

1	¿ QUÉ ES EL ABAP/4 ? .....	3
2	CARACTERÍSTICAS DEL ABAP/4. ....	3
3	APLICACIONES DEL ABAP/4. ....	3
4	ENTORNO DE DESARROLLO EN ABAP/4. ....	4
	4.1 Creando un programa ABAP/4. ....	4
	4.2 Trabajando con el editor de programas. (Ver Anexo 1. 'SAP Editor') .....	5
5	FUNDAMENTOS DE LA PROGRAMACIÓN DE REPORTS. ...	5
	5.1 Tipos de instrucciones. ....	5
	5.2. Objetos de datos. ....	6
	5.3. Estructura de un programa. ....	7
6	DECLARANDO Y PROCESANDO DATOS .....	7
	6.1. Tipos de Campos. ....	7
	6.2. Declaración de Campos. ....	7
	6.3 Asignando valores. ....	9
	6.4 Conversión de tipo. ....	11
	6.5 Operaciones Aritméticas en ABAP/4. ....	11
	6.6 Procesando campos de tipo texto. ....	11
	6.7 Variables del sistema. ....	13
7.	CONTROL DE FLUJO EN LOS PROGRAMAS ABAP/4. ..	13
	7.1 Formulando condiciones. ....	13
	7.2 Proceso de bucles. ....	15
	7.3 Sentencias de control. ....	16
8	INTRODUCCIÓN A LAS SENTENCIAS DE SALIDA DE REPORTS. ....	16
9	TABLAS INTERNAS. ....	17
	9.1 Como declarar tablas internas. ....	17
	9.2 Llenado de una tabla interna. ....	18
	9.3 Ordenar una tabla interna. ....	18
	9.4 Procesamiento de una tabla interna. ....	18
	9.5 Tratamiento de niveles de ruptura. ....	19
	9.6 Lectura de entradas de una tabla. ....	20
	9.7 Modificando tablas internas. ....	21
10	SUBROUTINAS. ....	22
	10.1 Tipos de subrutinas. ....	22
	10.2 Subrutinas internas. ....	22
	10.3 Subrutinas Externas y Módulos de función. ....	23
	10.4 Intercambio de datos mediante la memoria global de SAP. ....	25
11	DICCIONARIO DE DATOS. COMO LEER Y PROCESAR TABLAS DE LA .....	25
	BASE DE DATOS. ....	25
	11.1 Diccionario de datos. ....	25

11.2	Los datos en el sistema SAP.....	26
11.3	Instrucciones SQL de ABAP/4.....	26
11.3.1	SELECT.....	27
11.3.2.	INSERT.....	28
11.3.3.	UPDATE.....	29
11.3.4.	MODIFY.....	29
11.3.5	DELETE.....	30
11.4	Otros aspectos de la programación de BDD.	30
12	BASES DE DATOS LÓGICAS.....	32
12.1	¿Que es una Base de datos lógica ?.....	32
12.2	Utilización de las Bases de datos lógicas. .....	33
13	FIELD-GROUPS.....	35
14	FORMATEANDO UN LISTADO.....	38
14.1	Formato de los datos de salida.....	38
14.2	Formato de página.....	40
14.3	Selección de parámetros. Pantalla de selección (SELECTION SCREEN).....	41
14.4	Elementos de texto y Mensajes.....	45
15	FIELD SYMBOLS.....	47
16	BATCH INPUTS.....	48
16.1	Introducción.....	48
16.2	Fase de generación del Batch Input.....	49
16.2.1	Sistema externo.....	50
16.2.2	El programa Batch Input.....	50
16.2.3	El fichero de colas.....	51
16.3	Fase de procesado de una sesión.....	52
16.4	Consejos prácticos en la utilización de Batch Inputs.....	53
16.5	Codificación de Batch Inputs.....	55
17	TRATAMIENTO DE FICHEROS DESDE UN PROGRAMA EN ABAP/4.....	60

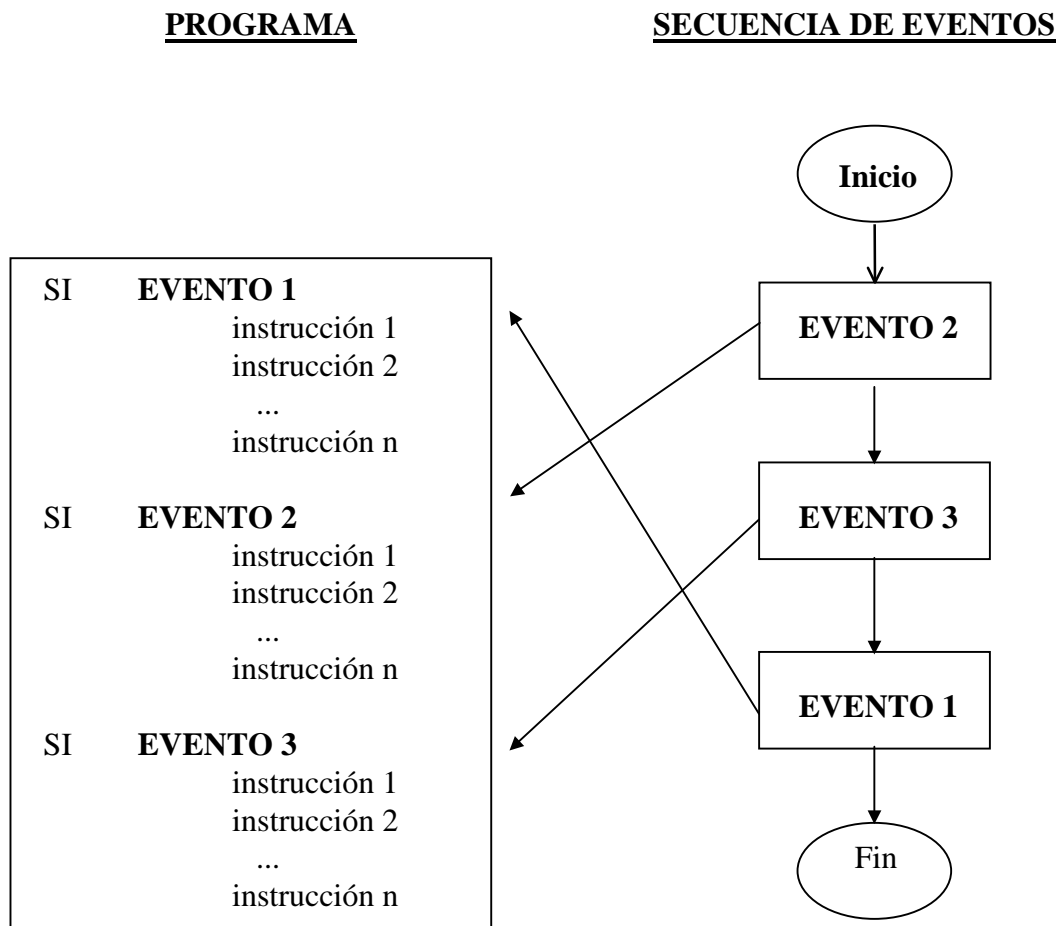
## 1 ¿ QUÉ ES EL ABAP/4 ?

(**Advanced Business Application Programming 4th Generation**).

El ABAP/4 es un lenguaje de programación de 4a. Generación (4GL) orientado tal como su definición específica, 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.

## 2 CARACTERÍSTICAS DEL ABAP/4.

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.



## 3 APLICACIONES DEL ABAP/4.

- **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 destinadas 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 tiene las siguientes características: Orientado a pantalla. Listados cortos y con datos resumidos. Informaciones detalladas en sublistados o ventanas controlado 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 los softwares de presentación (WINDOWS ...). 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.

ABAP/4 posee instrucciones para realizar programas de comunicaciones según la norma LU.6.2 (SNA).

## 4 ENTORNO DE DESARROLLO EN ABAP/4.

### 4.1 Creando un programa ABAP/4.

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

Ejemplo práctico: Creando Zxx00000 xx: iniciales

Herramientas->Case ... Desarrollo->Actualizar programas->Desarrollo ABAP/4

Introducir nombre programa. Crear.  
Introducir el título del programa.  
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).  
**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.

## 4.2 Trabajando con el editor de programas. (Ver Anexo 1. 'SAP Editor')

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 esta.

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 Diccionario de datos. La tabla TRDIR contiene información de los programas del sistema.

## 5 FUNDAMENTOS DE LA PROGRAMACIÓN DE REPORTS.

### 5.1 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.
---	--------------------	--

## 5.2. 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.
```

### 5.3. Estructura de un programa.

<b>REPORT</b> <nombre>	→	Nombre programa
<b>TABLES:</b>	→	Tablas que se utilizan
<b>DATA:</b>	→	Variables internas
<b>TOP-OF-PAGE.</b> <Sentencias>	→	Por inicio de página ejecutar las instrucciones que se indiquen.
<b>END-OF-PAGE.</b> <Sentencias>	→	Por fin de página ejecutar las instrucciones que se indiquen.
<b>START-OF-SELECTION.</b> <Sentencias>	→	Por inicio de programa ejecutar las instrucciones indicadas.
<b>END-OF-SELECTION.</b> <Sentencias>	→	Por Fin de programa ejecutar las instrucciones indicadas.

La secuencia de eventos no es relevante.

## 6 DECLARANDO Y PROCESANDO DATOS

### 6.1. Tipos de Campos.

Los tipos de datos que se pueden utilizar en ABAP /4 son:

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	Número Empaquetado
X	1	1-29870	x'00'	Hexadecimal
D	8	8	00000000	Fecha YYYYMMDD
T	6	6	000000	Hora HHMMSS

### 6.2. 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.
```

### 6.3 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 'RIVERLAND BCN.',  
        VAR2(15) VALUE 'HOLA'.  
MOVE VAR1+10(4) TO VAR2+5(4).  
WRITE VAR2.
```

Resultado:

HOLA BCN.

VAR1

R	I	V	E	R	L	A	N	D		B	C	N	.	
---	---	---	---	---	---	---	---	---	--	---	---	---	---	--

VAR2

H	O	L	A											
---	---	---	---	--	--	--	--	--	--	--	--	--	--	--

**MOVE VAR1+10(4) TO VAR2+5(4).**

VAR2

H	O	L	A		B	C	N	.						
---	---	---	---	--	---	---	---	---	--	--	--	--	--	--

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) 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 variable normales. Un **CLEAR** inicializa todos los campos del registro. Podremos conseguir una asignación mas potente con **MOVE-CORRESPONDING**.

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

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

## 6.4 Conversión de tipo.

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

Podemos ver un extracto de las posibles conversiones en el **Anexo 2** “Type conversion table”.

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>.**

## 6.5 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 , SUBTRACT 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.**

## 6.6 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.

Ejemplo:

H	O	L	A	
---	---	---	---	--

SHIFT campo BY 2 PLACES.

L	A			
---	---	--	--	--

**SHIFT <campo> BY 2 PLACES CIRCULAR (RIGHT).**

Realiza un desplazamiento cíclico hacia la izquierda o si se especifica hacia la derecha.

Ejemplo:

H	O	L	A	
---	---	---	---	--

SHIFT campo BY 2 PLACES CIRCULAR.

L	A		H	O
---	---	--	---	---

- 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.

donde la regla = <C1S1C2S2...> y Cn son los caracteres a sustituir y Sn los caracteres de sustitución.

- 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.

## **6.7 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.

En el **Anexo3** hay una relación de todas ellas.

## **7. CONTROL DE FLUJO EN LOS PROGRAMAS ABAP/4.**

### **7.1 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	= , <b>EQ</b>
Distinto	<> , <b>EN</b>
Mayor	> , <b>GT</b>
Menor	< , <b>LT</b>
Mayor o igual	>= , <b>GE</b>
Menor o igual	<= , <b>LE</b>

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.
```

## 7.2 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.

### 7.3 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.

## 8 INTRODUCCIÓN A LAS SENTENCIAS DE SALIDA DE REPORTS.

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.

En el **capítulo 14** veremos todas las posibilidades para las selecciones y entrada de parámetros.

## **9 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**.

### **9.1 Como declarar tablas internas.**

**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.

## 9.2 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>.**

## 9.3 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.**

## 9.4 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.

## 9.5 Tratamiento de niveles de ruptura.

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

<b>AT FIRST.</b> ... <b>ENDAT.</b>	→	Realiza las instrucciones que hay a continuación del AT FIRST para la primera entrada de la tabla.
<b>AT LAST.</b> ... <b>ENDAT.</b>	→	Realiza las instrucciones que hay a continuación del AT LAST para la última entrada de la tabla.
<b>AT NEW &lt;campo&gt;.</b> ... <b>ENDAT.</b>	→	Realiza las instrucciones que hay a continuación del AT NEW para cada inicio de nivel de ruptura.
<b>AT END OF &lt;campo&gt;.</b> ... <b>ENDAT.</b>	→	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.

```

Podemos ver un ejemplo práctico de tratamiento de niveles de ruptura en el **BC ABAP/4 : Programming Reports 8-17 , 8-18.**

## 9.6 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>.**

## 9.7 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>.**

## 10 SUBROUTINAS.

### 10.1 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.

### 10.2 Subrutinas internas.

**PERFORM <modulo>.**       $\longrightarrow$       Llamada a un procedimiento o subprograma.

**FORM <modulo>**  
    ...       $\longrightarrow$       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>.**

### **10.3 Subrutinas Externas y 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>.           o
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.



## 10.4 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.**

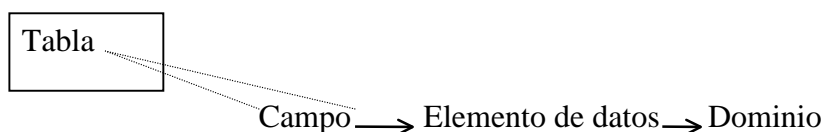
Para recuperar de memoria:

**IMPORT <campo> ... INTO MEMORY.**

## 11 DICCIONARIO DE DATOS. COMO LEER Y PROCESAR TABLAS DE LA BASE DE DATOS.

### 11.1 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 :



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 posibles. 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.

## 11.2 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.

## 11.3 Instrucciones SQL de ABAP/4.

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

Estas 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.

### 11.3.1 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 tablas 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>.**

### **11.3.2. 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.

### 11.3.3. 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 <b>SY-SUBRC</b> = 0	Todos los registros modificados.
Si <b>SY-SUBRC</b> = 4	No todos los registros han sido modificados.
En <b>SY-DBCNT</b>	Tendremos la cantidad de registros modificados.

### 11.3.4. 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.

### **11.3.5 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 <b>SY-SUBRC</b> = 0	Todos los registros han sido borrados.
Si <b>SY-SUBRC</b> = 4	No todos los registros han sido borrados.
En <b>SY-DBCNT</b>	Tendremos la cantidad de registros borrados.

### **11.4 Otros aspectos de la programación de BDD.**

- El **control del mandante** es automático. Siempre 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 'Acreeador : 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.

Para obtener información sobre el mecanismo de autorizaciones de SAP, ver el curso **CA010 El concepto de autorizaciones SAP**.

- 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.

Podemos encontrar información complementaria sobre la utilización del interface EXEC SQL en el **Cap. 1 del manual 'ABAP/4 Special Techniques'**.

## **12 BASES DE DATOS LÓGICAS.**

### **12.1 ¿Que es una Base de datos lógica ?**

Para obtener datos en un programa existen dos posibilidades:

- Programar la lectura de datos de la base de datos en el mismo programa con la instrucción SELECT.

- Dejar que otro programa de lectura (BDD lógica) lea los datos y se los proporcione en la secuencia apropiada.

En un report se pueden simultanear los dos tipos de selección de datos.



Una base de datos lógica (**LDB**) proporciona una visión lógica de las tablas físicas, pudiendo relacionar tablas entre si. Las LDB simplifican la programación de reports ofreciendo accesos de lectura, verificación de autorizaciones y selecciones estandarizadas.

La comunicación entre el programa de lectura y el report que utiliza la base de datos lógica se realiza mediante los eventos **PUT** y **GET**.

Por regla general utilizaremos bases de datos lógicas que ya existen en el sistema, aunque también es posible crear nuevas y modificarlas. (Transacción **ALDB**).

Si utilizamos LDB ya creadas en el sistema, únicamente tendremos que utilizar un evento para recoger la información que el programa de lectura (que ya existe) nos va dando.

Si por el contrario nos decidimos a crear una LDB con la transacción ALDB, el sistema generará todo lo necesario para utilizar la base de datos lógica, incluyendo el programa de lectura.

## 12.2 Utilización de las Bases de datos lógicas.

Las bases de datos lógicas tienen un nombre de tres caracteres, siendo el último carácter el módulo funcional al que va dirigido.

Ejemplo :

KDF : clientes FI

En el programa que va a utilizar bases de datos lógicas será necesario especificar en los atributos del programa la LDB que va a ser utilizada. Y en el código simplemente utilizaremos el evento **GET**.

```
GET <tablaBDD1>.  
    <sentencias evento>  
.....  
GET <tablaBDD2>.  
    <sentencias evento>  
.....
```

Mediante el GET dispondremos de un registro de la base de datos que especifiquemos, siempre y cuando esta tabla esté dentro de la estructura de la base de datos lógica.

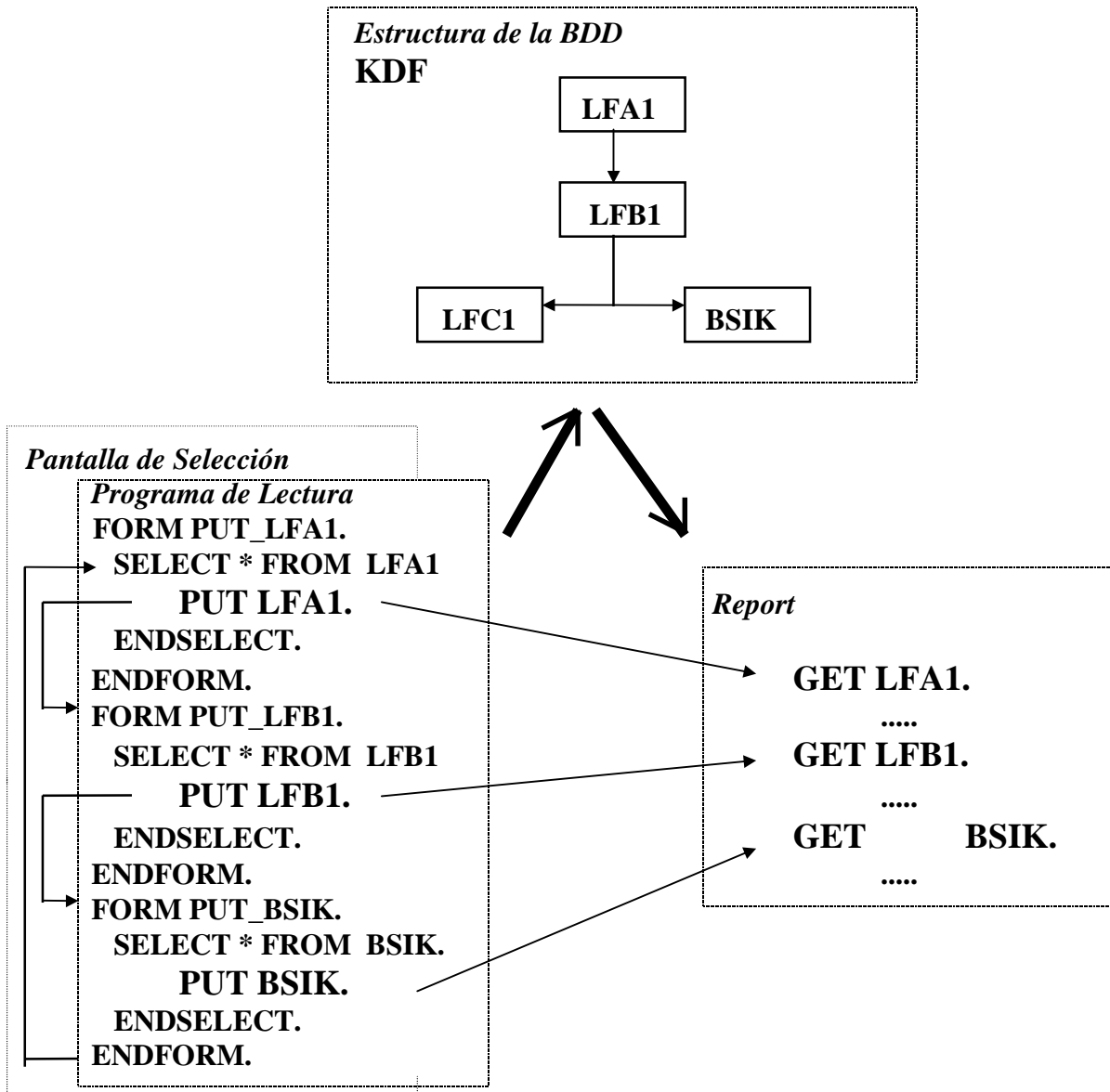
Para comunicar el programa de lectura con nuestro report se utiliza el **PUT**, que suministra el registro de la BDD que especifiquemos, previamente habrá realizado el SELECT.

```
PUT <tablaBDD>.
```

Una base de datos lógica tiene tres componentes fundamentales :

- Una definición de la **estructura de las tablas** que utiliza.
- Una **pantalla de selección** de los datos a leer. (SELECT-OPTIONS)

- Un **programa de lectura** de datos de la BDD. (PUT).



También existe la posibilidad de utilizar el evento:

**GET <tabBDD> LATE.**

...

Este evento se produce cuando se han procesado todas las entradas de tablas subordinadas a un registro de datos de una tabla, y antes de que el sistema solicite la siguiente entrada de la misma tabla (mismo nivel jerárquico).

Existe una instrucción de salto o finalización de lectura de una tabla, **REJECT**.

**REJECT.**

Esta instrucción sale del proceso del registro en curso y continua con el proceso del siguiente registro dentro del mismo nivel de jerarquía.

Si indicamos un nombre de tabla, lo que hará será continuar con el siguiente registro de la tabla especificada. <tabla> no puede ser un nivel de jerarquía más profundo que el actual.

**REJECT <tabla>.**

En principio, únicamente utilizaremos la sentencia GET , ya que utilizaremos LDB que ya existen en el sistema.

Si necesitamos crear una nueva debido a que se han de desarrollar muchos reports con una estructura de lectura muy similar, y esta no está en ninguna base de datos lógica, utilizaremos la transacción ALDB. (Para más información sobre los pasos a seguir ver **Cap: 10, 11 y 12 del BC180-ABAP/4 Interface de datos** o **Cap: 15 del ABAP/4 Programing Reports**).

### **13 FIELD-GROUPS.**

En el capítulo 9 ya vimos que cuando queremos ordenar y/o controlar las rupturas de campos en un report, es necesario utilizar las tablas internas. Sin embargo existe otra utilidad del ABAP/4 que nos facilita estos procesos de ordenación y rupturas, en el caso de que sean complejos.

Supongamos un listado en el que las líneas sean de muy distinto tipo, por ejemplo, un listado de proveedores con datos generales de este, (dirección ...) y las ventas que nos han realizado cada uno de los proveedores, ordenados por distintos campos y con subtotales. En este caso no tendremos más remedio que utilizar diversas tablas internas, una para cada tipo de línea, ordenar estas tablas internas y procesarlas adecuadamente.

Para casos como este, ABAP/4 nos ofrece la técnica especial de los **FIELD GROUPS's**.

Esta técnica consiste en crear conjuntos de datos intermedios. ('intermediate datasets').

Se definen los diferentes registros con idéntica estructura, dentro de un mismo tipo de registro (FIELD GROUP). Será necesario definir todos los FIELD GROUP al inicio del report con :

**FIELD-GROUP : HEADER, <f\_g\_1>, <f\_g\_2>...**

El FIELD GROUP **HEARDER** es fijo. Contendrá los campos por los cuales queremos ordenar el conjunto de datos intermedio.

Para determinar que campos pertenecen a cada FIELD GROUP, utilizamos la instrucción :

```
INSERT <campo1> <campo2> .....<campo_n> INTO HEADER.  
INSERT <campo1> <campo2> .....<campo_n> INTO <f_g_1>.  
....
```

Un campo podrá estar dentro de varios FIELD GROUPS.

Para llenar con datos los conjuntos de datos intermedios se utiliza la instrucción:

```
EXTRACT <f_g_1>.
```

Esta instrucción asigna los contenidos de los campos especificados en el INSERT al FIELD GROUP indicado.

En cada EXTRACT, el sistema realiza automáticamente una extracción de los datos del FIELD GROUP HEADER, estos precederán siempre a los datos del FIELD GROUP sobre el que realizamos el EXTRACT.

Datos HEADER	Datos <f_g>
--------------	-------------

Si algún campo de la cabecera no se llena, tomará el valor 0, de forma que el proceso de ordenación funcione correctamente.

Veamos el funcionamiento de los FIELD GROUP's con un ejemplo:

Para realizar un listado de partidas de proveedores, ordenado por código de proveedor y números de documentos de las diferentes partidas.

```
TABLES: LFA1,BSIK.  
FIELD-GROUPS : HEADER, DIRECCION, IMPORTES.  
INSERT LFA1-LIFNR BSIK-BELNR INTO HEADER.  
INSERT LFA1-NAME1 LFA1-STRAS LFA1-PSTLZ LFA1-ORT01  
                INTO DIRECCION.  
INSERT BSIK-DMBTR INTO IMPORTES.  
  
*-----  
GET LFA1.  
    EXTRACT DIRECCION.  
GET BSIK.  
    EXTRACT IMPORTES.  
*-----
```

En cada EXTRACT se va llenando el conjunto de datos intermedios.

EXTRACT DIRECCION	
PROVEEDOR1	RIVERLAND DIAGONAL 618 BARCELONA

## EXTRACT IMPORTE

PROVEEDOR1 DOC1	100.000
-----------------	---------

Así el dataset se irá llenando:

PROVEEDOR1	RIVERLAND DIAGONAL 618 BARCELONA
PROVEEDOR1 DOC1	100.000
PROVEEDOR1 DOC2	200.000
PROVEEDOR2	SAP A.G. PABLO PICASSO 28020 MADRID
PROVEEDOR2 DOC1	250.000
PROVEEDOR2 DOC2	1.200.000

Una vez extraídos los datos, los podemos procesar de forma similar a como lo hacíamos en las tablas internas.

En primer lugar ordenaremos el 'dataset', con la instrucción **SORT**. La ordenación se realizará por los campos que indica el **HEADER**.

Posteriormente podemos procesar los datos en un **LOOP...ENDLOOP** , Pudiendo utilizar las instrucciones de ruptura por campos **AT NEW** y **AT END OF**. También podemos utilizar estos eventos por inicio y final de registro (**FIELD-GROUP**).

Además podemos comprobar si para un registro, existen registros asociados de otro tipo, con el evento:

```
AT <f_g1> WITH <f_g2>.  
...  
ENDAT.
```

Por ejemplo: Si existen registros de importes para un registro de dirección, imprimir en el report los datos de dirección.

```
AT DIRECCION WITH IMPORTES.  
WRITE: LFA1-NAME1 .....  
ENDAT.
```

También podemos contar o sumar por campos con las instrucciones:

```
CNT (<campo>).  
SUM (<campo>).
```

Así podríamos completar nuestro listado de proveedores del ejemplo con:

```
END-OF-SELECTION.  
SORT.  
LOOP.  
AT DIRECCION WITH IMPORTES.  
WRITE: LFA1-NAME1, LFA1-STRAS,  
LFA1-PSTLZ, LFA1-ORT01.  
ENDAT.
```

```
AT IMPORTES.  
    WRITE: BSIK-BELNR, BSIK-DMBTR.  
ENDAT.  
AT END OF LFA1-LIFNR.  
    SKIP.  
    WRITE: 'Suma proveedor', LFA1-LIFNR,  
          SUM(BSIK-DMBTR).  
    SKIP.  
ENDAT.  
ENDLOOP.
```

## 14 FORMATEANDO UN LISTADO.

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

### 14.1 Formato de los datos de salida.

Ya hemos visto en el capítulo 8 un resumen de las sentencias de salida de reports más básicas.

**WRITE** /<offset>(<long>) '<datos a visualizar>'.  
El offset indica el número de columnas que se debe saltar antes de escribir los datos.

**ULINE** /<offset>(<long>) '<datos a visualizar>'.  
El offset indica el número de columnas que se debe saltar antes de escribir los datos.

**SKIP** <n>.  
Salta n líneas.

**NEW-PAGE**.  
Comienza una nueva página.

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>.  
Escribe el contenido de <campo2> justo debajo del contenido de <campo1>.

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

**POSITION** <columna>.  
Escribe el texto a partir de la columna indicada.

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

**SKIP TO LINE** <n>.  
Salta a la línea 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_Character> 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.

## 14.2 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 **NEW-PAGE**.

- 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 instrucción 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.

## **14.3 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 cualquier tipo en la pantalla de selección.

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

\* En el capítulo 8 ya vimos 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 :

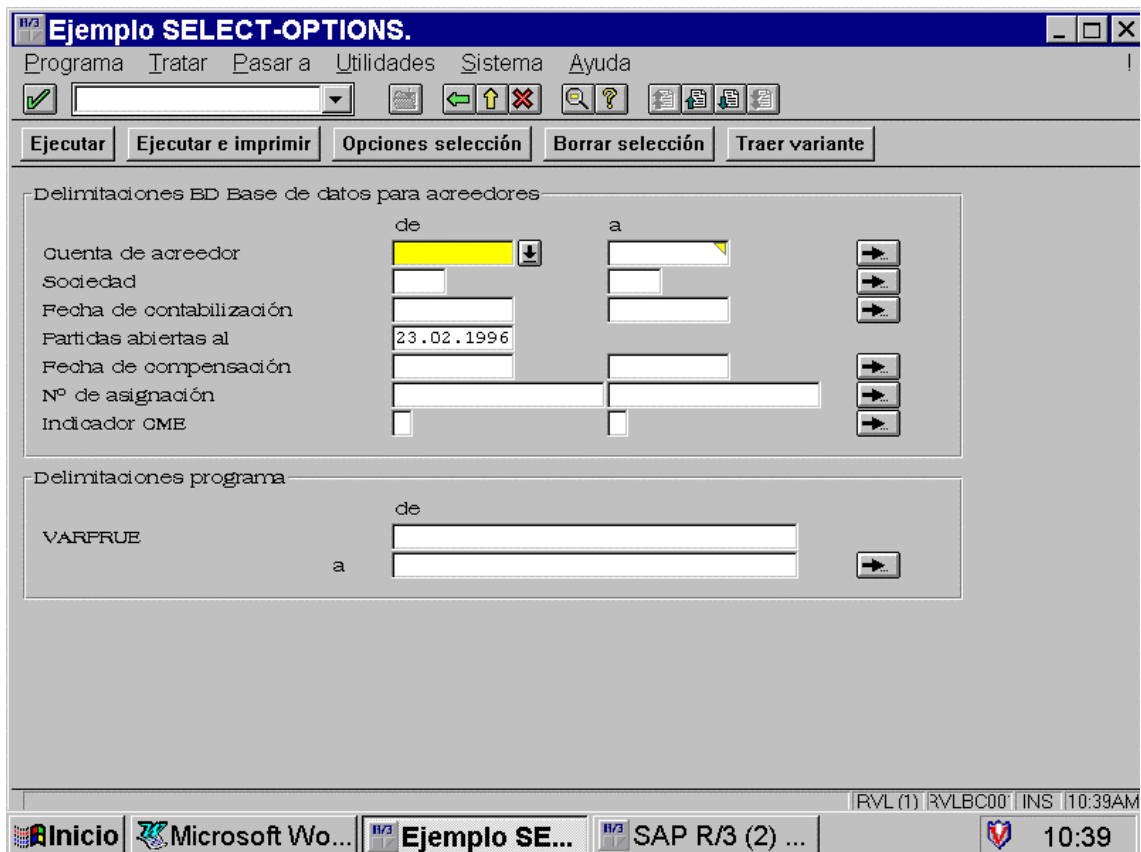
**SELECT-OPTIONS <var> FOR <campo\_tabla>.**

**<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.

Ejemplo :



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 (Incluido/Excluido) 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 Click 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 **SELECT-OPTIONS**.

- 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 (ver 13.4).

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.

### **14.4 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:

<b>A</b>	= Cancelación o 'Abend' del proceso.
<b>E</b>	= Error. Es necesaria una corrección de los datos.
<b>I</b>	= Información. Mensaje meramente informativo. El proceso continuará con un ENTER.
<b>S</b>	= Confirmación. Información en la pantalla siguiente.
<b>W</b>	= 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 END-OF-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*

## 15 FIELD SYMBOLS.

Cuando tenemos que procesar una variable, pero únicamente conocemos de que variable se trata y como tenemos que procesarla, en tiempo de ejecución, lo haremos mediante los 'field symbols'. Por ejemplo, si estamos procesando cadenas, y queremos procesar una parte de la cadena cuya posición y longitud depende del contenido de la misma, utilizaremos 'field symbols'. Los Field Symbol tienen cierta similitud con los punteros o apuntadores de otros lenguajes de programación.

Los Field Symbol permiten soluciones elegantes a problemas pero su utilización incorrecta puede implicar resultados impredecibles.

Los Field Symbol se declaran con:

**FIELD-SYMBOLS : <«Field Symbol»>.**

La declaración se realizará en la rutina o módulo de función donde se utilice.


Para asignar un campo a un 'Field Symbol' utilizaremos la instrucción ASSIGN. Una vez asignado, cualquier operación que realicemos sobre el field symbol afectará al campo real. No hay ninguna diferencia entre utilizar el campo o el field symbol.

**ASSIGN <campo> TO <«Field Symbol»>.**

Ejemplos :

1.-


```
FIELD-SYMBOLS <F>.  
ASSIGN TRDIR-NAME TO <F>.  
MOVE 'ZPRUEBA' TO <F>.  
WRITE TRDIR-NAME.
```



'ZPRUEBA'

2.-

```
FIELD-SYMBOLS <F>.  
TEXT0 = 'ABCDEFGH' .  
INICIO = 2 .  
LONGITUD = 5 .  
ASSIGN TEXT0+INICIO(LONGITUD) TO <F>.  
WRITE <F>.
```



'CDEFG'

3.-

```
* Rellena con ceros por la izquierda.  
FORM PONER_CEROS USING NUMERO VALUE(LONGITUD).  
FIELD-SYMBOLS: <PUNTERO>.  
LONGITUD = LONGITUD - 1.  
ASSIGN NUMERO+LONGITUD(1) TO <PUNTERO>.  
WHILE <PUNTERO> EQ SPACE.  
    SHIFT NUMERO RIGHT.
```

```
WRITE '0' TO NUMERO(1).  
ENDWHILE.  
ENDFORM.
```


También es posible utilizar asignación dinámica. Esto permite asignar un campo que sólo conocemos en tiempo de ejecución a un field symbol.

Será necesario encerrar el campo entre paréntesis en la asignación del field symbol.

**ASSIGN (<campo>) TO <<Field Symbol>>.**

Ejemplo:

```
DATA: CAMPO(10).  
FIELD-SYMBOLS: <F>.  
MOVE 'TRDIR-NAME' TO CAMPO .  
ASSIGN (CAMPO) TO <F>.  
WRITE <F>.
```



Para información adicional sobre los Field Symbols ver **Cap: 10 del ABAP/4 Programing Reports**.

## 16 BATCH INPUTS.

### 16.1 Introducción.

Cuando se instala una aplicación en productivo es necesario dar de alta toda la información indispensable para que la empresa pueda funcionar (proceso de migración de datos o conversión).

Por ejemplo, antes de poder generar facturas reales será necesario introducir todos los clientes activos y todos los productos que están a la venta.

Para realizar la carga de productos que están a la venta se debería ejecutar manualmente la transacción '*Alta de material*' tantas veces como productos tengamos y la misma operación con '*Alta de clientes*' para todos los clientes. En el caso de que la empresa tenga muchos productos y muchos clientes, la carga inicial será muy costosa.

Generalmente todos estos datos maestros (clientes, materiales, proveedores,...) ya están en el antiguo sistema informático. Por lo tanto lo ideal será disponer un mecanismo que nos permitiese trasladar los datos de un sistema a otro.

A la hora de la migración de datos de un sistema externo a SAP, tenemos dos posibilidades:

- Realizar programas que llenen todas las bases de datos SAP involucradas, mediante instrucciones directas de SAP-SQL .
- Utilizar la **técnica del Batch Input de SAP**.

Para muchas transacciones, la primera de las opciones es inviable, debido a la complejidad de la estructura de datos SAP y para mantener la integridad de la misma, la



cantidad de validaciones que se deberían realizar sobre los datos de entrada sería enorme. Como consecuencia, tanto el coste en diseño, codificación y pruebas sería altísimo.

En cambio, la técnica de los Batch Input de SAP nos permite realizar todas las verificaciones automáticamente, con un coste en diseño y desarrollo mínimo. En este capítulo veremos como utilizar la técnica de los Batch Input.

Un Batch Input es un método **seguro y fiable** de transferir datos hacia un sistema SAP. Suele utilizarse cuando deben realizarse un elevado número de altas , modificaciones o borrados.

Para garantizar la integridad del sistema, los datos son sometidos a los mismos controles de validación y a las mismas operaciones de base de datos SAP, como si fueran introducidos manualmente y uno por uno, por el usuario. Es decir realmente la técnica del Batch Input consiste en simular repetidamente un proceso online (transacción), durante un proceso Batch.

El proceso de carga de datos se realiza en dos fases:

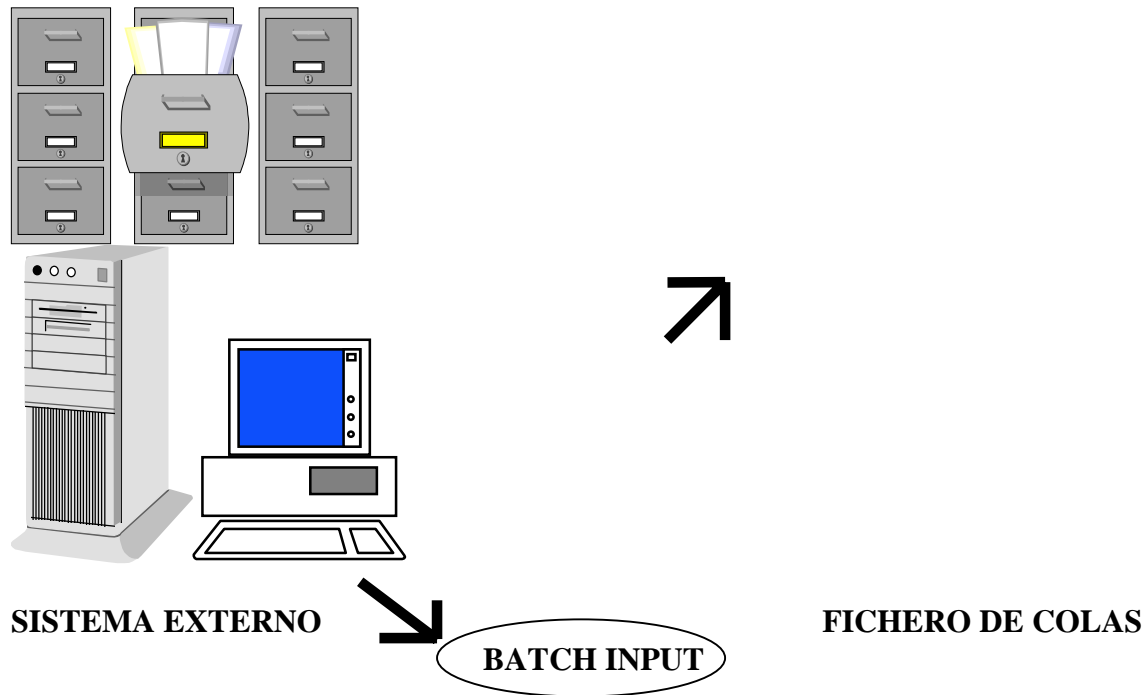
**Fase de Generación** : A partir de una fuente de información, como puede ser un fichero de entrada, donde estarán todos los datos que queremos cargar en SAP, se transformarán estos datos en un formato determinado, para almacenarlo en una estructura de SAP que llamaremos fichero de colas.

**Fase de Proceso** : A partir de la información grabada durante la fase de generación en el fichero de colas, se cargarán los datos físicamente en la base de datos.

Con la técnica del Batch Input, se realiza una simulación del diálogo del usuario con la máquina, es decir haremos exactamente lo mismo con la única diferencia de que la entrada de datos en vez de ser manual, será automática a partir de un fichero de colas.

## **16.2 Fase de generación del Batch Input.**

En esta fase se realiza la transferencia de los datos de un sistema externo a un fichero de colas. Para ello se debe codificar un programa de Batch Input.



Veamos cada uno de estos elementos:

#### 16.2.1 Sistema externo.

La extracción de los datos de un **sistema externo** suele ser realizada por el departamento de informática de la empresa donde va a ser instalado SAP, ya que es quien mejor conoce la estructura de su actual sistema informático. Normalmente el resultado final de esta extracción de datos será un fichero secuencial con los datos necesarios para cargar en SAP. El programa Batch Input, leerá este fichero y transformará los datos a un formato determinado para poder almacenarlos en el fichero de colas.

El fichero secuencial tendrá una estructura de registro que deberá ser conocida por el equipo de desarrollo de SAP. Generalmente, y siempre que sea posible, se asociará un registro a una transacción de datos SAP. Por ejemplo en el caso de altas de materiales, en un registro se guardarán todos los datos necesarios para dar de alta un único material.

Por regla general, el sistema externo es un fichero secuencial en el que se encuentran los datos con los que se desean simular las transacciones. No obstante no tiene que ser necesariamente un fichero secuencial, sino que puede ser cualquier fuente de información que tengamos (tablas físicas de SAP, tablas de otras bases de datos relacionales, etc...).

#### 16.2.2 El programa Batch Input.

Es el único desarrollo que se debe hacer en ABAP/4.

El **programa de Batch Input** leerá el fichero secuencial y transformará los datos a un formato determinado, para almacenarlos en una entrada del fichero de colas. Dichas entradas se denominan **sesiones**. Cada programa de Batch Input genera una sesión. Estas sesiones pueden contener una o múltiples transacciones.

Una transacción en SAP consta de una serie de pasos de diálogo. El programa de Batch Input debe preparar los datos para cada uno de los pasos de diálogo de la transacción.

Por ejemplo, imaginemos que para dar de alta un material el sistema ejecuta una transacción de tres pantallas:

*Pantalla 1:* Entrada de los datos sobre el diseño de material (peso, altura, volumen...).

*Pantalla 2:* Entrada de los datos sobre ventas del material (precio, descuentos...).

*Pantalla 3:* Entrada de los datos sobre la producción (costes, almacenaje...).

El programa que genere la sesión de altas de materiales deberá por tanto, programar la secuencia de acciones y pantallas en el mismo orden que la transacción y preparar los datos en cada una de estas pantallas, para cada material que se quiera dar de alta. Por ello antes de programar un Batch Input es necesario un conocimiento exhaustivo de la transacción que se desea simular, puesto que ganaremos mucho tiempo si estudiamos previamente el funcionamiento de esta.

Como se codifica un Batch Input lo veremos más adelante.

El resultado de esta etapa será una sesión de Batch Input grabada en un fichero y que posteriormente deberá procesarse para cargar físicamente los datos en el sistema SAP.

### 16.2.3 El fichero de colas.

Todos los programas Batch Input graban entradas (sesiones) en el **fichero de colas**. Para posteriormente poder identificar cual es la sesión que nos interesa procesar, las sesiones poseen un formato determinado:

Nombre de la sesión.  
Usuario que ha creado la sesión.  
Mandante en el que debe procesarse  
Número de transacciones que contiene.  
Número de pantallas que contiene.  
Datos adicionales.

Una sesión de Batch Input puede encontrarse en uno de los siguientes estados.

■ **A procesar** : Si la sesión todavía no ha sido procesada.

- **Procesada** : Si las transacciones que componen la sesión han sido ejecutadas íntegramente sin errores.
- **Erróneas** : Si en la sesión aún quedan transacciones que no se han procesado correctamente. Cuando una sesión está en estado incorrecto, no quiere decir que las transacciones que contenía no hayan sido procesadas, sino que algunas se han procesado y otras no. Estas transacciones erróneas las podremos reprocesar más adelante, es decir nunca perdemos una transacción a no ser que explícitamente borremos la sesión.
- **Siendo creada** : Si hay un programa Batch Input que está generando una sesión en ese momento.
- **En proceso** : Si se está procesado en ese instante la transacción.
- **Fondo** : Si se ha lanzado la sesión para que se procese pero todavía no ha comenzado a ejecutar por falta de recursos del sistema.

### 16.3 Fase de procesado de una sesión.

Para gestionar el fichero de colas utilizaremos la transacción **SM35 (Sistema -> Servicios -> Batch Input -> Tratar)**.

Mediante esta transacción podemos consultar, eliminar y procesar todas las sesiones de Batch Input.

Una vez generada la sesión con el programa Batch Input, accederemos a la transacción SM35 y marcaremos la sesión que nos interesa procesar.

Existen 3 tipos de procesamiento:

**Procesar visible.**

**Procesar visualizando sólo errores.**

**Procesar en invisible.**

Durante la ejecución de una sesión se irá grabando en un 'log' de proceso, el resultado de cada transacción. Entre la información que nos ofrece el log destaca:

- Hora de inicio de proceso de la sesión.
- Hora de inicio de proceso de cada transacción.
- Mensajes de incidencia o de proceso correcto. (los mismos que daría la transacción en el caso de ejecutarla manualmente).
- Estadística final de proceso :
  - Nº Transacciones leídas.
  - Nº Transacciones procesadas con éxito.
  - Nº Transacciones erróneas.

Siempre que existan transacciones con errores se podrán reprocesar.

- **Procesamiento Visible** : Con este método se procesa cada una de las transacciones visualmente, es decir, el usuario va visualizando todas y cada una de las pantallas que

hemos programado. El usuario únicamente debe ir pulsando <intro> para saltar de una pantalla a otra. Asimismo, si se cree conveniente, se permite modificar los valores de algún campo de la pantalla.

Si una transacción no interesa procesarla, podemos cancelarla (pudiendo ser ejecutada con posterioridad) o podemos borrarla (no se podrá ejecutar). Todas las transacciones que cancelemos se grabarán en la sesión y la sesión pasará a estar en estado incorrecto.

No devuelve el control del sistema al usuario hasta que todas las transacciones hayan sido procesadas o cancelemos el Batch Input.

- **Procesamiento Invisible :** El sistema procesará en modo batch la transacción. Es decir toda la ejecución es transparente al usuario. El usuario recupera el control del sistema inmediatamente. Para ver el resultado de la ejecución de una sesión, tendrá que ver el 'log' de proceso una vez haya finalizado.
- **Procesamiento visualizando sólo errores :** El sistema procesará cada una de las transacciones en modo invisible hasta que detecte un error, en cuyo caso parará el proceso en la pantalla donde se ha producido el error, pudiendo entonces el usuario detectar y corregir dicho error o cancelar la transacción. Una vez corregido el error o cancelada la transacción, el sistema continua procesando el resto de transacciones.

No devuelve el control del sistema al usuario hasta que todas las transacciones hayan sido procesadas o cancelemos el Batch Input.

#### 16.4 Consejos prácticos en la utilización de Batch Inputs.

- Para conocer el código de la transacción, el nombre de las pantallas de cada transacción y los nombres de los campos que se desean completar haremos lo siguiente :
  - Código de la transacción : Entrar en la transacción a simular e ir a **Sistema -> Status**.
  - Nombre de la pantalla : Una vez estamos en la pantalla que necesitamos, hacemos lo mismo que en el punto anterior, anotando el programa (Dynpro) y el número de dynpro.
  - Nombre de los campos : Una vez situados sobre el campo en cuestión, pulsar F1 y seguidamente el botón de datos técnicos. Anotaremos el nombre de la tabla de base de datos y del campo.
- Es posible que mientras se está procesando una sesión de Batch Input, el sistema caiga, provocando la pérdida de la misma. Cuando el sistema vuelva a la situación normal, la sesión aparentemente se encuentra en estado *Procesando* . En realidad esto no es cierto ya que la sesión no está haciendo nada, pero tampoco hemos perdido nada. El sistema habrá ejecutado todas las transacciones hasta el momento de la caída, y podemos recuperar de una manera segura el resto de la sesión de la siguiente forma:

Desde la transacción SM35, marcar la sesión de Batch Input es cuestión. Elegir **Juego de datos -> Liberar** . En ese momento la sesión pasa a modo *a procesar* y podemos ejecutar las transacciones que faltaban.

- Antes de procesar una sesión de Batch Input podemos comprobar si los datos de entrada y la secuencia de pantallas que hemos programado es la esperada. Para ello desde la SM35 seleccionaremos la sesión que queremos analizar y haremos :

**Pasar a -> Análisis -> Juego de datos.**

- Si se está ejecutando una transacción en modo Invisible, podemos ir viendo el 'Log' de proceso de las transacciones que se van ejecutando. Una utilidad práctica es, en el caso de un elevado número de transacciones, mirar el tiempo de proceso de una transacción y extrapolar este dato para todo el proceso, para tener una idea de la hora en la que finalizará el proceso.
- Antes de realizar un programa de Batch Input es aconsejable asegurarse de que SAP no disponga ya del mismo. Por ejemplo SAP nos ofrece bastantes Batch Inputs para carga de datos. Por ejemplo :
  - Carga de clientes.
  - Carga de proveedores.
  - Carga de documentos contables.
  - Carga de pedidos pendientes.
  - Carga de condiciones.
  - Carga de stocks...
- Nótese que entre la fase de generación y la fase de procesado, existe un tiempo indeterminado. Si este tiempo es muy grande, es posible que durante la fase de procesado se produzcan numerosos errores, ya que es posible que haya cambiado el estado en el que se llevo a cabo la fase de generación.

Por ejemplo, Si generamos una sesión de Batch Input donde se intenta modificar un cierto material, y antes de que se mande procesar esta sesión, el material se da de baja, durante la ejecución de la sesión el sistema se quejará de que dicho material no existe.

- Otra posible causa de errores muy común durante el procesamiento de sesiones, es que en aquellos campos que tienen tablas de verificación, introduzcamos valores que no estén dados de alta en las tablas de verificación. Por ejemplo, si indicamos una sociedad que no está en la tabla T001 (sociedades).
- Otra manera de lanzar sesiones de Batch Input es ejecutando el report **RSBDCSUB**. Por ejemplo podemos ejecutar la sesión de Batch Input inmediatamente después de ser generada, llamando a este report con los parámetros adecuados desde el mismo programa ABAP/4 que genera la sesión.

## 16.5 Codificación de Batch Inputs.

Hasta ahora hemos visto que la técnica del Batch Input consiste en la generación de una sesión con los datos a introducir en el sistema y el procesamiento de los datos en el sistema destino. En este apartado veremos como codificar el Batch Input para generar sesiones de este tipo y otras dos técnicas más de Batch Input (CALL TRANSACTION y CALL DIALOG).

Para introducir los valores en las distintas pantallas de cada transacción utilizaremos una tabla interna con una estructura estándar. (BDCDATA).

```
DATA: BEGIN OF <tab_B_I> OCCURS <n>.  
      INCLUDE STRUCTURE BDCDATA.  
END OF <tab_B_I>.
```

Los campos que componen esta tabla interna son:

- **PROGRAM** : Nombre del programa donde se realiza el tratamiento de cada pantalla (Dynpro) de la transacción.
- **DYNPRO** : Número de la pantalla de la cual queremos introducir datos.
- **DYNBEGIN** : Indicador de que se inicia una nueva pantalla.
- **FNAM** : Campo de la pantalla. (35 Caracteres como máximo).
- **FVAL** : Valor para el campo de la pantalla. (80 Caracteres como máximo).

Obtendremos la información del nombre del programa y el nombre del dynpro con **Sistema -> Status**.

Obtendremos el nombre del campo con **F1** (Datos Técnicos) o podemos ver todos los campos de una pantalla con el screen painter (Field list).

En esta tabla interna grabaremos un registro por cada campo de pantalla que informemos y un registro adicional con la información de cada pantalla.

El primer registro de cada pantalla, en la tabla interna BDCDATA, contendrá los datos que identifican la pantalla: Nombre del programa (PROGRAM), nombre de la pantalla (DYNPRO), y un indicador de inicio de dynpro (DYNBEGIN).

Ejemplo:      Transacción : FSS1  
                 Programa : SAPMF02H  
                 Dynpro : 0102

```
BDCDATA-PROGRAM = 'SAPMF02H'.  
BDCDATA-DYNPRO = '0102'.  
BDCDATA-DYNBEGIN = 'X'.  
APPEND BDCDATA.
```

Seguidamente para cada campo de la pantalla que informemos, grabaremos un registro rellenando únicamente los campos FNAM (con el nombre del campo de pantalla) y FVAL (con el valor que le vamos a dar).

Ejemplo :      Rellenar campo RF02H-SAKNR con variable VAR\_CTA.  
                  Rellenar campo RF02H-BUKRS con variable VAR\_SOC.

```
CLEAR BDCDATA.  
BDCDATA-FNAM = 'RF02H-SAKNR'.  
BDCDATA-FVAL = VAR_CTA.  
APPEND BDCDATA.
```

```
CLEAR BDCDATA.  
BDCDATA-FNAM = 'RF02H-BUKRS'.  
BDCDATA-FVAL = VAR_SOC.  
APPEND BDCDATA.
```

El programa Batch Input tiene que formatear los datos tal y como lo haría el usuario manualmente. Teniendo en cuenta que :

- Sólo se permiten caracteres.
- Los valores han de ser de menor longitud que la longitud de los campos.
- Si los valores de entrada son de longitud menor que el campo SAP, tendremos que justificar a la izquierda.

Si necesitamos informar campos que aparecen en pantalla en forma de tabla, tendremos que utilizar índices para dar valores a cada línea de pantalla y grabar en la tabla interna un registro por cada línea de pantalla.

Ejemplo :

```
CLEAR BDCDATA.  
BDCDATA-FNAM = 'campo(índice)'.  
BDCDATA-FVAL = 'valor'.  
APPEND BDCDATA.
```

Si necesitamos proveer de una tecla de función a la pantalla, usaremos el campo **BDC\_OKCODE**. El valor del campo será el número de la tecla de función precedido de una barra inclinada.

Ejemplo :

```
CLEAR BDCDATA.  
BDCDATA-FNAM = BDC_OKCODE.  
BDCDATA-FVAL = '/13'.                      “ F13 = Grabar.  
APPEND BDCDATA.
```

También utilizamos el campo BDC\_OKCODE para ejecutar funciones que aparecen en la barra de menús. Para saber el código de la función, Pulsar **F1** sin soltar el botón del ratón, sobre el menú deseado.



Si necesitamos colocar el cursor en un campo en particular, usaremos el campo **BDC\_CURSOR**. El valor del campo será el nombre del campo donde nos queremos situar.

Ejemplo :

```
CLEAR BDCDATA.  
BDCDATA-FNAM = BDC_CURSOR.  
BDCDATA-FVAL = 'RF02H-BUKRS'.  
APPEND BDCDATA.
```

Para insertar sesiones en la cola de Batch Input, seguiremos los siguientes pasos en la codificación :

**1.-** Abrir la sesión de Batch Input utilizando el módulo de función **BDC\_OPEN\_GROUP**.

**2.-** Para cada transacción de la sesión :

**2.a.-** Llenaremos la tabla **BDCDATA** para entrar los valores de los campos en cada pantalla de la transacción.

**2.b.-** Transferir la transacción a la sesión, usando el módulo de función **BDC\_INSERT**.

**3.-** Cerrar la sesión usando **BDC\_CLOSE\_GROUP**.

A continuación veremos como funcionan los módulos de función que necesitamos para generar un Batch Input.

- **BDC\_OPEN\_GROUP** : Este módulo de función nos permite abrir una sesión. En el programa no podemos abrir otra sesión hasta que no se hayan cerrado todas las sesiones que permanezcan abiertas.

**CALL FUNCTION 'BDC\_OPEN\_GROUP'**  
**EXPORTING**

<b>CLIENT</b>	= <mandante>
<b>GROUP</b>	= <nombre_sesión>
<b>HOLDDATE</b>	= <fecha>
<b>KEEP</b>	= <indicador>
<b>USER</b>	= <usuario>

**EXCEPTIONS**

<b>CLIENT_INVALID</b>	= 01
<b>DESTINATION_INVALID</b>	= 02
<b>GROUP_INVALID</b>	= 03
<b>HOLDDATE_INVALID</b>	= 04
<b>INTERNAL_ERROR</b>	= 05
<b>QUEUE_ERROR</b>	= 06
<b>RUNNING</b>	= 07.

Donde:

**CLIENT** : Es el mandante sobre el cual se ejecutará la sesión de Batch Input, si no se indica este parámetro se tomará el mandante donde se ejecute el programa de generación de la sesión.

**GROUP** : Nombre de la sesión de Batch Input, con la que identificaremos el juego de datos en la transacción SM35 de tratamiento de Bath Input.

**HOLDDATE** : Si indicamos este parámetro, el sistema no permitirá ejecutar la sesión hasta que no sea la fecha indicada. Sólo el administrador del sistema podrá ejecutar una sesión antes de esta fecha.

**KEEP** : Si informamos este parámetro con una 'X', la sesión será retenida en el sistema después de ser ejecutada y sólo un usuario con autorizaciones apropiadas podrá borrarla.

**USER** : Es el usuario que de ejecución de la sesión.

- **BDC\_INSERT** : Este módulo de función inserta una transacción en la sesión de Batch Input.

#### **CALL FUNCTION 'BDC\_INSERT'**

##### **EXPORTING**

**TCODE** = <Transacción>

##### **TABLES**

**DYNPROTAB** = <Inttab>

##### **EXCEPTIONS**

**INTERNAL\_ERROR** = 01

**NOT\_OPEN** = 02

**QUEUE\_ERROR** = 03

**TCODE\_INVALID** = 04.

Donde :

**TCODE** : Es el código de la transacción que vamos a simular.

**DYNPROTAB** : Es la tabla interna, con estructura BDCDATA, donde especificamos la secuencia de pantallas de la transacción y los distintos valores que van a tomar cada campo que aparece en cada pantalla.

- **BDC\_CLOSE\_GROUP** : Con esta función cerraremos la sesión una vez ya hemos transferido todos los datos de las transacciones a ejecutar.

#### **CALL FUNCTION 'BDC\_CLOSE\_GROUP'**

##### **EXCEPTIONS**

**NOT\_OPEN               = 01**  
**QUEUE\_ERROR       = 02.**

Podemos resumir las características de la técnica de las sesiones de Batch Input :

- Procesamiento retardado (asíncrono).
- Transferencia de datos a múltiples transacciones.
- Actualización de la base de datos inmediata (síncrona). No se ejecuta una nueva transacción hasta que la anterior no actualiza los datos en base de datos.
- Generación de un 'Log' para cada sesión.
- Imposibilidad de generar varias sesiones simultáneamente desde un mismo programa.

Como ya hemos citado anteriormente existen otras dos técnicas de Batch Input, el **CALL TRANSACTION** y el **CALL DIALOG**. En ambas, a diferencia de la técnica de sesiones, la ejecución de las transacciones es inmediata, es decir la ejecución de las transacciones es controlada por nuestro programa ABAP/4 y no posteriormente desde la SM35 lo cual puede resultar interesante en ciertas ocasiones.

• **CALL TRANSACTION :**           Características :

- Procesamiento síncrono.
- Transferencia de los datos a una única transacción.
- Actualización de la base de datos síncrona y asíncrona. El programa decide que tipo de actualización se realizará.
- La transacción y el programa que la llama tendrán áreas de trabajo (LUW) diferentes. El sistema realiza un COMMIT WORK inmediatamente después del CALL TRANSACTION.
- No se genera ningún 'Log' de ejecución.

Como se utilizará la técnica del CALL TRANSACTION :

En primer lugar llenaremos la tabla BDCDATA de la misma manera que hemos explicado a lo largo de este capítulo.

Usar la instrucción CALL TRANSACTION para llamar a la transacción.

**CALL TRANSACTION <transaccion>**  
**USING        <tabint>**  
**MODE        <modo\_ejec>**  
**UPDATE     <tipo\_actual>.**

Donde :

**<tabint>**           Tabla interna (con estructura BDCDATA).

**<modo\_ejec>**       Modo de ejecución.

Puede ser :    **'A'**    Ejecución visible.

**'N'**    Ejecución invisible. Si ocurre algún error en la ejecución de la transacción el código de retorno será distinto de cero.

**‘E’** Ejecución visualizando sólo errores.

**<tipo\_actual>** Tipo de actualización en la base de datos.

Puede ser : **‘S’** Actualización Síncrona. (inmediata).

**‘A’** Actualización Asíncrona. (hasta que no termina la transacción no graba en BDD).

Después del CALL TRANSACTION podemos comprobar si SY-SUBRC es 0, en cuyo caso la transacción se habrá ejecutado correctamente. En caso contrario, SAP llena otros campos del sistema que contienen el número, identificación, tipo y variables del mensaje online que haya emitido la transacción en el momento del error.

SY-MSGID = Identificador de mensaje.

SY-MSGTY = Tipo de mensaje (A,E,I,W...)

SY-MSGNO = Número de mensaje.

SY-MSGV1...SY-MSGV4 = Variables del mensaje.

De modo que para ver que ha ocurrido podemos ejecutar la instrucción:

MESSAGE ID SY-MSGTY

TYPE SY-MSGTY NUMBER SY-MSGNO

WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.

- **CALL DIALOG** : Características :

- Procesamiento síncrono.
- Transferencia de los datos a una única transacción.
- La transacción y el programa tendrán la misma área de trabajo (LUW). Es decir hasta que en el programa no se realiza un COMMIT WORK no se actualiza la base de datos.
- No se genera ningún ‘Log’ de ejecución.

Como se utilizará la técnica del CALL DIALOG :

Llenaremos la tabla BDCDATA.

Usar la instrucción CALL DIALOG para llamar a la transacción.

**CALL DIALOG <Mod\_diálogo>**

**USING** **<tabint>**

**MODE** **<modo\_ejec>.**

## **17 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**).

Una utilidad típica de estas instrucciones, como ya hemos explicado en el capítulo anterior, será para las interfaces entre otros sistemas y SAP, vía Batch Input.



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.