sistema ignorará el criterio de ordenación, si contiene un nombre de componente inválido, producirá un error en tiempo de ejecución. Para cualquier campo, componente de una tabla interna, se puede indicar un offset (posición) y longitud. En <orden> se puede especificar cómo se debe ordenar la tabla, de forma descendente, DESCENDING, o de formas ascendente, ASCENDING,. Por defecto el orden es ascendente.

- AS TEXT -> Influye en el método de ordenación para campos alfanuméricos. Sin esta opción, el sistema ordena los campos binariamente, de acuerdo con el sistema donde tengamos el sistema SAP R/3. Con la opción AS TEXT, el sistema ordena los campos alfanuméricos de forma alfabética. Si especificamos AS TEXT antes de la cláusula BY, sólo afectará a los campo de tipo C. Si lo especificamos después de la cláusula BY sólo podrá ser utilizada en campos de tipo C, con lo cual la ordenación no es estable. Esto quiere decir que líneas con la misma clave de ordenación no tienen por qué ser ordenadas de la misma forma en distintas ejecuciones. Si no hay suficiente espacio en memoria para ordenar la tabla, el sistema escribe los datos en un fichero externo temporal. El nombre de ese fichero está especificado en el parámetro DIR_SORTTMP del perfil de arranque de SAP.

Ejemplo:

```
DATA: ONR(7), DATE TYPE D, POSITION(3) TYPE N,
      CUSTOMER(16),
      PNR(5) TYPE N, NAME(10), UNITS TYPE I,
      ORDERS TYPE I.
FIELD-GROUPS: HEADER, ORDER, PRODUCT, DATE_FIRST.
INSERT ONR DATE POSITION INTO HEADER.
INSERT CUSTOMER          INTO ORDER.
INSERT PNR NAME UNITS    INTO PRODUCT.
INSERT DATE ONR POSITION INTO DATE_FIRST.
ONR = 'GF00012'. DATE = '19921224'.
POSITION = '000'. CUSTOMER = 'Good friend (2.)'.
EXTRACT ORDER.
ADD 1 TO POSITION.
PNR = '12345'. NAME = 'Screw'.  UNITS = 100.
EXTRACT PRODUCT.
ADD 1 TO POSITION.
PNR = '23456'. NAME = 'Nail'.   UNITS = 200.
EXTRACT PRODUCT.
ONR = 'MM00034'. DATE = '19920401'.
POSITION = '000'. CUSTOMER = 'Moneymaker'.
EXTRACT ORDER.
ADD 1 TO POSITION.
PNR = '23456'. NAME = 'Nail'.   UNITS = 300.
EXTRACT PRODUCT.
ADD 1 TO POSITION.
PNR = '34567'. NAME = 'Hammer'. UNITS = 4.
EXTRACT PRODUCT.
ONR = 'GF00011'. DATE = '19921224'.
POSITION = '000'. CUSTOMER = 'Good friend (1.)'.
EXTRACT ORDER.
ADD 1 TO POSITION.
PNR = '34567'. NAME = 'Hammer'. UNITS = 5.
EXTRACT PRODUCT.
SORT BY DATE_FIRST.
LOOP.
```

```
  AT ORDER.
    WRITE: /, / DATE, ONR, POSITION,
             CUSTOMER, 'ordered:'.
  ENDAT.
  AT PRODUCT.
    WRITE: /    DATE, ONR, POSITION,
             PNR, NAME, UNITS.
  ENDAT.
ENDLOOP.
```

El resultado en pantalla sería:

```
01041992 MM00034 000 Moneymaker ordered:
01041992 MM00034 001 23456 Nail 300
01041992 MM00034 002 34567 Hammer 4
24121992 GF00011 000 Good friend (1.) ordered:
24121992 GF00011 001 34567 Hammer 5
24121992 GF00012 000 Good friend (2.) ordered:
24121992 GF00012 001 12345 Screw 100
24121992 GF00012 002 23456 Nail 200
```

SPLIT

Definición

Divide un campo alfanumérico en varios campos, gracias a algún separador.

Sintaxis:

SPLIT <campo> AT <delimitador> INTO <c1> ... <cn>.

Esta sentencia utiliza el campo indicado en <delimitador> para separar los campos <c1> ... <cn> el contenido del campo <campo>. Si no hay especificado los suficientes campos para poder separar todo el contenido del campo <campo>, sobre el último se rellena el resto del campo <campo>. Si todos los campos destino son lo suficientemente grandes como para almacenar las partes de <campo>, SY-SUBRC vale 0. En caso contrario SY-SUBRC vale 4.

También se puede situar las partes del campo que se quiere separar en una tabla interna con el siguiente formato:

SPLIT <campo> AT <delimitador> INTO TABLE <tabla>.

Por cada parte del campo <campo el sistema añade una nueva línea en la tabla interna <tabla>.

Ejemplo 1:

```
DATA: NAMES(30)    VALUE 'Charly, John, Peter',
      ONE(10),
      TWO(10),
      DELIMITER(2) VALUE ','.
SPLIT NAMES AT DELIMITER INTO ONE TWO.
```

ONE valdrá "Charly" y TWO tendrá el valor "John, Pete".

Ejemplo 2:

```
DATA: BEGIN OF ITAB OCCURS 10,
        WORD(20),
      END   OF ITAB.
SPLIT 'STOP Two STOP Three STOP   ' AT 'STOP' INTO TABLE ITAB.
```

La tabla interna tendrá tres líneas, la primera en blanco, la segunda contiene "Two" y la tercera contiene "Three".

Véase también: SHIFT, CONCATENATE, SEARCH.

START-OF-SELECTION

Definición
Este evento nos permite crear un bloque de proceso después de procesar la pantalla de selección y antes del evento GET.
Sintaxis:
START-OF-SELECTION.
Se puede utilizar para cargar cierta información después de procesar la pantalla de selección y antes del evento GET.
Véase también: INITIALIZATION, END-OF-SELECTION.

STATICS

Definición
Declaración de objetos de datos internos a un procedimiento (subrutina o módulo de función) con valores estáticos, es decir, el valor quedará retenido en el procedimiento. Si llamamos varias veces al procedimiento, el valor del objeto de dato permanecerá sin cambios de una llamada a otra.
Sintaxis:
STATICS <campo> [(<longitud>)] [<tipo>] [<valor>] [<decimales>].
Declaración de una variable. Para comprender el significado de cada parámetro ver la sentencia DATA.
STATICS: BEGIN OF <registro>.
...
END OF <registro>.
Declaración de un registro o *field-string*. Para comprender el significado de cada parámetro ver la sentencia DATA.
STATICS: BEGIN OF <tabla> OCCURS <n>.
...
END OF <tabla>.
Declaración de un registro o tabla interna. Para comprender el significado de cada parámetro ver la sentencia DATA.
Véase también: DATA.

STOP

Definición
Permite abandonar cualquier bloque de proceso e ir directo al bloque de proceso del evento END-OF-SELECTION. El abandono se realiza de forma incondicional.
Sintaxis:
STOP.
Véase también: EXIT, REJECT, CHECK.

SUBMIT

Definición
Ejecuta un programa.
Sintaxis:
SUBMIT { <programa> | (<programa>) } [LINE-SIZE <columnas>] [LINE-COUNT <filas>]
[TO SAP-POOL
[DESTINATION <destino>]
[COPIES <copias>]

[LIST NAME <nombre>]
[COVER TEXT <texto>]
[LIST AUTHORITY <autorización>]
[INMEDIATELY <flag>]
[KEEP IN SPOOL <flag>]
[NEW LIST IDENTIFICATION <flag>]
[DATASET EXPIRATION]
[LINE-COUNT <líneas>]
[LINE-SIZE <columnas>]
[LAYOUT <layout>]
[SAP COVER PAGE <modo>]
[COVER PAGE <flag>]
[RECEIVER <receptor>]
[DEPARTMENT <departamento>]
[ARCHIVE MODE <modo>]
[ARCHIVE PARAMETERS <parámetros>]
[WITHOUT SPOOL DYNPRO]]
[SPOOL PARAMETERS <parámetros>]
[ARCHIVE PARAMETERS <parámetros>]
[WITHOUT SPOOL DYNPRO]
[VIA SELECTION-SCREEN]
[AND RETURN]
[EXPORTING LIST TO MEMORY]
[USER <usuario> VIA JOB <job> NUMBER <número>]
[USING SELECTION-SETS OF PROGRAM <programa1>].

Esta sentencia llama al REPORT <programa>. El nombre del programa se puede especificar dinámicamente utilizando una variable entre paréntesis. El significado de las cláusulas es el siguiente:

- LINE-SIZE -> El programa se visualiza con el número de columnas definido en <columnas>.
- LINE-COUNT -> El programa se visualiza con el número de líneas definido en <filas>.
- TOP SAP-SPOOL -> Especifica los parámetros de impresión. El significado de las cláusulas es el siguiente:
  - DESTINATION -> Dispositivo de salida.
  - COPIES -> Número de copias.
  - LIST NAME -> Nombre del listado.
  - LIST DATASET -> Nombre del *spool dataset*.
  - COVER TEXT -> Título de la cubierta.
  - LIST AUTHORITY -> Autorización de display.
  - INMEDIATELY -> Impresión inmediata.
  - KEEP IN SPOOL. Guardar el listado después de la impresión.
  - NEW LIST IDENTIFICATION -> Identificación del nuevo listado.
  - DATASET EXPIRACION -> Fecha de expiración.
  - LINE-COUNT -> Líneas por página.
  - LINE-SIZE -> Columnas del informe.
  - LAYOUT -> Formato de impresión.
  - SAP COVER PAGE -> Con carátula de impresión.
  - COVER PAGE -> Selección de cubierta.
  - RECEIVER -> Usuario receptor.
  - DEPARTMENT -> Nombre del departamento.
  - ARCHIVE MODE -> Modo de archivo.

    - ARCHIVE PARAMETERS -> Parámetros de archivo.
    - WITHOUT SPOOL DYNPRO -> Salta la pantalla de control de impresión.
    - SPOOL PARAMETERS -> Parámetros de impresión.
- VIA SELECTION-SCREEN -> Aparece al usuario la pantalla de selección.
- AND RETURN -> El programa llamador permanece en espera mientras se ejecuta el programa llamado. Cuando el programa llamado termina se devuelve control al programa llamador. Para ello se crea una sesión interna.
- EXPORTING LIST TO MEMORY -> El listado de salida del programa llamado no aparece en pantalla. En su lugar se guarda en memoria. El programa llamador puede leer el listado de la memoria. Esta cláusula no puede ser utilizada en combinación con la cláusula TO SAP-SPOOL.
- USER <usuario> VIA JOB <job> NUMER <n> -> Se utiliza para incluir un paso de job en un proceso de background.
- USING SELECTION-SETS OF PROGRAM -> El programa llamado utiliza las variantes del programa <programa1>. Si los parámetros y criterios de selección de ambos programas no coinciden, las variables de <programa1> se pueden borrar.

Los mensajes de error que se pueden producir son los siguientes:

- LOAD_PROGRAM_NOT_FOUND -> El programa específico no existe.
- SUBMIT_WRONG_TYPE -> El programa especificado no es un report.
- SUBMIT_IMPORT_ONLY_PARAMETER -> Valor inválido pasado por parámetro.
- SUBMIT_WRONG_SIGN -> Valor inválido pasado a un criterio de selección.
- SUBMIT_IN_ITAB_ILL_STRUCTURE -> Tabla pasado como criterio con estructure errónea.

La sentencia *submit*, con una determina cláusula también nos permite añadir pasos a un job creado con el módulo de función JOB_OPEN.

```
SUBMIT <report> AND RETURN
   USER <usuario>
   VIA JOB <nombre_job> NUMBER <id_job>
   TO SAP-SPOOL SPOOL PARAMETERS <parámetros_impresión>
                ARCHIVE PARAMETERS <parámetro_archivo>
                WITHOUT SPOOL DYNPRO.
```

SUBTRACT

Definición

Resta el contenido de dos campos.

Sintaxis:

SUBTRACT <n> FROM <m>

Resta el contenido de dos estructuras.

Ejemplo:

```
DATA NUMBER TYPE P VALUE 3,
    RESULT TYPE I VALUE 7.
SUBTRACT NUMBER FROM RESULT.
```

RESULT valdrá 4 y NUMBER valdrá 3.

Véase también: COMPUTE, SUBTRACT-CORRESPONDING.

SUBTRACT-CORRESPONDING

Definición
Substrae el contenido de campos de una estructura.
Sintaxis:
SUBTRACT-CORRESPONDING <n> FROM <m>.
Substrae el contenido de los componentes del registro <n> del registro <m>, para aquellos componentes que se llamen igual.
Ejemplo:

```
DATA: BEGIN OF PERSON,
       NAME(20)     VALUE 'Paul',
       MONEY TYPE I VALUE 5000,
      END   OF PERSON,
      BEGIN OF PURCHASES OCCURS 10,
        PRODUCT(10),
        MONEY TYPE I,
      END   OF PURCHASES.
PURCHASES-PRODUCT = 'Table'.
PURCHASES-MONEY = 100.
APPEND PURCHASES.
PURCHASES-PRODUCT = 'Chair'.
PURCHASES-MONEY =  70.
APPEND PURCHASES.
LOOP AT PURCHASES.
  SUBTRACT-CORRESPONDING PURCHASES FROM PERSON.
ENDLOOP.
```

El campo PERSON-MONEY valdrá 4830
Véase también: SUBTRACT, ADD-CORRESPONDING, MULTIPLY-CORRESPONDING, DIVIDE-CORRESPONDING.

SUM

Definición
La sentencia SUM sólo se puede utilizar en el bucle LOOP..ENDLOOP. Con ella, el sistema suma todos aquellos campos numéricos, componentes de la tabla interna, dejando el resultado sobre el área de trabajo de la tabla interna.
Sintaxis:
SUM.
Si la sentencia SUM puede ser utilizada en la sentencia AT .. ENDAT. En este caso, los campos a sumar serán:

- FIRST -> Suman todas las líneas.
- LAST -> Suman todas las líneas.
- NEW <campo> -> Suman todas las línea para el valor del campo que activa la sentencia.
- END OF <campo> -> Suman todas las línea para el valor del campo que activa la sentencia.

Si algún componente de la tabla interna es a su vez una tabla interna, la sentencia SUM no puede ser utilizada.

SUMMARY

Definición
Se utiliza para resaltar las líneas de salida.
Sintaxis:
SUMMARY.
Después de esta sentencia, todas las sentencias de escritura tienen alta intensidad.
Véase también: FORMAT.

SUPRESS DIALOG

Definición
Suspende la presentación de una pantalla en el procesamiento de un dynpro. Esta sentencia hace que el sistema no muestre la pantalla.
Sintaxis:
SUPRESS DIALOG.
El flujo de proceso continua en el proceso PAI. Esta sentencia debería ser utilizada en el proceso PBO.
Lógicamente esta sentencia sólo tiene sentido en el proceso PBO del dynpro.

SYNTAX-CHECK

Definición
Nos permite verificar la sintaxis de un programa.
Sintaxis:
SYNTAX-CHECK FOR <tabla> MESSAGE <f> LINE <g> WORD <h>
[PROGRAM <f1>][INCLUDE <f2>] [OFFSET <f3>] [TRACE-TABLE <t1>]
[DIRECTORY ENTRY <f4>] [REPLACING <f5>] [FRAME ENTRY <f6>] [MESSAGE-ID <f7>]
[ID <id> TABLE <tab>].
El código del programa está contenido en <tabla>. Si la sentencia detecta algún error durante la comprobación sintáctica, los campos <f>, <g> y <h> son rellenados con la siguiente información:

- <f> -> Contiene el texto del mensaje de error. Tipo alfanumérico.
- <g> -> Contiene la línea del programa donde aparece el error. Tipo numérico.
- <h> -> Contiene la palabra con el error. Tipo alfanumérico.

La variable SY-SUBRC puede tomar los siguiente valores:

- 0 -> El programa no tiene errores de sintaxis.
- 4 -> El programa contiene errores de sintaxis.
- 8 -> El programa tiene errores que no son de sintaxis.

El significado de las cláusula es el siguiente:

- PROGRAM -> Especifica el nombre del programa en <f1>. Si no se utiliza la cláusula DIRECTORY ENTRY, el nombre del programa se utiliza para determinar los atributos, por ejemplo, si se trata de un programa, un include o una base de datos lógica. El campo <f1> debe tener el formato de la variable del sistema SY-REPID.
- INCLUDE <f2> -> Si existe un error y este aparece en un include, <f2> contiene el nombre del programa include. El campo <f1> debe tener el formato de la variable del sistema SY-REPID.
- OFFSET -> Si existe un error, <f3> contiene la posición de la palabra incorrecta en la línea incorrecta. <f3> debe ser declarado de tipo entero.

- TRACE-TABLE -> La traza de salida se almacena en la tabla <t1>. Para activar o desactivar la traza durante las comprobaciones sintácticas del programa se utiliza la sentencia SYNTAX-TRACE.
- DIRECTORY ENTRY -> Los atributos del programa requerido para la comprobación sintáctica del programa se especifican en el campo <f4>, el cual debe tener la estructura de la tabla TRDIR.
- REPLACING -> Si el programa a validar contiene un include con el nombre especificado en <f5>, el contenido de la tabla interna deber ser tomado.
- FRAME ENTRY -> Los atributos del programa principal se especifican en el campo <f6>, que debe tener la estructura de la tabla TRDIR.
- MESSAGE-ID -> Si ocurre un error, el campo <f7> contiene la clave del mensaje relevante. <f7> debe tener la estructura de la tabla TRMSG.
- ID <id> TABLE <tabla> -> Devuelve información de la comprobación sintáctica. No debe ser utilizado por usuarios. <id> contiene el tipo de información que será escrito en la tabla interna <tab>. <id> puede contener los siguientes valores:
  - MSG -> Mensajes warnings.
  - CORR -> Correciones.
  - SYMB -> Dumps.
  - DATA -> Objetos del programa.
  - DPAR -> Parámetros del programa.
  - TYPE -> Tipos de datos del programa.
  - FOTY -> Tipos de datos utilizados en subrutinas.
  - FUTY -> Tipos de datos utilizados en módulos de función.
  - TYCH -> Componentes de tipos de objetos.
  - CROS -> Objetos de datos referenciados.
  - STR -> Identificadores.
  - FORM -> Rutinas FORM.
  - FPAR -> Parámetros FORM.
  - PERF -> Llamadas a rutinas (PERFORM).
  - APAR -> Parámetros PERFORM.
  - FUNC -> Módulos de función.
  - FFPA -> Parámetros de módulos de función.
  - CALL -> Llamadas a CALL FUNCTION.
  - FAPA -> Parámetros CALL FUNCTION.
  - HYPH -> Objetos de datos con guiones en el nombre.
  - INCL -> Include del programa.

SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Véase también: SYNTAX-CHECK FOR DYNPRO.

SYNTAX-CHECK FOR DYNPRO

Definición
Comprueba la sintaxis de una dynpro.
Sintaxis:
SYNTAX-CHECK FOR DYNPRO <h> <f> <e> <m> MESSAGE <c1> LINE <g> WORD <c2> WORD <c3>[OFFSET <c4>][TRACE-TABLA <tabla>].

Toda la información necesaria para validar el dynpro se toma del campa <h> y de las tablas internas <f>, <e> y <m>. Este campo y tablas internas tiene la misma estructura y significado que el de la sentencia IMPORT DYNPRO. Si ocurre un error de sintaxis el campo <c1> se rellena con el texto del mensaje, <c2> se rellena con la línea de la pantalla donde ocurre el error y <c3> contiene la palabra con el error. La variable del sistema SY-SUBRC toma el valor 0 su no existen errores, en caso contrario valdrá 4. El significado de las cláusulas es el siguiente:

- OFFSET -> Si ocurre un error, el campo <c4> contiene la posición de la palabra incorrecta.
- TRACE-TABLE -> Cualquier salida del trazador se deposita en la tabla <tabla>.

SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.
Véase también: SYNTAX-CHECK.

SYNTAX-TRACE

Definición
Activa / desactiva el trazador sintáctico.
Sintaxis:
SYNTAX-TRACE { ON [OPTION { CODING | EXPAND }] | OFF }.
El significado de las cláusulas es el siguiente:

- ON -> Se activa el trazador sintáctico.
- OFF ->Se desactiva el trazador sintáctico.
- OPTION CODING -> Sólo se activa en el programa actual.
- OPTION EXPAND -> No tiene ningún efecto.

SAP creó esta sentencia para uso interno. Se puede utilizar pero hay que tener en cuenta que SAP puede cambiar o eliminar la sintaxis sin previo aviso.

TABLES

Definición
Hace que una tabla de la base de datos, una vista o una estructura del diccionario de datos sea reconocida por un programa.
Sintaxis:
`TABLES <objeto-bd>.`
<objeto-bd> es el nombre del objeto del diccionario de datos. Además el sistema genera un registro con la misma estructura que la definida en el diccionario de datos que podrá ser utilizada en el programa., por ejemplo, para realizar operaciones de lectura/escritura sobre la base de datos. La secuencia y los nombre de los componentes del registro (comúnmente llamado área de trabajo) de la tabla son los mismos que los definidos en el objeto en el diccionario de datos. Para referenciar un componente de un área de trabajo de una tabla, vista o estructura del diccionario de datos se utiliza la siguiente sintaxis:
`<objeto-bd>-<componente>`
La correspondencia de tipos de datos entre el diccionario de datos y el programa es la siguiente:

Tipo de dato en diccionario de datos	**Tipo de dato en programa**
ACCP	N(6)
CHAR n	C(n)
CLNT	C(3)
CUKY	C(5)
CURR n, m, s	P((n+2)/2) DECIMALS m [NO-SIGN]
DEC n,m, s	P((n+2)/2) DECIMALS m [NO-SIGN]
DATS	D

FLTP	F
INT1	Sin correspondencia
INT2	Sin correspondencia
INT4	I
LCHR n	C(n)
LRAW n	X(N)
LANG	X(1)
NUMC n	N(n)
PREC	X(2)
QUAN n, m, s	P((n+2)/2) DECIMALS m [NO-SIGN]
RAW n	X(n)
UNIT n	C(n)
VARC c	C(n)

TOP-OF-PAGE

Definición

Este evento define un bloque de proceso que se activa cuando el sistema detecta que vamos a comenzar a escribir en la página actual. El número de líneas por página se define en la sentencia REPORT. Se suele utilizar este evento para componer cabeceras de páginas.

Sintaxis:

```
TOP-OF-PAGE.
```

En los listados secundarios el sistema no muestra la cabecera de página estándar, ni tampoco activa el evento TOP-OF-PAGE. Para crear cabeceras de página en los listados secundarios se utiliza el siguiente evento:

```
TOP-OF-PAGE DURING LINE-SELECTION.
```

El sistema activa este evento en todos los listados secundarios. Si queremos crear diferentes cabeceras de página para los diferentes listados secundarios debemos programar el bloque de proceso del evento de acuerdo, por ejemplo, con las variables del sistema SY-LSIND o SY-PFKEY en combinación con las sentencias de control IF o CASE. Cuando nos desplazamos verticalmente en un listado secundario, la cabecera permanece fija.

Ejemplo:

```
PROGRAM DOCUEXAM NO STANDARD PAGE HEADING.
START-OF-SELECTION.
  WRITE: / 'line 1'.
  WRITE: / 'line 2'.
  WRITE: / 'line 3'.
TOP-OF-PAGE.
  WRITE: / 'Heading'.
  ULINE.
```

La salida en pantalla sería:

```
Heading
------------------------------
line 1
line 2
line 3
```

Véase también: END-OF-PAGE, AT LINE-SELECTION, AT USER-COMMAND.

TRANSFER

Definición
Escribe registros sobre un fichero en el servidor de aplicación.
Sintaxis:
```
TRANSFER <campo> TO <fichero> [ LENGTH <longitud> ].
```
Escribe el valor del campo <campo> en el fichero <fichero>. El modo de transferencia se especifica en la sentencia OPEN DATASET. Si el fichero no esta abierto para escritura, el sistema intenta abrirlo en modo binario o con las opciones de la última sentencia OPEN DATASET para ese fichero, sin embargo, es recomendable abrir siempre el fichero con la sentencia OPEN DATASET. <fichero> puede ser un literal o un campo como en la sentencia OPEN DATASET. <campo> pude ser de un tipo elemental, o un string que no contenga tablas internas como componentes. Las tablas internas no se puede escribir directamente sobre ficheros.
Con la opción LENGHT se puede especificar la longitud del datos que va a ser transferido. El sistema transfiere los primeros <longitud> bytes sobre el fichero. Si el campo <longitud> es más pequeño que el campo, se trunca. Si <longitud> es más grande que el campo, el campo se rellena con espacios en blanco.
Véase también: OPEN DATASET, CLOSE DATASET.

TRANSLATE

Definición
Se utiliza para convertir caracteres de mayúscula a minúscula o viceversa, o para realizar sustituciones de caracteres a través de ciertas reglas.
Sintaxis:
```
TRANSLATE <campo> TO  { UPPER | LOWE } CASE.
```
Con la opción UPPER, las letras minúsculas de <campo> se convierten en mayúsculas. Y con la opción LOWER las letras mayúsculas se convierten en minúsculas.
Para utilizar reglas de conversión se utiliza la siguiente sintaxis.
```
TRANSLATE <campo> USING <regla>.
```
Esta sentencia reemplaza todos los caracteres de <campo> que cumplan la regla de sustitución <regla>. La regla de sustitución contiene pares de letras, la primera de ellas indica el carácter a sustituir, la segunda indica el carácter de sustitución. <regla> puede ser una variable.
Ejemplo:
```
DATA: LETTERS(10) VALUE 'abcX',
      CHANGE(6)   VALUE 'aXBY'.
TRANSLATE LETTERS USING CHANGE.
```
El campo LETTERS contendrá 'XbcX' .
Véase también: REPLACE, OVERLAY.

TYPE-POOL

Definición
Introduce la definición de un *type-groups*.
Sintaxis:
```
TYPE-POOL <nombre>.
```
Un *type-groups* en el diccionario de datos no es más que un fragmento de programa, compuesto de declaración de tipo o constantes. La primera sentencia que debe aparecer es TYPE-POOL. El nombre de todos los tipos o constantes debe comenzar con el nombre del *type-groups* y un guión bajo (_).
Ejemplo:
```
TYPE-POOL ABCDE.
TYPES: ABCDE_PACKED TYPE P,
       ABCDE_INT    TYPE I.
```
Véase también: TYPE-POOLS.

TYPE-POOLS

Definición
Para utilizar un *type-groups* en un programa se utiliza la sentencia TYPE-POOLS.
Sintaxis:
```
TYPE-POOLS <nombre>.
```
La variable <nombre> identifica el *type-groups* a utilizar. En un programa se pueden utilizar varios *type-groups*.
Ejemplo:
```
TYPE-POOLS VERI1.
DATA X TYPE VERI1_TYP1.
```
Véase también: TYPE-POOL.

TYPES

Definición
Declaración de tipos de datos.
Sintaxis:
```
TYPES <tipo> [ <longitud> ] <tipo> [ <decimales> ].
```
Los parámetros de estas sentencias son los mismos que los utilizados en la sentencia DATA. El único parámetro no se utiliza es <valor> ya que la sentencia TYPES no tiene memoria asociada, y por lo tanto no se le puede asignar ningún valor.
```
TYPES: BEGIN OF <registro>,
              ...
       END OF <registro>.
```
Declaración de un tipo de datos con estructura de registro.
```
TYPES <tabla> <tipo> OCCURS <n>.
```
Para crear un tipo de dato de tabla interna es necesario especificar la cláusula OCCURS en la sentencia TYPES. Las líneas de la tabla interna tendrán el tipo de dato especificado en <tipo>. En <tipo> se puede especificar las cláusulas TYPE o LIKE.
Con la cláusula LIKE se puede referenciar una estructura de la base de datos. De esta forma se puede crear tablas internas con la misma estructura de tablas del diccionario de datos. <n> especifica el número inicial de líneas de la tabla interna. El sistema reserva memoria para el número de líneas especificado en el momento en que escribamos la primera línea sobre la tabla interna. Si añadimos más líneas a la tabla interna que las especificadas en <n>, el sistema expande el espacio reservado en memoria automáticamente. Si no hay suficiente espacio en memoria para la tabla interna, la ampliación se realizará en disco (área de paginación).
Ejemplo:
```
DATA:  BEGIN OF PERSON,
         NAME(20),
         AGE TYPE I,
       END OF PERSON.
TYPES  TYPE_PERSONS LIKE PERSON OCCURS 20.
DATA   PERSONS TYPE TYPE_PERSONS.
PERSON-NAME = 'Michael'.
PERSON-AGE  = 25.
APPEND PERSON TO PERSONS.
PERSON-NAME = 'Gabriela'.
PERSON-AGE  = 22.
APPEND PERSON TO PERSONS.
```

ULINE

Definición
Para escribir líneas horizontales se puede utilizar la sentencia ULINE.
Sintaxis:
```
ULINE [ AT ] [ / ] [ <posición> ] [ (<longitud>) ].
```
El significado de la cláusula AT es el mismo que en la sentencia WRITE. Si no se especifica la cláusula AT, el sistema escribe una nueva línea horizontal.

UNPACK

Definición
Desempaqueta el contenido de un campo sobre otro.
Sintaxis:
```
UNPACK <campo1> TO <campo2>.
```
Desempaqueta el campo <campo1> en el campo <campo2>. Es la operación contraria a la sentencia PACK.
Ejemplo:
```
DATA: P_FIELD(2) TYPE P VALUE 103,
      C_FIELD(4) TYPE C.
UNPACK P_FIELD TO C_FIELD.
```
P_FIELD: P'103C' -> C_FIELD: C'0103'.
Véase también: PACK.

UPDATE

Definición
Sentencia utilizada para modificar registros de una tabla de la base de datos.
Sintaxis:
```
UPDATE { <tabla> | (<tabla>) } [ CLIENT SPECIFIED ] [ FROM <área> ].
```
Con esta variante de la sentencia UPDATE modificamos una sola línea de la tabla especificada. El significado de la cláusula FROM es la siguiente:

- FROM -> El área de trabajo <área> sobrescribe la línea de la tabla que tenga la misma clave primaria. La tabla <tabla> debe estar declarada con la sentencias TABLES. Si no especificamos la cláusula FROM, será el contenido del área de trabajo de la propia tabla el que sobrescriba la línea de la tabla con la misma clave primaria. El área de trabajo <área>debe tener la misma longitud que el área de trabajo de la tabla. Para asegurarnos de que el área de trabajo <área> tiene la misma estructura que el área de trabajo de la tabla <tabla> se suele declarar con la sentencia DATA (o TYPE) una estructura utilizando la cláusula LIKE <tabla>.

Si la operación concluye satisfactoriamente, SY-SUBRC vale 0 y SY-DBCNT vale 1, en caso contrario SY-SUBRC vale 4 y SY-DBCNT vale 0. Se puede especificar el nombre de la tabla en tiempo de ejecución como veíamos en sentencias anteriores con la opción (<campo>). Con esta opción, la cláusula FROM es obligatoria y <área> no puede ser la propia área de trabajo de la tabla.
```
UPDATE <tabla> [ CLIENT SPECIFIED ] SET <set1> ... <setn>  [ WHERE
<condición> ].
```
Esta variante nos permite modificar una o varias líneas de la tabla <tabla> en función de las especificaciones de la cláusula WHERE. Dicha cláusula tiene las mismas opciones que las vistas para la sentencia SELECT. Si no especificamos la cláusula WHERE, todas las líneas de la tabla se cambian. <tabla> debe estar declaradas con la sentencia TABLES.

Para identificar las columnas a cambiar se utiliza la cláusula SET. La lista de comandos especificados en la cláusula SET debe respetar uno de los siguientes formatos:

- - <f> = <g>. -> <f> es un campo de la tabla <tabla>. <g> es un valor.
- - <f> = <f> + <g>. -> <f> es un campo de la tabla <tabla>. <g> es un valor. El valor de <f> es incrementado con el valor de <g>.
- - <f> = <f> - <g>. -> <f> es un campo de la tabla <tabla>. <g> es un valor. El valor de <f> es decrementado con el valor de <g>. <g> puede ser un literal o una variable.

Para ver la regla de conversión entra campos de la base de datos y campos de un programa, ver la cláusula INTO de la sentencia SELECT.
SY-DBCNT contiene el número de líneas cambiadas. SY-SUBRC es 0 si al menos una línea es cambiada y 4 si ninguna línea es cambiada. En esta variante de sentencia no se puede especificar el nombre de la tabla en tiempo de ejecución.

```
UPDATE { <tabla> | (<tabla>) } [ CLIENT SPECIFIED ] [ FROM TABLE
<tabla_interna> ].
```

Otra forma de actualizar varias líneas de una tabla es con la anterior variante de la sentencia UPDATE. Se puede definir la tabla que hay que actualizar estáticamente con la opción <tabla>, o dinámicamente con la opción (<tabla>).
Las líneas de la tabla interna sobrescriben las líneas de la tabla internas que tengan la misma clave primaria. Las líneas de la tabla interna <tabla_interna> deben tener la misma longitud que el área de trabajo de la tabla. Para asegurarnos de esto, se suele declarar la tabla interna (en el DATA o TYPE) con la misma estructura que la tabla de diccionario (cláusula LIKE).
Si el sistema no puede cambiar una línea porque la clave primaria no existe en la tabla de la base de datos, el sistema continúa con la siguiente línea de la tabla interna. Si todas las líneas se procesan satisfactoriamente, SY-SUBRC vale 0, en caso contrario vale 4. SY-DBCNT toma el valor del número de líneas actualizadas por la sentencia.
Con esta sentencia sucede lo mismo que con la sentencia INSERT. Es mucho más eficiente modificar un conjunto de líneas en una sola operación (segunda variante de la sentencia UPDATE) que modificarlas una a una (primera variante de la sentencia UPDATE).
Ejemplo 1:

```
UPDATE SCUSTOM SET: DISCOUNT  = '003',
                    TELEPHONE = '0621/444444'
               WHERE ID       = '00017777'.
```

Ejemplo 2:

```
TABLES SCUSTOM.
DATA   WA LIKE SCUSTOM.
WA-ID        = '12400177'.
WA-TELEPHONE = '06201/44889'.
UPDATE SCUSTOM FROM WA.
```

Véase también: MODIFY, INSERT.

WAIT

Definición
Sentencia utilizada para esperar la terminación de la ejecución asíncrona de un módulo de función.
Sintaxis:

```
WAIT UNTIL <expresión_lógica> [ UP TO <segundos> SECONDS ].
```

La sentencia espera a que la expresión lógica <expresión_lógica> sea TRUE. Esta sentencia sólo debe ser utilizada en combinación con la sentencia CALL FUNCTION STARTING NEW TASK <tarea>
Véase también: CALL FUNCTION STARTING NEW TASK.

WHILE .. ENDWHILE

Definición
Ejecuta un bloque de sentencias más de una vez, hasta que una condición se cumpla.
Sintaxis:

```
WHILE <condición> [ VARY <f> FROM <f1> NEXT <f2> ].
        <bloque_de_sentencias>.
ENDWHILE.
```

El sistema procesa el bloque de sentencias hasta que se cumpla la condición o hasta que el sistema procesa una sentencia EXIT, STOP o REJECT. Para condición se puede utilizar cualquier expresión lógica. El campo del sistema SY-INDEX contiene el número de veces que el bucle ha sido ejecutado. El sistema permite anidar sentencias WHILE, así como combinarlas con otras sentencias de bucle.
La opción VARY actúa de la misma forma en que actúa la opción VARYING en la sentencia DO. Al igual en la sentencia DO, se puede utilizar más de una opción VARY en una sentencia WHILE.
Ejemplo:

```
DATA: SEARCH_ME TYPE I,
      MIN       TYPE I VALUE 0,
      MAX       TYPE I VALUE 1000,
      TRIES     TYPE I,
      NUMBER    TYPE I.
SEARCH_ME = 23.
WHILE NUMBER <> SEARCH_ME.
  ADD 1 TO TRIES.
  NUMBER = ( MIN + MAX ) / 2.
  IF NUMBER > SEARCH_ME.
    MAX = NUMBER - 1.
  ELSE.
    MIN = NUMBER + 1.
  ENDIF.
ENDWHILE.
```

Véase también: CALL FUNCTION STARTING NEW TASK.

WINDOW

Definición
Esta sentencia muestra el listado secundario que se presenta en una venta en lugar de en una pantalla completa.
Sintaxis:

```
WINDOW STARTING AT <ci> <fs> [ ENDING AT <cd> <fi> ].
```

<ci> y <fs> determina la esquina superior izquierda. <ci> determina la columna izquierda y <fs> la fila superior. <cd> y <fi> determinan la esquina inferior derecha. <cd> determina la columna derecha y <fi> la fila inferior.
La cláusula ENDING es opcional, si no se utiliza, la ventana alcanza la esquina inferior derecha de la pantalla.
Ejemplo:

```
WINDOW STARTING AT 1  15
       ENDING   AT 79 23.
WRITE 'Text'.
```

WRITE

Definición
Saca datos por pantalla.
Sintaxis:
`WRITE [ AT ][ / ][ <posición> ] [ <longitud> ] <campo> [ <opciones> ].`
Esta sentencia saca el campo <campo> en su formato estándar por la salida activa. La salida activa por defecto es la pantalla. El campo <campo> puede ser cualquier objeto de dato, field-symbol o parámetro formal, o texto simbólico.
En la pantalla normalmente los campos de salida están justificados a la izquierda. Utilizando varias sentencias WRITE los campos de salida aparecen uno detrás de otro separados por un espacio en blanco. Sino hay suficiente espacio para un campo de salida en la línea actual, el sistema provoca un salto de línea.
El formato de los campos en la pantalla depende de su tipo. Las características de los campos es la siguiente:

Tipo de dato	Longitud de salida	Posicionamiento
C	Longitud del campo	Justificado a la izquierda
D	8	Justificado a la izquierda
F	22	Justificado a la derecha
I	11	Justificado a la derecha
N	Longitud del campo	Justificado a la izquierda
P	2* Longitud del campo (+1)	Justificado a la derecha
T	6	Justificado a la izquierda
X	2 * Longitud del campo	Justificado a la izquierda

Los campos numéricos F, I y P están justificados a la derecha rellenados con espacios en blanco por la izquierda. Si hay suficiente espacio se escriben los separadores de miles. Estos separados van desapareciendo en función del espacio disponible en salida. Si el campos es del tipo P y tiene posiciones decimales, la longitud de salida se incrementa en 1. Con los tipos de datos D, el formato interno del campo difiere del formato de salida. Para campos de tipo D, el sistema lee el formato de salida especificado en el registro maestro de usuario. Posibles formatos de salida son DD/MM/YY o MM/DD/YYYY, donde, DD es el día, MM es el mes y YYYY es el año.
Se puede dar formato al campo de salida con las opciones /, <offset> y <longitud>. Con / forzamos un salto de línea antes de la salida. <posición> es un número o una variable de hasta tres dígitos de longitud que sirve para indicar la posición en el dispositivo de salida. <longitud> es un número o una variable de hasta tres dígitos de longitud que sirve para indicar la longitud de salida del campo. Si <posicion> y <longitud> son literales numéricos la cláusula es AT es opcional. Si utilizamos la opción <posición>, el campo saldrá en esa posición aunque no haya espacio, o haya sido utiliza esa posición por otro campo. Si <longitud> es menor que el campo de salida, si éste es alfanumérico se trunca y si es numérico se rellena por la izquierda con un asterisco (*).
Se pueden utilizar distintas opciones de formato con <opciones>. Las opciones son las siguientes:

- LEFT-JUSTIFIED -> Salida justificada a la izquierda.
- CENTERED -> Salida centrada.
- RIGHT-JUSTIFIED -> Salida justificada a la derecha.
- UNDER <campo> -> La salida se posiciona bajo el campo <campo>.
- NO-GAP -> El espacio en blanco posterior a la salida se omite.
- USING EDIT MASK <máscara> -> Se especifica un formato de salida.
- USING NO EDIT MASK -> Desactiva el formato que puede tener un campo del diccionario de datos.
- NO-ZERO -> Si un campo tiene ceros, se reemplazan por espacios en blanco. En campos de tipo C y N, los ceros a la izquierda se reemplaza automáticamente.

Las opciones para los campo numéricos es la siguiente:

- NO-SIGN -> Salida sin signo.
- DECIMALS <decimales> -> <decimales> especifica el número de dígitos después del punto.
- EXPONENT <exponente> -> En campos de tipo F, <exponente> especifica el exponente.
- ROUND <r> -> En campos de tipo P, se multiplican por 10 elevado a a <r>.
- CURRENCY <moneda> -> Formato sobre la base de datos de la moneda <moneda> definida en la tabla TCURX.
- UNIT <unidad> -> El número de decimales se fija sobre la base de la unidad <unidad> especificada en la tabla T006 para los campos de tipo P.

Opciones para campo de tipo DD (día), MM (Mes), YYYY (Año):

- DD/MM/YY -> El separador se define en el registro maestro de usuario.
- MM/DD/YY -> El separador se define en el registro maestro de usuario.
- DD/MM/YYYY -> El separador se define en el registro maestro de usuario.
- MM/DD/YYYY -> El separador se define en el registro maestro de usuario.
- DDMMYY -> Sin separadores.
- MMDDYY -> Sin separadores.
- YYMMDD -> Sin separadores.

También puede utilizarse las mismas opciones de la sentencia FORMAT anteponiendo la cláusula COLOR. El detalle de opciones se verá con dicha sentencia.
También tenemos las siguientes opciones de la sentencia WRITE:

- WRITE <symbol> AS SYMBOL -> Si escribe el gráfico <símbolo>. Para poder utilizar esta opción es necesario incluir en el programa
- WRITE <icono> AS ICON -> Se escribe el icono <icono>. Para poder utilizar esta opción es necesario incluir en el programa la siguiente sentencia: INCLUDE <ICON>.
- WRITE <campo> AS CHECKBOX -> Se escribe el campo <campo> con la característica de checkbox.
- WRITE <campo> HOTSPOT -> Cuando situemos el cursor por encima del campo, éste cambiará a una mano, y con un solo click activaremos el evento AT LINE-SELECTION.

INCLUDE <SYMBOL> y INCLUDE <ICON> puede ser sustituida por INCLUDE <LIST>.

Ejemplo 1:

```
DATA: MARKFIELD(1) TYPE C VALUE 'X'.
...
WRITE MARKFIELD AS CHECKBOX.           "checkbox seleccionado
MARKFIELD = SPACE.
WRITE MARKFIELD AS CHECKBOX.           "deseleccionado
WRITE MARKFIELD AS CHECKBOX INPUT OFF. "deseleccionado, protegido
```

Ejemplo 2:

```
INCLUDE <LINE>.
ULINE /1(50).
WRITE: / SY-VLINE NO-GAP, LINE_TOP_LEFT_CORNER AS LINE.
ULINE 3(48).
WRITE: / SY-VLINE NO-GAP, SY-VLINE NO-GAP.
```

Ejemplo 3:

```
DATA: X TYPE P DECIMALS 3 VALUE '1.267',
      Y TYPE F            VALUE '125.456E2'.
WRITE: /X DECIMALS 0,  "salida: 1
       /X DECIMALS 2,  "salida: 1.27
       /X DECIMALS 5,  "salida: 1.26700
       /Y DECIMALS 1,  "salida: 1.3E+04
       /Y DECIMALS 5,  "salida: 1.25456E+04
       /Y DECIMALS 20. "salida: 1.25456000000000E+04
```

Ejemplo 4:

```
DATA HOUR TYPE P DECIMALS 3 VALUE '1.200'.
WRITE (6) HOUR UNIT 'STD'. "Salida:   1,2
HOUR = '1.230'.
WRITE (6) HOUR UNIT 'STD'. "Salida: 1,230
```

Ejemplo 5:

```
DATA TIME TYPE T VALUE '154633'.
WRITE (8) TIME USING EDIT MASK '__:__:__'.  "Salida: 15:46:33
```

Ejemplo 6:

```
DATA: FIELD(10) VALUE 'abcde'.
  WRITE: / '|' NO-GAP, FIELD LEFT-JUSTIFIED  NO-GAP, '|',
         / '|' NO-GAP, FIELD CENTERED        NO-GAP, '|',
         / '|' NO-GAP, FIELD RIGHT-JUSTIFIED NO-GAP, '|'.
* salida: |abcde     |
*          |  abcde   |
*          |     abcde|
```

Ejemplo 7:

```
INCLUDE <SYMBOL>.
WRITE: / SYM_RIGHT_HAND AS SYMBOL,    " salida de un símbolo
         'Tip, Note',
         SYM_LEFT_HAND  AS SYMBOL.    " salida de un símbolo
```

Véase también: WRITE TO.

WRITE TO

Definición

Cuando asignamos valores a objetos de datos con la sentencia, se puede utilizar las opciones de la sentencia WRITE a excepción de UNDER y NO-GAP.

Sintaxis:

```
WRITE { <c1> [ +<o1> ][ <(l1)> ] | (<c1>)} TO <c2>[ +<o2> ] [ <l2>) ] [
<opciones> ].
```

La sentencia WRITE TO escribe el contenido del campo fuente <c1> sobre el campo destino <c2>. <c1> puede ser cualquier objeto de datos, <c2> sólo puede ser una variable. <c2> no puede ser un literal o una variable. El contenido de <c1> permanece sin cambios.

La sentencia WRITE TO siempre comprueba las especificaciones realizadas en el registro maestro de usuarios, por ejemplo, si el punto decimal debe aparecer con coma (,) o con punto (.). El campo destino siempre se considera como alfanumérico (tipo C). Por esta razón es conveniente que el campo destino sea siempre alfanumérico.

También se puede especificar el nombre del campo fuente en tiempo de ejecución, para ello, encerramos entre paréntesis el nombre del objeto de dato que contiene el nombre del objeto de dato que vamos a utilizar. Sin embargo, no se puede especificar en tiempo de ejecución el campo destino. Para ello deberemos utilizar *fild-symbols*.

Se puede especificar el offset y la longitud, tanto del campo fuente como del campo destino. En esta sentencia, el offset y la longitud del campo pueden ser variable.

SAP recomienda la asignación del campos con offset y longitud entre campos no numéricos. Con campos numéricos, el resultado puede ser imprevisible.

```
WRITE { <campo> [ +<o1> ] TO <tabla> [ +<o2> ] [ <l2>) ] INDEX <índice>.
```

El contenido de la sección del campo <campo> con desplazamiento de <o1> posiciones y longitud <l!> es sobrescrito en la tabla interna <tabla>, en la línea con índice <índice>, en la sección con desplazamiento de <o2> posiciones y longitud <l2>. Observemos que en esta sentencia no se utiliza en ningún caso el área de trabajo. Esto es una variante de la sentencia WRITE .. TO (utilizada para otros casos). La sentencia WRITE .. TO no reconoce la estructura de línea de la tabla interna. SAP recomienda el uso de esta sentencia sólo si queremos, por ejemplo, sustituir flag del que sabemos exactamente su posición. También puede ser utilizada para tablas con estructuras de un único campo alfanumérico; tablas con esta estructura son importantes, por ejemplo, para la generación automática de programas.

Ejemplo 1:

```
DATA: NAME(5)  VALUE 'FIELD',
      FIELD(5) VALUE 'Harry',
      DEST(18) VALUE 'Robert James Smith',
      OFF      TYPE I,
      LEN      TYPE I.
OFF = 7.
LEN = 8.
WRITE (NAME) TO DEST+OFF(LEN).
```

El campo DEST valdrá " Robert Harry ith ".

Véase también: WRITE.