I . Características del lenguaje ABAP/4

Las siglas ABAP/4 [2] provienen de: **A**dvanced **B**usiness **A**plication **P**rogramming **4**th Generation.

El ABAP/4 es un lenguaje de programación de 4ª Generación (4GL) orientado tal como su definición especifica, al desarrollo de aplicaciones de negocios.

Todos los módulos disponibles en SAP han sido programados en este lenguaje de programación.

Además podemos escribir nuevas aplicaciones en ABAP/4 como complemento a las ya existentes o como apoyo a la configuración del sistema.

Es un lenguaje estructurado orientado a **eventos**. Es decir no es un clásico lenguaje de programación con estructura lineal (TOP-DOWN), sino que la secuencia de instrucciones depende del cumplimiento de una condición o evento.

Entre las distintas aplicaciones que se pueden desarrollar con este lenguaje se encuentran:

- **Reporting** (Clásico e interactivo).
- **Programación de diálogo** o Transacciones. (Diseño de superficies CUA y diseño de pantallas).
- **Otras aplicaciones**. (*Interfaces, Batch Input*, *Formularios SAP Script*, programas de comunicaciones...etc).

Una vez instalado SAP, la principal aplicación del ABAP/4 es la generación de informes ya sea porque no han sido contemplados por SAP o por que en la instalación se requiera un informe con formato muy concreto. Así pues ABAP tendrá muchas instrucciones estinadas a facilitarnos la tarea de programar 'reports '.

Podemos diferenciar claramente entre reporting **Clásico** y reporting **Interactivo.**

El reporting clásico se caracteriza por listados voluminosos o muy frecuentes, listados pre-impresos, con mezcla de informaciones detalladas y resumidas.

El reporting interactivo se caracteriza por ser orientado a pantalla, con listados cortos y con datos resumidos. Informaciones detalladas en sublistados o ventanas controlados por teclas de función.

Tanto el reporting clásico como el interactivo se pueden ejecutar en online (tiempo real), mientras que únicamente el clásico se puede ejecutar en *Batch* (diferido).

La programación de *diálogo* (transacciones) se caracteriza por estar enfocado a pantallas (*Dynpro*) que estarán controladas por módulos ABAP/4. Tendremos un editor de pantallas *Screen Painter* y un editor de superficies *CUA Painter* o *Menú Painter***.**

Con el *Screen painter* definiremos la composición de la información que aparece en la pantalla así como la lógica de proceso para la verificación y proceso de los datos introducidos.

El *CUA painter* (Common User Acces) permite organizar los elementos de la superficie gráfica, sin necesidad de conocer el software de presentación (Windows, Motif,...). Se especificará el contenido de la barra de menús, teclas de función y menús de acción.

Otras aplicaciones posibles del lenguaje de programación son la generación de *Batch Inputs* y programas de *comunicaciones*.

Un *Batch Input* es una utilidad de SAP para transferir información de forma segura y automatizada. Para ello simula mediante un proceso *Batch* la introducción de datos en el sistema vía transacción online.

II. Entorno de desarrollo en ABAP/4

Para crear un programa ABAP debemos seguir unos pasos que detallamos a continuación.

El paso previo a trabajar con programas es mantener los atributos de éste.

Veamos esto mejor con un ejemplo:

- Ir al menú *Herramientas* → *Banco de trabajo ABAP* → *Desarrollo* → *Editor ABAP/4* o bien usar la transacción se38 para acceder al editor de programas ABAP.
- Introducir nombre del programa y pulsar Crear.
- Introducir el título del programa en la ventana que aparece.
- Indicar *Tipo de programa* (obligatorio). Generalmente un 1 (*Report*).
- Status del programa (opcional).
- Aplicación sobre la que hacemos referencia en el programa. Con un * especificamos que puede hacer referencia a cualquier aplicación.
- Clase del programa (opcional).
- Grupo de Autorizaciones con las que se puede ejecutar o editar y modificar un programa (opcional).

- Base de datos lógica (opcional).
- Aplicación de la base de datos lógica (opcional).
- Imagen de selección (opcional).
- Inicio vía variante (opcional).
- Pulsar GRABAR.

Después de introducir los atributos del programa, SAP solicita la clase de desarrollo, que es una manera de agrupar los programas funcionalmente para facilitar los métodos de corrección y transporte. Si aún no se conoce la clase de desarrollo a la que se debe asignar el programa, consideraremos provisionalmente el programa como un objeto local-privado.

Podemos ejecutar distintas funciones desde la línea de comandos (**F1** para más información), o desde los distintos menús.

También existen múltiples comandos de línea.

Con F1 sobre una instrucción obtendremos información online acerca de ésta.

Podemos grabar o recuperar programas de un dispositivo local Disco duro o disquetera (en menú utilidades).

Una vez escrito el programa podemos **verificar** que sintácticamente no tenga ningún error y antes de poderlo ejecutar tendremos que **generar**. En el proceso de generación SAP transfiere el estado del programa *Time Stamp* a diversas tablas del Dicciona rio de datos. La tabla `TRDIR` contiene información de los programas del sistema.

III .Fundamentos de la programación de reportes

Tipos de instrucciones

Un *report* consiste en una serie de instrucciones ABAP que empieza por una **palabra clave** y termina con un **punto.**

Tipos de palabras claves:

• **Declarativas** : Para declarar los datos que vamos a usar a lo largo del programa. Por ejemplo: `DATA, TABLES.`

• **Eventos**: Especifica un evento, es el punto donde ABAP ejecuta un cierto proceso. Por ejemplo: `START-OF-SELECTION, TOP-OF-PAGE.`

• **Control**: Sentencias de control de flujo de programa. Por ejemplo: `IF`, `WHILE`.

• **Operativas**: Realizan funciones propias según el tipo de palabra clave. Por ejemplo: `WRITE, MOVE.`

Existen dos formas de utilizar comentarios en un report.

1. Con un asterisco (*) en la primera columna de una línea.
2. Con comillas (“) en mitad de una línea.

Podemos combinar sentencias consecutivas de mismo formato. Por ejemplo:

```
WRITE LFA1-LIFNR.
WRITE LFA1-NAME1
WRITE LFA1-ORT01.
```

Es equivalente a:

```
WRITE:LFA1-LIFNR,
LFA1-NAME1,
LFA1-ORT01.
```

Objetos de datos

Existen 3 **clases de objetos de datos** :

• **Campos de bases de datos** guardadas en el diccionario de datos.

Podemos declarar las tablas que queremos utilizar en un programa con la sentencia TABLES.

Ejemplo:
```
TABLES:LFA1.
....
WRITE: LFA1-LIFNR, LFA1-NAME1.
```

• **Literales**: literales de texto entre comillas o números.

Ejemplo:
```
WRITE 'DIRECCIÓN'.
COMPUTE SALES = AMOUNT / 100.
```

• **Variables internas**: Campos auxiliares con nombre de menos de 30 caracteres (sin incluir el carácter blanco). Se declaran con la sentencia DATA.

Ejemplo:
```
DATA:VENTAS_TOTALES TYPE P.
```

Estructura de un programa

Ahora vamos a ver la estructura general de un *report* ABAP. Emplearemos gramática *EBNF* para la notación.

REPORT <nombre> → Nombre programa

TABLES: → Tablas que se utilizan

DATA: → Variables internas

TOP-OF-PAGE. → Inicio de página.

<Sentencias> Ejecuta las instrucciones que se indiquen.

END-OF-PAGE. → Fin de página.

<Sentencias> Ejecuta las instrucciones que se indiquen.

START-OF-SELECTION. → Inicio de programa

<Sentencias> Ejecuta las instrucciones que se indiquen.

END-OF-SELECTION. → Fin de programa

<Sentencias> Ejecuta las instrucciones que se indiquen.

La secuencia de eventos no es relevante.

IV .Declarando y procesando datos

Tipos de Campos

En la Tabla 1 podemos apreciar los tipos de datos que se pueden utilizar en ABAP /4:

Tipos	Long.por defecto	Posible longitud	Valor inicial	Descripción
C	1	1-32000	ESPACIOS	Texto
F	8	8	0.0E+00	Punto flotante
I	4	4	0	Entero
N	1	1-32000	'0000 '	Texto numérico
P	8	1-16	0	Empaquetado
X	1	1-29870	x '00'	Hexadecimal
D	8	8	00000000	Fecha YYYYMMDD
T	6	6	000000	Hora HHMMSS

Tabla 1. Tipos de datos.

Declaración de Campos

Se declaran campos del *report* con la sentencia `DATA`.

Si no se indica lo contrario las variables serán del tipo carácter (texto) y la longitud 1.

Ejemplo: `DATA VAR_CAR.`
`DATA VAR_CAR(8).` → Creará una variable texto de longitud 8.

Con el parámetro `TYPE` podemos utilizar otros tipos de datos.

Ejemplo:

```
DATA NUM_CAR(5)TYPE N.
DATA NUMERO(2)TYPE P.
DATA FECHA_LIMITE TYPE D.
```

Con el parámetro `LIKE` podemos declarar una variable con los mismos atributos de longitud y tipo que una variable de base de datos.

Ejemplo: `DATA ACREEDOR LIKE LFA1-LIFNR.`

Con el parámetro `VALUE` podemos inicializar la variable con un valor distinto al que tiene por defecto.

Ejemplo: `DATA CONTADOR TYPE P VALUE 1.`

Un **registro de datos** es un conjunto de campos relacionados lógicamente en una estructura.

Ejemplo:
```
DATA: BEGIN OF PROVEEDOR
LIFNR LIKE LFA1-LIFNR,
NAME1 LIKE LFA1-NAME1,
CIUDAD(20)VALUE 'BARCELONA',
FECHA TYPE D,
END OF PROVEEDOR.
```

Posteriormente el acceso a los campos del registro de datos será:

```
WRITE: PROVEEDOR-NAME1,
PROVEEDOR-FECHA.
```

También usaremos la instrucción `DATA` para declarar tablas internas. Las tablas internas a diferencia de las de base de datos se guardarán en memoria y no en el diccionario de datos.

Ejemplo:

```
DATA: BEGIN OF MEJORES_PROVEEDORES OCCURS 100,
NOMBRE LIKE LFA1-NAME1,
CIUDAD LIKE LFA1-ORT1,
VENTAS LIKE LFC3-SOLLL,
END OF MEJORES_PROVEEDORES.
```

La cláusula `OCCURS` determina el número de líneas guardadas en memoria principal. Esto no significa que el tamaño máximo de la tabla sea el indicado, ya que si este se desborda los datos se guardan en un fichero de paginación, bajando lógicamente el tiempo de proceso de las tablas internas, pero evitando que el área global de almacenamiento destinado por SAP para tablas internas se agote.

Las tablas internas se declaran, inicializan y referencian como un registro de datos. También podemos utilizar la misma estructura que una tabla de base de datos. Para ello utilizaremos la instrucción `INCLUDE STRUCTURE.`

Ejemplo:

```
DATA BEGIN OF SOCIEDADES OCCURS 10.
INCLUDE STRUCTURE T001.
DATA END OF SOCIEDADES.
```

Asignando valores

Existen diversas formas de asignar valores a una variable en ABAP/4.
Una asignación directa, como resultado de una operación aritmética o como resultado de una conversión automática entre campos con valores de diferente tipo de datos.

La instrucción `MOVE` realiza un transporte del contenido del **var1** al campo **var2**.

```
MOVE <var1> TO <var2>.
```

Podemos sustituir esta última instrucción por:

```
<var2>=<var1>.
```

Que es la simplificación de:

```
COMPUTE <var2>=<var1>.
```

Donde la palabra clave `COMPUTE` es opcional.

También es posible referenciar o asignar valores a una parte de la variable utilizando el **offset**.

```
VARIABLE + offset(longitud)
```

Ejemplo:

```
DATA:VAR1(15)VALUE 'CORITEL MLG.',
VAR2(15)VALUE 'HOLA '.
MOVE VAR1+8(4) TO VAR2+5(4).
WRITE VAR2.
```

Resultado:

```
VAR1 → CORITEL MLG.
VAR2 → HOLA MLG..
```

Si se desean utilizar variables en el offset o la longitud se usará la instrucción `WRITE TO`.

Ejemplo:

```
OFF1 =10.
OFF2 =5.
LEN =4.
WRITE VAR1+OFF1(LEN)TO VAR2+OFF2(LEN).
```

Si se desea chequear la longitud o el tipo de una variable podemos utilizar la instrucción `DESCRIBE FIELD`.

Sintaxis:
```
DESCRIBE FIELD campo LENGTH longitud.
             " " TYPE tipo.
             " " OUTPUT-LENGTH long_salida.
             " " DECIMALS PLACES decimales.
```

Para chequear la longitud de un campo utilizamos la cláusula `LENGTH`.

Para conocer el tipo de datos del campo utilizamos `TYPE`.

Para conocer la longitud de salida utilizamos `OUTPUT-LENGTH`.

Para saber el número de decimales que tiene una cierta variable utilizaremos la cláusula `DECIMALS`.

Para inicializar las variables utilizamos la sentencia: `CLEAR <campo>.`

`CLEAR` inicializa al valor que tiene asignado como valor inicial (ver Tabla 1) sin tener en cuenta a las cláusulas `VALUE` que haya.

La asignación e inicialización de los registros de datos funciona de la misma forma que en las variables normales. Un `CLEAR` inicializa todos los campos del registro.

Podremos conseguir una asignación más potente con `MOVE-CORRESPONDING`**.**

```
MOVE-CORRESPONDING <reg1> TO <reg2>.
```

Esta instrucción mueve de reg1 a reg2 aquellos campos que tengan idéntico nombre.

Conversión de tipo

Si intentamos realizar una asignación de variables de distinto tipo, ABAP/4 intenta realizar una conversión automática de tipo.

Existe una instrucción adicional para la conversión `P->C`.

```
UNPACK <p_num> TO <string>.
```

Que desempaqueta `p_num` en la variable `string` colocando ceros a la izquierda.

Existe una instrucción adicional para la conversión `C->P`.

```
PACK <string> TO <p_num>.
```

Operaciones Aritméticas en ABAP/4

En ABAP/4 las 4 operaciones aritméticas básicas se pueden implementar:

• Con la instrucción `COMPUTE` y los símbolos +, -, /, *.

```
COMPUTE var1 = <Exp.Aritmética>.
```

Donde la palabra `COMPUTE` es opcional.

Si utilizamos paréntesis dejaremos un espacio en blanco precediendo y siguiendo al paréntesis.

• Con las instrucciones: `ADD TO, SUBSTRACT FROM, MULTIPLY BY` y `DIVIDE BY.`

También dispondremos de funciones matemáticas para los números de coma flotante:

```
EXP, LOG, SIN, COS, SQRT, DIV, MOD, STRLEN.
```

Procesando campos de tipo texto

ABAP/4 ofrece algunas instrucciones para el procesamiento de cadenas de texto.

• Para realizar un desplazamiento del contenido de un campo utilizamos `SHIFT`.

`SHIFT <campo>` → Realiza un desplazamiento de un carácter hacia la izquierda.

`SHIFT <campo> BY <n> PLACES (RIGHT)` → Realiza un desplazamiento de n caracteres hacia la izquierda o si se especifica hacia la derecha, introduciendo blancos por el lado opuesto.

```
SHIFT <campo> BY 2 PLACES CIRCULAR (RIGHT).
```
Realiza un desplazamiento cíclico hacia la izquierda o si se especifica hacia la derecha.

• Podemos reemplazar el contenido de ciertos campos con la instrucción `REPLACE`.

```
REPLACE <cadena1> WITH <cadena2> INTO <campo>.
```

Reemplaza `cadena1` por `cadena2` dentro de la variable campo. Si la variable del sistema `SY-SUBRC<>0` es que cadena1 no existe dentro de campo.

`REPLACE` únicamente sustituirá la primera aparición de `cadena1`.

• Existe otra instrucción de sustitución, `TRANSLATE`.

`TRANSLATE <campo> TO UPPER CASE` →Pasa a Mayúsculas

`TO LOWER CASE` → Pasa a Minúsculas.

`USING <regla>` → Reemplaza `campo` según la regla de sustitución indicada.

• La instrucción `SEARCH` busca una cadena dentro de un campo o una tabla.

```
SEARCH <campo>/<tabla> FOR <cadena>.
```

Si el Resultado es positivo `SY-SUBRC=0`. En caso de que sea una tabla `SY-TABIX` contiene la línea de la tabla donde se ha encontrado.

• Para borrar los blancos de una cadena utilizaremos `CONDENSE`.

```
CONDENSE <campo> (NO-GAPS).
```

Esta instrucción borra todos los blancos que se encuentren comenzando la cadena por la izquierda y en caso de encontrar series de blancos intermedios dejará únicamente uno por serie.
Ejemplo:
`" CURSO DE ABAP/4 "` → `"CURSO DE ABAP/4"`

La cláusula `NO-GAPS` borra todos los blancos estén donde estén.

Variables del sistema

ABAP/4 tiene algunas variables internas que se van actualizando automáticamente y que pueden ser utilizadas en los programas.

Todas ellas empiezan por el prefijo `SY-` y ya hemos utilizado alguna de ellas como `SY-SUBRC` que nos da el código de retorno de una instrucción o `SY-TABIX` que contiene la línea de proceso de una tabla interna.

V. Control de flujo en los programas ABAP/4

Formulando condiciones

En ABAP, como en todos los lenguajes estructurados, disponemos de una serie de instrucciones para subdividir el programa en bloques lógicos se ejecutarán cuando se cumpla una cierta condición.

Para introducir una condición utilizaremos la sentencia `IF...ELSE...ENDIF` que podrá aparecer en distintas modalidades.

```
IF <Cond.>.
...
ENDIF.
```

```
IF <Cond.>.
...
ELSE.
...
ENDIF.
```

```
IF <Cond.>.
 ...
ELSEIF.
 ...
ELSEIF.
...
ELSE.
...
:
ENDIF.
```

En las condiciones utilizamos los clásicos operadores.

Y	`AND`
O	`OR`
Igual =,	`EQ`
Distinto <>,	`EN`
Mayor >,	`GT`
Menor <,	`LT`
Mayor o igual >=,	`GE`
Menor o igual <=,	`LE`

Además existen operadores adicionales para comparar cadenas de caracteres.

`<f1> CO <f2>` (Contains Only): f1 sólo contiene caracteres de f2. En caso de ser cierta `SY-FDPOS` contiene la longitud de f1 y si es falsa contiene el offset del 1er. carácter que no cumple la condición.

`<f1> CN <f2>` (Contains Not Only): Negación de la anterior.

`<f1> CA <f2>` (Contains Any): f1 contiene como mínimo algún carácter de f2. Si es cierta SY-FDPOS contiene el offset del 1er.carácter de f1 que está en f2 y si es falsa contiene la longitud de f1.

`<f1> NA <f2>` (Contains Not Any): Negación de la anterior.

`<f1> CS <f2>` (Contains String): f1 contiene la cadena f2. Si la condición es cierta `SY-FDPOS` contiene el offset donde empieza f2 en f1 y si es falsa contiene la longitud de f1.

`<f1> NS <f2>` (Contains No String): Negación de la anterior.

`<f1> CP <f2>` (Contains Pattern): f1 corresponde al patrón f2. En el patrón podemos utilizar: + como cualquier carácter, * como cualquier cadena de caracteres, # para utilizar los caracteres +,*, # en la comparación. Si la condición es cierta `SY-FDPOS` contiene el offset de f2 en f1 y si es falsa contiene la longitud de f1.

`<f1> NP <f2>` (Contains No Pattern): Negación de la anterior.

También podremos utilizar operadores especiales:

`IF <f1> BETWEEN <f2> AND <f3>.` → Para chequear rangos

`IF <f1> IS INITAL.` → Para chequear valores iniciales.

Si queremos ejecutar diferentes instrucciones en función del contenido de un campo podemos utilizar la sentencia `CASE`.

```
CASE <campo>.
WHEN <valor1>.
....
WHEN <valor2>.
....
:
WHEN OTHERS.
....
ENDCASE.
```

Por último existe la instrucción condicional, `ON CHANGE OF...ENDON`, que permitirá la ejecución de un bloque de instrucciones, si se ha producido un cambio de valor de un cierto campo durante el acceso a base de datos o una tabla interna. Como procesar una tabla interna o un acceso a base de datos, ya lo veremos más adelante.

```
ON CHANGE OF <campo>.
....
ENDON.
```

Proceso de bucles

Para realizar procesos repetitivos utilizaremos **DO** y **WHILE**.

• La instrucción `DO` permite ejecutar un bloque de instrucciones tantas veces como se especifique.

```
DO <n> TIMES.
...
ENDDO.
```

En la variable del sistema `SY-INDEX` tendremos un contador del número de repeticiones.

Es posible anidar `DO` 's. En ese caso el `SY-INDEX` hará referencia al bucle en proceso.

• La instrucción `WHILE` permite ejecutar un bloque de instrucciones mientras se cumpla una condición.

```
WHILE <cond>.
...
ENDWHILE.
```

De la misma forma que la instrucción `DO`, `WHILE` permite anidar bucles.

Sentencias de control

Las sentencias descritas a continuación se utilizarán para terminar el procesamiento de un bucle o proceso.

• La instrucción: `CHECK <cond>.`

Realiza un chequeo de <cond> de forma que si dentro de un bucle la condición es **falsa**, saltará todas las instrucciones que siguen al `CHECK` e iniciará la siguiente pasada al bucle. Fuera de un bucle si la condición es **falsa**, saltará todas las instrucciones que siguen al `CHECK` hasta el final del evento o programa en proceso.

• La instrucción: `EXIT`.

Dentro de un bucle saldrá del bucle y fuera de un bucle saldrá del programa. Si la instrucción `EXIT` está dentro de varios bucles anidados, únicamente saldrá del bucle en proceso.

• La instrucción: `STOP`.

Con `STOP` finalizaremos el report (programa) en ejecución, pero antes ejecutaremos el evento `END-OF-SELECTION`.

• La instrucción: `LEAVE`.

Con `LEAVE` finalizaremos el *report* (programa) en ejecución, **sin ejecutar** el evento `END-OF-SELECTION`.

VI. Introducción a las sentencias de salida de reporte

A continuación veremos un resumen de las sentencias de salida de *reports* más básicas.

• Como ya hemos visto en los ejemplos de los capítulos anteriores para visualizar un valor utilizaremos la sentencia `WRITE.`

```
WRITE /(<offset>)(<long>) <datos a visualizar>.
```

Con la Barra **/** indicaremos si queremos saltar una línea o no antes de imprimir (opcional).

Con el Offset indicaremos la columna donde empezará la impresión (opcional).

Con Long. indicaremos la longitud de los valores a visualizar (opcional).

• Podemos imprimir una línea de Subrayados con la sentencia `ULINE`. Tendrá las mismas propiedades que el `WRITE`.

```
ULINE /(<offset>)(<long>).
```

• Para saltar una o varias líneas utilizaremos `SKIP`.

```
SKIP <n>.
```

Por defecto el salto será de una única línea.

• Para saltar una página utilizaremos `NEW-PAGE`.

• Para introducir parámetros en la ejecución del *report* existen varias opciones.

La fórmula más sencilla es la sentencia `PARAMETERS`.

```
PARAMETERS: <var> TYPE <tipo>
LIKE <tipo>
DEFAULT <valor> → Igual que el VALUE.
OBLIGATORY. → Obliga a introducir algún valor.
LOWER CASE. → Permite introducir minúsculas.
```

El nombre del parámetro no puede ser superior a 8 caracteres.

VII. Tablas internas

Si deseamos guardar una **colección de registros de datos de la misma estructura** en memoria sin necesidad de acceder a la base de datos y poder realizar operaciones diversas con este conjunto de información, utilizaremos las **tablas internas**.

Como declarar tablas internas

Las tablas internas se declararán de la siguiente manera:

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

Definiremos una tabla interna con n líneas en memoria, más una línea de cabecera o área de trabajo.

La cantidad de líneas que especifiquemos en el OCCURS no limita el tamaño de la tabla, sino la cantidad de registros que se guardan en memoria simultáneamente. Esto hace necesario un especial cuidado al proponer el número de líneas ya que un OCCURS muy grande supone un gran gasto de recursos del sistema y un `OCCURS` pequeño un acceso muy lento, ya que necesita de un proceso de paginación.

Llenado de una tabla interna

• `APPEND`: Añade un registro a una tabla interna con los valores que tengamos en el área de trabajo.

```
APPEND <intab>.
```

• `COLLECT`: Añade o suma la línea de cabecera. Sumará los campos de tipo P, F, I, si existe una línea en la tabla con campos idénticos (tipo C) a los del área de trabajo. El problema de esta instrucción es que es bastante lenta. Se puede sustituir por las instrucciones `READ` e `INSERT` o `MODIFY`.
• Podemos llenar una tabla interna con el contenido de una tabla de base de datos. Siempre que la tabla interna tenga la misma estructura que la tabla de base de datos.

```
SELECT * FROM <tab> INTO TABLE <tabint>.
```

Ordenar una tabla interna

Para clasificar una tabla interna utilizamos SORT.

```
SORT <intab>.
```

Esta instrucción realiza una ordenación por la estructura de la tabla sin tener en cuenta los campos P, I, F.

Para ordenar por el campo(s) que necesitemos (sea del tipo que sea):

```
SORT <intab> BY <campo1>....<campo n>.
```

Si no se indica lo contrario la ordenación por defecto es ascendente.

```
SORT ... ASCENDING o DESCENDING.
```

Procesamiento de una tabla interna

Podemos recorrer una tabla interna con la instrucción LOOP...ENDLOOP.

```
LOOP AT <intab> (WHERE <cond>).
...
ENDLOOP.
```

En cada iteración coloca la línea de la tabla que se está procesando en la línea de cabecera.

Podemos restringir el proceso de una tabla con una condición WHERE.

Si no existe ningún registro de la tabla que cumpla la condición especificada en la cláusula WHERE, la variable del sistema SY-SUBRC será distinta que 0. Dentro del LOOP la variable SY-TABIX contiene el índice de la entrada que está procesando en ese momento.

También es posible hacer un:

```
LOOP AT <intab> FROM <inicio> TO <fin>.
...
ENDLOOP.
```

Donde <inicio> y <fin> son índices de la tabla interna.

Tratamiento de niveles de ruptura

En el tratamiento de un `LOOP` podemos utilizar sentencias de control de ruptura.

```
AT FIRST.
...
ENDAT.
```
→ Realiza las instrucciones que hay a continuación del `AT FIRST` para la primera entrada de la tabla.

```
AT LAST.
...
ENDAT.
```
→ Realiza las instrucciones que hay a continuación del `AT LAST` para la última entrada de la tabla.

```
AT NEW <campo>
...
ENDAT.
```
→ Realiza las instrucciones que hay a continuación del `AT NEW` para cada inicio de nivel de ruptura.

```
AT END OF <campo>
...
ENDAT.
```
→ Realiza las instrucciones que hay a continuación del `AT END` para cada final de nivel de ruptura.

Si utilizamos la instrucción `SUM` dentro de un `AT...ENDAT` realizará la suma de todos los campos P, I, F de ese nivel de ruptura (para el cálculo de subtotales). El resultado lo encontraremos en el área de trabajo de la tabla.

Será necesario que la tabla interna esté ordenada en el mismo orden que la utilización de los niveles de ruptura.

Así la utilización conjunta de todas estas instrucciones será:

```
SORT <intab> BY <c1> <c2>.
LOOP AT <intab>.
AT FIRST.
...(SUM)...
ENDAT.

AT NEW <c1>.
...(SUM)...
ENDAT.


AT NEW <c2>.
...(SUM)...
ENDAT.
..... "Proceso Normal de la tabla
AT END OF <c2>.
...(SUM)...
ENDAT.
```

```
AT END OF <c1>.
...(SUM)...
ENDAT.
AT LAST
...(SUM)...
ENDAT.
ENDLOOP.
```

Lectura de entradas de una tabla

• Podemos buscar un registro concreto en una tabla sin necesidad de recorrerla.

```
READ TABLE <intab>.
```

Para ello en primer lugar rellenaremos la línea de cabecera con la clave de búsqueda y luego haremos el `READ`.

El resultado de la búsqueda lo tendremos en `SY-SUBRC`.

Si `SY-SUBRC = 0` la búsqueda ha sido positiva.

Si `SY-SUBRC <> 0` no ha encontrado el registro solicitado.

Existen otras extensiones a la instrucción `READ` que necesitarán que la tabla esté ordenada.

Podemos buscar por clave con:

```
READ TABLE <intab> WITH KEY <clave>.
```

No necesita llenar la línea de cabecera. Buscará desde el inicio de la tabla que carácter a carácter coincida con la clave.

• Es posible una búsqueda aún más rápida con una búsqueda binaria.

```
READ TABLE <intab> WITH KEY <clave> BINARY SEARCH.
```

• Una lectura directa de un registro de la tabla la podemos realizar con:

```
READ TABLE <intab> INDEX <num>.
```

Modificando tablas internas

Una vez llena la tabla interna tenemos la posibilidad de modificar los datos con una serie de sentencias ABAP/4.

• `MODIFY`: Podemos sobrescribir el contenido de la entrada <i> con el contenido de la línea de cabecera.

```
MODIFY <intab> (INDEX <i>).
```

Dentro de un `LOOP`, la cláusula `INDEX` es opcional. Por defecto será el contenido de la variable `SY-TABIX`.

• `INSERT`: Añade una entrada delante de la entrada <i> con el contenido de la línea de cabecera.

```
INSERT <intab> (INDEX <i>).
```

• `DELETE`: Para borrar una entrada de una tabla.

```
DELETE <intab> (INDEX <i>).
```

Otras instrucciones de manejo de tablas:

• Inicializar el área de trabajo o línea de cabecera.

```
CLEAR <intab>.
```

• Inicializar (borrar) contenido de una tabla.

```
REFRESH <intab>.
```

• Liberar el espacio ocupado por una tabla en memoria.

```
FREE <intab>.
```

• Para obtener información sobre una tabla interna.

```
DESCRIBE TABLE <tab>
LINES <contador_entradas>
OCCURS <valor_occurs>.
```

VIII.Subrutinas

Tipos de subrutinas

Existen 3 tipos de subrutinas o subprogramas.

Internas: El Subprograma y la llamada a éste están en el mismo programa.

Externas : El Subprograma y la llamada a éste están en programas distintos.

Biblioteca de funciones (Módulos de función) : Funciones externas al programa con *interface* de llamada claramente definido.

Subrutinas internas

```
PERFORM <modulo>.  → Llamada a un procedimiento o subprograma.
FORM <modulo>
....  → Subprograma.
ENDFORM.
```

El programa principal y el procedimiento se podrán comunicar mediante parámetros.

```
...
PERFORM <modulo> USING var1 var2...
...
FORM <modulo> USING var1 var2...
...
ENDFORM.
```

Los parámetros pueden ser pasados por **valor** (E) o por **referencia** (E/S). Por defecto serán por referencia.

Si queremos utilizar parámetros por valor, la cabecera del módulo será:

```
FORM <modulo> USING VALUE(var1).
...
ENDFORM.
```

Tanto las variables definidas al inicio del *report* como las tablas son globales a todas las subrutinas y por tanto accesibles en cualquier momento.

Si encontramos alguna instrucción del tipo CHECK o EXIT que signifique salir de un cierto FORM, previamente ejecutará el ENDFORM y por tanto se pasarán los parámetros que tenga el procedimiento.

También es posible pasar como parámetro tablas internas.

```
PERFORM <modulo> TABLES <intab>...
USING <var1><var2>...
FORM <modulo> TABLES <intab>
USING <var1>...
ENDFORM.
```

Especificaremos las tablas siempre antes que el resto de parámetros.

En este caso sólo se pueden hacer operaciones con filas enteras, pero no nos podremos referenciar sobre campos concretos de la tabla o hacer COLLECTS, ya que no se conocerá la estructura de la tabla.

Podemos pasar como parámetros registros de datos o áreas de trabajo con:

```
PERFORM <modulo> USING <reg>.
FORM <modulo> USING <reg> STRUCTURE <estructura>.
...
ENDFORM.
```

Es decir con la cláusula STRUCTURE podemos pasar la estructura de una tabla, entonces podemos acceder a campos de una tabla pasada como parámetro con:

```
PERFORM <modulo> TABLES <intab> USING <var1>...
FORM <modulo> TABLES <intab> STRUCTURE <estructura>
USING <var1>...
ENDFORM.
```

Dentro de cada subrutina es posible declarar datos con la sentencia DATA, que sólo serán visibles dentro del módulo donde esté declarado. ABAP/4 creará un espacio para esas variables que será liberado al salir del módulo. Por tanto se podrán utilizar variables con el mismo nombre que variables globales, aunque el valor que tengan será siempre el local en el módulo.

Las tablas de base de datos son globales a todo el programa, si se quiere utilizar una tabla localmente en una subrutina, se debe declarar con LOCAL, al inicio de la subrutina, en vez de con TABLES.

```
LOCAL <tabla>.
```

Subrutinas externas y Módulos de función

En un report, podemos llamar a subrutinas que no se encuentren dentro de nuestro código. Para ello disponemos de la modalidad de llamada a subrutina pasándole el programa, el uso de includes, y las llamadas a módulos de función.

• Si queremos llamar a una subrutina que está en un programa distinto utilizamos:

```
PERFORM <sub> (<programa>) USING...
```

• También existe la posibilidad de añadir porciones de código del tipo `INCLUDE` con la instrucción:

```
INCLUDE <report>.
```

En el código del `INCLUDE` no utilizaremos la sentencia `REPORT...`

• Los **módulos de función** son módulos especiales guardados en una librería central, y agrupados por la función que realizan. Principalmente se caracterizan por un *interface* **definido** y porque realizan **tratamiento de excepciones.**

Se caracterizan por un *interface* definido ya que su diseño facilita el paso de parámetros tanto de entrada como de salida.

```
CALL FUNCTION <funcion>.
EXPORTING <par_E>=<valor>
...
IMPORTING <par_S>=<valor_ret>
...
TABLES <tab_Func>=<tab_Prog>
...
EXCEPTIONS <excep>=<valor>
...
```

Donde en el `EXPORTING` especificamos los parámetros de entrada, en el `IMPORTING` (opcional) el resultado o retorno de la función y en `TABLES` (opcional) las tablas que se utilizan como parámetros.

Los módulos de función también se caracterizan por realizar un tratamiento de excepciones. En el *interface* de los módulos de función se indican los valores de excepciones para el retorno del módulo, que posteriormente con el `SY-SUBRC` se pueden comprobar.

El código de la función puede activar excepciones mediante las instrucciones:

```
MESSAGE...RAISING <excepcion>.
```

Ó

```
RAISE <excepcion>.
```

Para acceder a la biblioteca de módulos de función es posible utilizar el comando SHOW FUNCTION * desde el editor de programas o desde el tratamiento de módulos de función del menú **Herramientas ->CASE ->desarrollo ->Actualizar programas->módulos de función**, desde donde podremos además crearlos y mantenerlos.

Intercambio de datos mediante la memoria global de SAP

Es posible intercambiar datos entre *reports* distintos (llamados desde instrucciones `SUBMIT`) a través de la memoria de SAP.

Para grabar en memoria:

```
EXPORT <campo>...INTO MEMORY.
```

Para recuperar de memoria:

```
IMPORT <campo>...INTO MEMORY.
```

IX. Diccionario de datos. Como leer y procesar tablas de la base de datos.

Diccionario de datos

El diccionario de datos (D.D.) es una fuente de información centralizada.

Los distintos objetos del Diccionario de datos están estructurados en:

Campo
Tabla
Elemento de datos
Dominio

Los **elementos de datos** describen el significado de un campo independientemente de las tablas donde se utilicen. Es decir, tienen un carácter semántico.

Los **dominios** describen el campo de valores posible s. Tendrán un carácter técnico.

Ejemplo:

```
TABLAS:SKB1,SKM1…
CAMPO: STEXT
ELEM.DATOS: STEXT_SKB1
DOMINIO: TEXT50
FORMATO INTERNO: Tipo C de 50 Posiciones
```

Tendremos a nuestra disposición un sistema de información del diccionario de datos, **Info-System**, que proporciona información sobre contenido de las tablas, campos, dominios, programas...etc.

Existen diversos tipos de tablas:

-Tablas `TRANSP` (transparentes): Tablas normales relacionales (SQL).
-Tablas `POOL`: Tablas SAP que se guardan junto a otras tablas SAP en una única tabla física de BDD. Mejorando el acceso a los registros.
-Tablas `CLUSTER`: Varias tablas que se guardan en un cluster de BDD. Se guardan registros de varias tablas SAP con la misma clave cluster, en el mismo cluster físico de la base de datos.

El diccionario de datos se dice que es integrado y activo. **Integrado** porque integra el D.D. con el *Screen-Painter*, Programas ABAP, *Dynpros*, *Superficies CUA*...y **Activo** porque si modificamos algún objeto del diccionario de datos, el sistema automáticamente regenera el '*Time Stamp*' de los programas que utilicen esos objetos.

Los datos en el sistema SAP

Podemos clasificar los datos del sistema en datos maestros, datos de movimientos, y datos del sistema.

• **Datos maestros**: Son datos que no se modifican muy a menudo.

Ej: Materiales, Cuentas, Bancos, Clientes...
Se almacenarán en tablas transparentes.

• **Datos de movimientos** : Datos muy volátiles y con gran volumen de generación.

Ej: Facturas, Pedidos...

Se suelen guardar en tablas tipo `CLUSTER` todos ellos con formato parecido (documentos).

• **Datos del sistema o de control**: Muchas tablas con pocos datos. Se suelen guardar en tablas de tipo `POOL`.

Instrucciones SQL de ABAP/4

ABAP/4 tiene un subconjunto de sentencias SQL para su aplicación sobre tablas de la base de datos SAP.

Éstas son:

```
SELECT, INSERT, UPDATE, MODIFY, DELETE, COMMIT WORK,
ROLLBACK WORK.
```

Además de las variables del sistema:

`SY-SUBRC`: Código de retorno de una operación.
`SY-DBCNT`: Cantidad de registros afectados por la operación procesada.

I) SELECT.
La sentencia `SELECT` será la instrucción fundamental para leer información de la base de datos.

• **Lectura de un único registro:**

```
SELECT SINGLE * FROM <tab> WHERE <cond>.
```

Como realizamos la búsqueda de un registro, en la condición sólo podremos utilizar la igualdad y el operador AND, ya que especificaremos toda la clave del registro.

Si SY-SUBRC=0 → Registro encontrado. Resultado en área de trabajo.
Si SY-SUBRC=4 → No existe el registro buscado.

• **Lectura Iterativa**: Selección de un grupo de registros.

```
SELECT * FROM <tab>
(WHERE <cond>).
ENDSELECT.
```

Selecciona todos los registros que cumplan la condición de la cláusula WHERE, o todos en caso de no utilizarla. El resultado lo tendremos en el área de trabajo, es decir en cada iteración del bucle SELECT...ENDSELECT tendremos un registro leído en dicha área.

Si **SY-SUBRC** =0 → Algún registro encontrado.
Si **SY-SUBRC** =4 → No existe ningún registro que cumpla la condición del WHERE.

Si la condición del WHERE se acerca a la clave de la tabla, la búsqueda de registros será más óptima.

Otras posibilidades del WHERE:

SELECT * FROM <tab> WHERE <campo>...
BETWEEN <var1> AND <var2>. → Si <campo> está entre los valores <var1> y <var2>.
LIKE <literal enmascarado>. → Si <campo> cumple la máscara.

Se pueden utilizar:

'_' como carácter cualquiera.
'%' como una cadena de caracteres.
IN (<var1>, <var2>...). → Si <campo> esta en el conjunto de valores <var1>, <var2>...

• **Otras lecturas:**

Podemos leer una tabla de base de datos y simultáneamente llenar una tabla interna con el resultado de la lectura.

```
SELECT * FROM <tab> INTO TABLE <intab> (WHERE <cond>).
```

Llena la tabla interna <intab> machacando los registros que pudiera tener esta. Si queremos que respete los registros que tenía la tabla interna antes de realizar el `SELECT` tendremos que utilizar:

```
SELECT * FROM <tab> APPENDING TABLE <intab> (WHERE <cond>).
```

Podemos indicar un orden en el proceso de selección de registros.

```
SELECT *...ORDER BY <campo1> <campo2>...
```

Si queremos seleccionar un registro para bloquearlo de posibles modificaciones.

```
SELECT SINGLE FOR UPDATE * FROM <tab>.
```

II) INSERT.

La sentencia `INSERT` permite introducir registros sencillos o el contenido de una tabla interna en una base de datos SAP.

```
INSERT <tab>.
```

Grabará en la BDD el registro de cabecera. Por tanto previamente a esta instrucción moveremos los valores que queremos introducir sobre el área de trabajo de la tabla.

Si `SY-SUBRC=0` → Registro insertado.
Si `SY-SUBRC>0` → La clave del registro que queríamos insertar ya existía en la tabla.

También es posible introducir datos desde una tabla interna.

```
INSERT <tab> FROM TABLE <intab>.
```

Si `SY-SUBRC=0` → Registros insertados.

Si existe algún registro en la base de datos con clave igual a algún registro de la tabla interna, se producirá un error de ejecución del programa.

La tabla interna podrá tener la misma estructura que la tabla de base de datos utilizando `INCLUDE STRUCTURE` en su declaración.

III) UPDATE.

La sentencia UPDATE permite modificar el contenido de uno o varios registros.

```
UPDATE <tab>.
```

Modifica el registro de la base de datos que está especificado en el registro de cabecera.

Si queremos modificar el contenido de más de un registro a la vez:

```
UPDATE <tab> SET <campo>=<valor> WHERE <cond>.
```

Con este UPDATE, todos los registros que cumplan <cond> modificarán el contenido del <campo> por <valor>.

También es posible utilizar la cláusula SET con:

```
<campo>=<campo>+<valor>o
<campo>=<campo>-<valor>
```

Es posible modificar registros desde una tabla interna:

```
UPDATE <tab> FROM TABLE <intab>.
```

Si el sistema no puede actualizar un registro, el proceso no finalizará sino que continuará con el siguiente registro.

Si SY-SUBRC=0 Todos los registros modificados.
Si SY-SUBRC=4 No todos los registros han sido modificados.

En SY-DBCNT Tendremos la cantidad de registros modificados.

IV) MODIFY.

La sentencia MODIFY se utilizará cuando no estemos seguros si utilizar un INSERT o un UPDATE. Es decir, cuando no sepamos con certeza si un registro existe o no, para modificarlo o añadirlo.

```
MODIFY <tab>.
MODIFY <tab> FROM TABLE <intab>.
```

En caso de que sepamos si existe o no un registro, por eficacia utilizaremos INSERTs o UPDATEs.

V) DELETE.

Para realizar borrados de datos se aplica la sentencia DELETE.

```
DELETE <tab>.
```

Borrará el registro que especifiquemos en el área de trabajo.

Para borrar más de un registro (todos los que cumplan una cierta condición).

```
DELETE FROM<tab> WHERE <cond>.
```

Podemos borrar de BDD todos los registros de una tabla interna.

```
DELETE FROM <tab> FROM TABLE <intab>.
```

Si SY-SUBRC=0 → Todos los registros han sido borrados.
Si SY-SUBRC=4 → No todos los registros han sido borrados.

En SY-DBCNT → Tendremos la cantidad de registros borrados.

Otros aspectos de la programación de BDD

• El **control del mandante** es automático. Siemp re se procesará el mandante en uso.

Si queremos controlar manualmente el mandante en una instrucción de lectura o actualización utilizaremos la cláusula CLIENT SPECIFIED. Es decir, si queremos obtener o modificar datos de un cliente diferente al de entrada.

• Las instrucciones INSERT, DELETE, MODIFY y UPDATE se utilizarán en la medida que sea posible el menor número de veces sobre tablas SAP. Siempre se intentará insertar o modificar datos mediante transacciones estándares SAP o vía *Batch Input*. Ya que no siempre es fácil conocer la compleja estructura de toda la base de datos SAP y así nos aseguramos no producir alguna inconsistencia en la base de datos.

• El **Bloqueo de objetos** :
Para bloquear un registro en el momento de una actualización sobre éste utilizamos FOR UPDATE.

```
SELECT SINGLE FOR UPDATE * FROM <tab>.
```

Si queremos bloquear todos los objetos que están involucrados en una actualización será necesario utilizar el *SAP looking Technique*. Cada aplicación tiene muchos módulos de función para bloquear objetos. Para buscarlos será necesario ir al mantenimiento de módulos de función y buscar por la clave ***enqueue*** o ***dequeue***.

• **Actualización de la base de datos o Recuperación**:
Para finalizar una unidad de procesamiento lógico (**LUW**) de base de datos se utiliza un `COMMIT WORK`, que realiza un `UPDATE` físico en la base de datos, haciendo irrevocable cualquier modificación en la base de datos. Si deseamos deshacer todas las operaciones realizadas sobre la base de datos desde el último `COMMIT WORK`, realizaremos un `ROLLBACK WORK`**.**

• **Chequeo de autorizaciones**:
Las instrucciones SQL de SAP no realizan ninguna verificación de autorizaciones, lo cual resulta peligroso ya que todo el mundo puede acceder a todos los datos que acceda un *report*.

Es responsabilidad del programador el comprobar si un usuario está autorizado a acceder a esa información.

Para chequear las autorizaciones de un determinado usuario utilizaremos la instrucción `AUTHORITY-CHECK`.

```
AUTHORITY-CHECK OBJECT <objeto_de_autorización>
ID <Campo1> FIELD <f1>
ID <Campo2> FIELD <f2>
ID <Campo3> DUMMY.
...
```

Donde <nombre(n)> son los campos de autorización del objeto y <f(n)> es un valor posible de autorización.

El parámetro `DUMMY` indicará que no hace falta verificar ese campo.

Si `SY-SUBRC=0` → Usuario autorizado.
Si `SY-SUBRC<>0` → Usuario NO autorizado.

Ejemplo:

Verificar el objeto de autorización ‘Acreedor: Autorizaciones para sociedades’ ((F_LFA1_BUK), para saber si el usuario puede efectuar la operación Visualizar (01), sobre proveedores de la sociedad 0001.

```
AUTHORITY CHECK OBJECT 'F_LFA1_BUK'
ID 'ACTVT' FIELD '01'
ID 'BUKRS' FIELD '0001'.
```

Para obtener una documentación más exhaustiva sobre el funcionamiento del `AUTHORITY-CHECK`, ver la **documentación online** del editor de ABAP/4.

• Sentencias en **SQL nativo**:
Podemos ejecutar cualquier sentencia de SQL permitida por el gestor de base de datos sobre el que corra el sistema R/3, utilizando **EXEC SQL**. En este caso las instrucciones de base de datos no están restringidas al subconjunto SAP-SQL que hemos estado estudiando a lo largo de este capítulo.

Gracias al *interface* `EXEC SQL` también es posible acceder a datos externos a SAP, desde un programa en ABAP/4.

Sintaxis:

```
EXEC SQL.
<Instrucciones SQL-Nativas>.
ENDEXEC.
```

Tenemos que tener en cuenta en la utilización de SQL nativo, que no todas las bases de datos SAP pueden ser accedidas con este sistema, ya que no todas tienen una representación física de tabla en el gestor de base de datos. Por ejemplo las tablas de tipo `POOL` y `CLUSTER` no son tablas reales de base de datos, aunque sean consideradas como tales y mantenidas por el diccionario de datos.

X. Formateando un listado

ABAP/4 tiene una serie de instrucciones especialmente diseñadas para que la generación de reports sea más sencilla.

Formato de los datos de salida

Ya hemos visto en el capitulo 8 un resumen de las sentencias de salida de r*eports* más básicas.

```
WRITE /<offset> (<long>) <datos a visualizar>.
ULINE /<offset> (<long>) <datos a visualizar>.
SKIP <n>.
NEW-PAGE.
```

Además de estas sentencias fundamentales tenemos a nuestra disposición otras posibilidades:

• Para escribir un campo, variable o literal justamente debajo de otros sin tener que calcular la columna, utilizamos la cláusula `UNDER` del `WRITE`.

```
WRITE <campo2> UNDER <campo1>.
```

• Si queremos especificar la columna de un texto en forma de variable utilizamos.

```
POSITION <columna>.
```

• Si queremos ir a una determinada línea dentro de la misma página.

```
SKIP TO LINE <n>.
```

Cuando utilizamos la instrucción `WRITE` con números empaquetados, el sistema trunca por la izquierda en caso de ser necesario (deja un * como indicador de que ha truncado) y rellena con blancos si sobra espacio. Tenemos que tener en cuenta que si es negativo el signo ocupará una posición. Si se especifican los decimales con la cláusula `DECIMALS` del `DATA`, el punto o coma decimal también ocupará una posición. El signo decimal (punto o coma) estará determinado por los valores del registro de usuario.

Ejemplo:

```
DATA NUMERO TYPE P DECIMALS 2 VALUE -123456.
WRITE NUMERO.
1.234,56-
```

Y si no cabe el número:

```
WRITE (6)NUMERO.
*4,56-
```

• Podemos formatear la salida de un número empaquetado.

Evitamos que aparezca el signo con `NO-SIGN`.

```
WRITE <campo> NO-SIGN.
```

Para visualizar importes correctamente dependiendo de la moneda del importe, usaremos el `CURRENCY` o el `CURRENCY LOCAL`, según la moneda a visualizar.

```
WRITE <campo_importe> CURRENCY <moneda>.

WRITE <campo_importe> CURRENCY LOCAL.
```

• Si se desea formatear la salida de un campo según una cierta máscara utilizaremos el parámetro `USING EDIT MASK` de la instrucción `WRITE`.

```
WRITE <campo> USING EDIT MASK <mascara>.
```

Los caracteres de la máscara pueden ser:

'_': Un carácter del campo a formatear.
':': Un separador. Puede ser cualquier carácter especial menos el '_'.
'LL ': Justifica por la izquierda (valor por defecto). (Al principio de la máscara).
'RR ': Justifica por la derecha. (Al principio de la máscara).

Ejemplo:

```
WRITE /(8) SY-UZEIT USING EDIT MASK '__:__:__'.
```

• Si queremos suprimir los ceros iniciales de una cadena de caracteres haremos : `WRITE <campo_Caracter> NO-ZERO.`

• Para formatear fechas es posible realizar :

```
WRITE <campo_Fecha> DD/MM/YY.
WRITE <campo_Fecha> MM/DD/YY.
WRITE <campo_Fecha> DD/MM/YYYY.
WRITE <campo_Fecha> MM/DD/YYYY.
```

• Podemos modificar los atributos de pantalla para un campo.

```
FORMAT INTENSIFIED ON/OFF.
FORMAT INVERSE OFF/ON.
FORMAT INPUT OFF/ON.
```

```
FORMAT COLOR n.
FORMAT RESET.
```

Ver la **documentación online** del editor ABAP/4 para obtener información más detallada sobre los usos y sintaxis posibles de esta instrucción.

Formato de página

También hay un grupo de instrucciones destinadas a dar formato a la salida del *report*, ya sea por pantalla o por impresora.

• Podemos hacer tratamientos por inicio y fin de página con los eventos :

```
TOP-OF-PAGE y END-OF-PAGE.
```

`END-OF-PAGE` no se ejecutará si el salto de página se produce con un `NEWPAGE.`

• Si no queremos que la cabecera del *report* sea la estándar de SAP, ya que la queremos controlar nosotros directamente en el evento `TOP-OF-PAGE`, utilizaremos:

```
REPORT <Zxxxxxxx> NO STANDARD PAGE HEADING.
```

• El formato de la página de *report* se define también desde la instrucción `REPORT.`

`REPORT <Zxxxxxxx> LINE-SIZE <n>` → Ancho de línea.
`LINE-COUNT <n(m)>` → Líneas por página (n). Si se desea se pueden reservar líneas para un pie de página (m).
`PAGE-COUNT <n>.` → Nº máximo de páginas.

• Podemos impedir que con un salto de página se corten líneas que pertenezcan a una agrupación de líneas con significado lógico propio. Con la instrucción `RESERVE` reservamos un número de líneas.

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

Esta instrucció n se colocará justo antes del `WRITE` que se quiere 'reservar ', si no cabe se imprimirá en la siguiente página.

• Hay varias formas de imprimir un *report* :

- Una vez ha salido el *report* por pantalla con la opción de 'Imprimir'.
- Imprimir sin visualizar por pantalla con la opción 'Imprimir'desde la pantalla de selección o de parámetros.

Desde el programa ABAP/4 podemos controlar la impresión con la instrucción: `NEW PAGE PRINT ON/OFF` → Pantalla o impresora.

NO-DIALOG → No visualiza la pantalla de opciones de impresión.
LINE-COUNT <n> → Líneas por página.
LINE-SIZE <n> → Tamaño de línea.

DESTINATION <des> → Impresora destino.
IMMEDIATELY <x> → Impresión inmediata S/N.

Para más información sobre otras opciones, ver la ayuda del editor de ABAP/4.

• Para determinar formatos especiales de impresión utilizaremos la instrucción

```
PRINT-CONTROL.
PRINT-CONTROL FONT <n>
CPI <n>
LPI <n>
SIZE <n>
COLOR <color>
LEFT MARGIN <col>.
...
```

Para más información sobre otras opciones, ver la ayuda del editor de ABAP/4.

Selección de parámetros. Pantalla de selección (SELECTION SCREEN)

Si deseamos introducir una serie de delimitaciones en la ejecución de un *report* a nivel de parámetros, dispondremos de dos posibilidades.

-El PARAMETERS que permite utilizar parámetros de cua lquier tipo en la pantalla de selección.

-El SELECT-OPTIONS que permite determinar un criterio de selección de los datos a utilizar en el *report*.

Veamos la sintaxis principal de la sentencia PARAMETERS.

```
PARAMETERS: <var> TYPE <tipo>

LIKE <tipo>
DEFAULT <valor> → Igual que el VALUE.
OBLIGATORY → Obliga a introducir algún valor.
LOWER CASE. → Permite introducir minúsculas.
```

La instrucción SELECT-OPTIONS:

```
SELECT-OPTIONS <var> FOR <campo_tabla>.
```

<var> como mucho tendrá 8 caracteres.

La variable <var> tomará los posibles valores a seleccionar y <campo_tabla> nos indica para que campo y de que tabla será utilizado el parámetro (esto implícitamente nos está dando el tipo y la longitud de los posibles valores).

Con esta sentencia, automáticamente en la pantalla de selección se podrán introducir rangos de valores posibles para el parámetro.

Para cada sentencia SELECT-OPTIONS, el sistema crea una tabla interna con el nombre de <var>. Cada registro de la tabla está formado por los campos:

```
<var>-LOW, <var>-HIGH, <var>-SIGN, <var>-OPTION.
```

El contenido de cada registro será respectivamente: el valor inferior, el superior, el signo (**I**ncluido / **E**xcluido) y el operador.

En la pantalla de selección si queremos realizar una selección compuesta de más de una condición (más de un registro en la tabla interna), tendremos que hacer un Clic sobre la Flecha situada a la derecha de cada campo.

Para seleccionar los datos de lectura en tiempo de ejecución mediante los valores de selección, utilizaremos la cláusula WHERE de la instrucción SELECT y el operador IN, que buscará en la tabla de base de datos todos los registros que cumplan las condiciones incluidas en la tabla interna de la pantalla de selección.

```
SELECT-OPTIONS <var> FOR <campo>.
...
SELECT * FROM <tab> WHERE <campo> IN <var>.
```

En la pantalla de selección aparecerá el texto **<var>** como comentario a la selección de datos, si queremos que el texto sea distinto al nombre de la variable tendremos que ir a la opción **Textos de selección** del menú **Pasar a ->Elementos de Texto**.

Veamos ahora que otras opciones existen en la utilización de la instrucción SELECTOPTIONS.

• Para asignar valores iniciales a un criterio de selección utilizamos la cláusula DEFAULT.

```
SELECT-OPTIONS <var> FOR <campo> DEFAULT <valor>.
```

Si queremos inicializar un rango de valores (inferior y superior) usaremos:

```
SELECT-OPTIONS <var> FOR <campo> DEFAULT <ini> TO <fin>.
```

• Podemos hacer que se acepten valores en minúsculas.

```
SELECT-OPTIONS <var> FOR <campo> LOWER CASE.
```

• Podemos obligar a que se introduzcan valores de selección inevitablemente.

```
SELECT-OPTIONS <var> FOR <campo> OBLIGATORY.
```

• También es posible desactivar la posibilidad de introducir selecciones con condiciones compuestas (Desaparecerá la flecha).

```
SELECT-OPTIONS <var> FOR <campo> NO-EXTENSION.
```

• También es posible formatear a nuestro gusto la pantalla de selección con `SELECTION-SCREEN.`

Podemos introducir comentarios para un parámetro.

```
SELECTION-SCREEN COMMENT <col> (<long>) TEXT-nnn.
```

Indicándole la columna, la longitud del comentario, y el texto del comentario lo situaremos en un texto numerado.

Si además queremos que al pulsar F1 (help), sobre el comentario, aparezca la misma ayuda que sobre el campo:

```
SELECTION-SCREEN COMMENT <col> (<long>) TEXT-nnn FOR FIELD
<campo>.
```

Otras posibilidades pueden ser, intercalar líneas en blanco o subrayados en la pantalla de selección.

```
SELECTION-SCREEN SKIP <n>.

SELECTION-SCREEN ULINE <col> (<long>).
```

Es posible también utilizar varias páginas de selección con:

```
SELECTION-SCREEN NEW-PAGE.
```

• Podemos realizar verificaciones de los datos entrados en la pantalla de selección con el evento.

```
AT SELECTION-SCREEN ON <campo>.
…
ENDAT.
```

• Podemos realizar varias selecciones en la misma línea con :

```
SELECTION-SCREEN BEGIN OF LINE.
…
SELECTION-SCREEN END OF LINE.
```

En este caso no aparecen los textos de selección.

Elementos de texto y Mensajes

El entorno de desarrollo de programas en ABAP/4 nos permite manejar elementos de texto sin necesidad de codificarlos en el programa.

Los elementos de texto pueden ser títulos de *reports*, cabeceras de *reports*, textos de selección y textos numerados.

Podemos acceder a la pantalla de tratamiento de los elementos de textos desde el editor de programas: **Pasar a ->Elementos de texto**.

Con los **Títulos y Cabeceras** podemos tratar el título, cabeceras de *report* y cabeceras de columna que saldrán por pantalla e impresora.

Con los **Textos de selección** trataremos los comentarios que acompañan a los parámetros del tipo `PARAMETERS` o `SELECT-OPTIONS`.

Con los **Textos numerados** podemos utilizar constantes de tipo texto sin necesidad de declararlas en el código del programa. Los nombres de las constantes serán `TEXT-nnn`, donde **`nnn`** es un número de tres dígitos. Además podemos mantener los textos numerados en varios idiomas.

Otras de las facilidades que nos ofrece ABAP/4 para el formateo y control de *reports*, es la de los **mensajes de diálogo** .Los mensajes de diálogo son aquellos mensajes que aparecen en la línea de mensajes y que son manejables desde un programa.

Los mensajes están agrupados en áreas de mensajes. Para indicar que área de mensajes vamos a utilizar en un *report* utilizamos `MESSAGE-ID` en la instrucción `REPORT`.

```
REPORT <report> MESSAGE-ID <area>.
```

Podemos ver, crear y modificar áreas de mensajes desde el editor: **Pasar a -> Mensajes.**

Para visualizar un mensaje utilizamos la sentencia `MESSAGE.`

```
MESSAGE Tnnn.
```

Donde **nnn** es el número de mensaje dentro de su respectiva área de mensajes y **T** es el tipo de mensaje

A = Cancelación o 'Abend ' del proceso.
E = Error. Es necesaria una corrección de los datos.
I = Información. Mensaje meramente informativo. El proceso continuará con un ENTER.
S = Confirmación. Información en la pantalla siguiente.
W = Warning. Nos da un aviso. Podemos cambiar los datos o pulsar 'intro' para continuar.

Si se emiten mensajes del tipo W o E en eventos `START-OF-SELECTION` o `ENDOF-SELECTION` o `GET` se comportan como si fueran del tipo A.

Podemos acompañar los mensajes de parámetros variables.

```
MESSAGE Tnnn WITH <var1> <var2>...
```

En la posición del mensaje que se encuentre el símbolo `&`, podemos utilizar para visualizar el valor que le pasemos como parámetro a la instrucción `MESSAGE`.

No podemos utilizar más de 4 parámetros por mensaje.

Los datos sobre mensajes están en la tabla `T100`.

Ejemplo:

```
Área de mensajes ZZ.
Mensaje: 005 = Entrada &-& incorrecta.
REPORT ZPRUEBA MESSAGE-ID ZZ.
....
IF....
MESSAGE A005 WITH SKA1 KTOPL.
ENDIF.
```

El mensaje obtenido será:

A: Entrada SKA1-KTOPL Incorrecta

XI Tratamiento de ficheros desde un programa en ABAP/4

ABAP/4 dispone de una serie de instrucciones para manejar ficheros binarios o de texto.

(OPEN, CLOSE, READ, TRANSFER).

• Para abrir un fichero utilizaremos la sentencia OPEN**.**

```
OPEN DATASET <fichero>.
```

FOR OUTPUT / (INPUT) → Escritura/Lectura (por defecto)

IN BINARY MODE / IN TEXT MODE → Binario (por defecto)/Texto.

Si SY-SUBRC=0 → Fichero abierto correctamente.
SY-SUBRC=8 → Fichero no se ha podido abrir.

• Para cerrar un fichero utilizamos CLOSE.

```
CLOSE DATASET <fichero>.
```

• Si queremos leer de un fichero utilizamos READ.

```
READ DATASET <fichero> INTO <registro> (LENGTH <long>)
```
→ Guarda en <long> la longitud del registro leído.

Ejemplo:

```
DATA: BEGIN OF REC,
LIFNR LIKE LFA1-LIFNR,
BAHNS LIKE LFA1-BAHNS,
END OF REC.

OPEN DATASET '/usr/test'.

DO.
     READ DATASET '/usr/test' INTO REC.
     IF SY-SUBRC NE 0.
          EXIT.
     ENDIF.
     WRITE: /REC-LIFNR,REC-BAHNS.
ENDDO.

CLOSE DATASET '/usr/test'.
```

Nota: Se puede leer de hasta 4 ficheros simultáneamente

Si `SY-SUBRC=0` → Fichero leído correctamente.
Si `SY-SUBRC=4` → Fin de fichero encontrado.

• Para escribir sobre un fichero disponemos de la instrucción `TRANSFER`**.**

`TRANSFER <registro> TO <fichero> (LENGTH <long>)` → Transfiere la longitud especificada en la variable <long>.

Por defecto la transferencia se realiza sobre un fichero secuencial (texto) a no ser que se abra el fichero como binario.

En el caso de que el fichero no se encuentre todavía abierto, la instrucción `TRANSFER` lo intentará en modo binario y escritura.

NOTAS: