Curso

ABAP/4

Indíce

Parte I.- Introducción a SAP R/3

1- Introducción a SAP R/3

¿Qué es SAP R/3?
Componentes de SAP R/3
Aplicaciones Funcionales
Sistema Básico
Sistema Operativo

2.- **Características de SAP R/3**

Sistema Cliente-Servidor
Tecnología de Sistemas Abiertos
Integración de Aplicaciones
Entorno de desarrollo
Herramientas para la configuración del sistema
Servicio de Soporte, Formación, consultoría e implantación (OSS)
Euro y Año 2000

3.- **Entorno de Desarrollo ABAP/4**

Programación
Navegación
Debugging
Organización del desarrollo
Concepto de Mandante
Concepto de Transacción

4.- **Aplicaciones del ABAP/4**

¿Qué es ABAP/4?
Reporting (Clásico e Interactivo)
Programación de diálogo o transacciones (diseño de pantalla)
Otras aplicaciones
Modularización

Parte II.- Introducción a ABAP/4

5.- **Fundamentos de la programación de Reports**

5.1 Tipos de Instrucciones
5.2 Objetos de datos
5.3 Estructura de un programa

6.- Declarando y procesando datos

6.1 Tipos de campos
6.2 Declaración de campos
6.3 Asignando valores

Introducción a SAP R/3

1 Introducción a SAP R/3

SAP R/3 es un sistema empresarial integrado diseñado para ayudar a las organizaciones a ejecutar procesos empresariales, como gestionar inventarios, crear solicitudes,

procesar pedidos de venta, pagar facturas, etc. SAP R/3 abarca un amplio espectro de procesos empresariales.

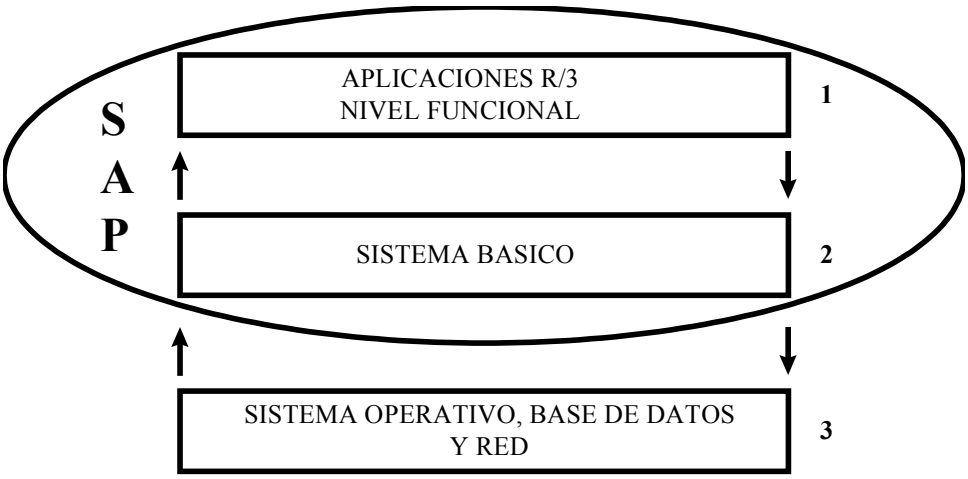
SAP R/3 proporciona un sistema único integrado de gestión de las necesidades común a todos los departamentos de una corporación. Esa integración es la gran ventaja que aporta SAP R/3. Además, como SAP R/3 es un sistema basado en cliente-servidor, su versatilidad es aún mayor.

SAP R/3 consiste en una seria de áreas de aplicación las cuales estudiaremos más adelante.

Desde el punto de vista funcional y de su arquitectura técnica SAP R/3 puede definirse como un software abierto basado en la tecnología cliente-servidor, diseñado para manejar las necesidades de información de una empresa.

Se trata de un paquete de software estándar (en contraposición al desarrollo a medida) que puede modelar los procesos de negocios de una empresa en su propio modelo de datos.

Los niveles o componentes del SAP R/3 y la función que realizan están representados dentro de la elipse.



Componentes de SAP R/3

1 .- Aplicaciones funcionales, entre las cuales mencionaremos las siguientes:

FI - Modulo de Contabilidad.
IM - Modulo de Gestión de Inversiones.
CO - Modulo de Costes.
HR - Modulo de Recursos Humanos.
SD - Modulo de Ventas y Distribución.
MM - Modulo de Gestión de Materiales
PP - Modulo de Producción.
PS - Modulo de Proyectos.
TR - Modulo de Tesorería.

QM - Modulo de Calidad.
PM - Modulo de Mantenimiento.
IS - Modulo de Soluciones Sectoriales.
WF - Modulo de Work Flow

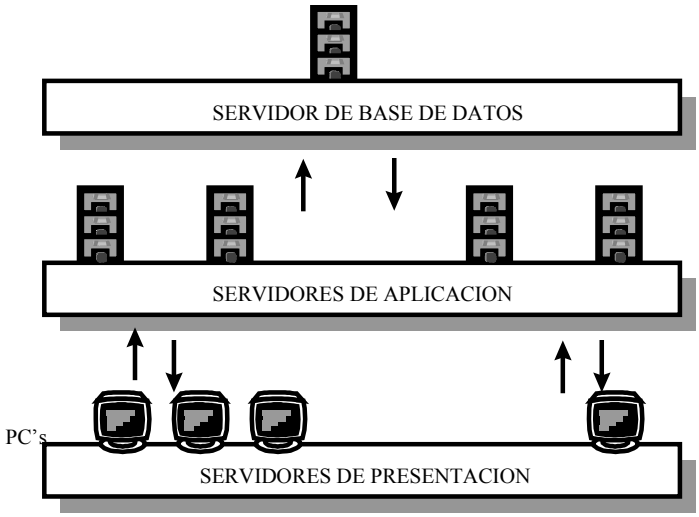
2.- Sistema básico, es el encargado de la interfaz entre el sistema operativo y las aplicaciones R/3 incluyendo componentes tales como el entorno de desarrollo ABAP, herramientas de administración del sistema, manejo de jobs, autorizaciones, etc.

3 .- Sistema Operativo, gestión de la base de datos y la red cuyo software viene incluido en SAP R/3.

2 Características de SAP R/3

SAP R/3 ofrece para gestionar los distintas funciones de una empresa las siguientes características:

- **Sistema Cliente - Servidor**



Arquitectura Cliente- Servidor

Sistema Cliente Servidor.- en la computación cliente-servidor, una parte del procesamiento se ejecuta en el PC de sobremesa (cliente) y la otra en computadoras centrales compartidas (servidores). La presentación y el preprocesamiento se ejecutan en el PC, la información se almacena en los servidores.

Servidor Base de Datos.- Este es el servidor central que contiene la base de datos (el sistema de gestión de base de datos) y se conoce generalmente como servidor de base de datos.

Servidor de Aplicaciones.- Contienen la lógica de proceso del sistema incluyendo servicios como el de impresión, peticiones de usuario, servicios para procesar los jobs de fondo, etc.

Servidores de Presentación.- Tareas relacionadas con la interface usuario y la presentación de datos (normalmente PC's).

La comunicación entre los tres niveles anteriores se realiza mediante el protocolo estándar TCP/IP.

- **Tecnología de Sistemas Abiertos**

  Significa que la aplicaciones pueden funcionar sobre múltiples sistemas operativos (UNIX, WINDOWS NT, AS400, etc.) y gestores de bases de datos (ORACLE, INFORMIX, ADABAS, etc.), siendo el código fuente de las aplicaciones ABAP completamente reutilizables y transportables entre los distintos sistemas.

  SAP soporta muchas GUI (interfaces gráficas de usuario) tales como Windows 3.11, Windows 95, Windows 98, Windows NT, Macintosh, etc. La GUI diseñada por SAP es la SAP GUI y esta orientada a ventanas, botones, iconos, barras de menú, barras de herramientas, etc.

- **Integración de Aplicaciones**

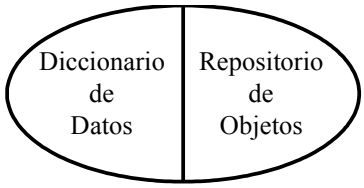
Todas las aplicaciones R/3 están integradas y relacionadas con capacidad de hacerlo en tiempo real, es decir, la información se actualiza constantemente. Que significa esto: cualquier cambio que se realice por ejemplo en una base de datos se reflejará inmediatamente en todos los componentes de SAP: screen painter, menu painter, diccionario, etc.

- **Entorno de desarrollo**

  Incluye todas las herramientas necesarias para el diseño y desarrollo de programas, pantallas, menús, módulos de funciones, etc. Contiene también funciones para realizar la depuración de programas y pruebas de rendimiento.

  Todas las aplicaciones estándar de R/3 están realizadas en ABAP.

  Se ha diseñado un entorno de desarrollo llamado Development Workbench que se encuentra integrado dentro del sistema R/3 y permite al cliente desarrollar soluciones especificas o ampliar las estándar en el núcleo de entorno de desarrollo se encuentran el Repositorio de objetos y el Diccionario de datos.



Núcleo de Entorno de Desarrollo

Dentro de el Diccionario de datos se encuentra lo siguiente: definición de tablas, valores permitidos, relaciones entre tablas, etc.

Repositorio de Objetos: programas, datos del diccionario, dynpros, documentación, etc.

- **Herramientas para la configuración del sistema**

  La parametrización es la piedra angular de una implantación SAP R/3. Algunas tareas de parametrización son tan sencillas como introducir el país donde esta situada la empresa y otras son tan complicadas orientadas a áreas o industrias especificas que requiere conocimientos técnicos de configuración como de actividades empresariales.

- **Servicios de Soporte, Formación, consultoría e implantación (OSS)**

  SAP ha dispuesto un amplio conjunto de servicio de calidad para ayudar a su cliente durante el proceso de implantación y soporte de los sistemas R/3, estos servicios abarcan desde información de aplicaciones, formación, servicio de instalación hasta consultoría.

SAP realiza la gran mayoría de los servicios a través de conexiones remotas con red de comunicación internacional. El sistema de servicio en línea de SAP se llama OSS (Online Service System).

- **Euro y Año 2000**

  Soporte completo para solucionar los problemas del cambio del milenio (Año 2000) y la aparición de la moneda Europea EURO.

3 Entorno de Desarrollo ABAP/4

El entorno de desarrollo ABAP/4 consiste de las siguientes herramientas :

Para	**La herramienta / componente**	**Se utiliza para**
Programación	Diccionario ABAP	Definir, mantener y almacenar el diccionario de datos del sistema R/3. Contiene todos los objetos

		del diccionario, tales como tablas, relaciones, documentación, etc.
	Editor ABAP	Crear y mantener los programas ABAP para editar módulos de función, bases de datos lógicas y la lógica de programación de las pantallas (Dynpros)
	Librería de Funciones	Definir y mantener modulo de función ABAP (rutinas de propósito general que pueden ser utilizadas en otros programas ABAP)
	Screen Painter	Diseñar y mantener las pantallas e interfaces gráficas de usuario en R/3.
	Menu Painter	Diseñar y mantener los menús para los interfaces gráficos de usuario.
Navegación	Object Browser	Gestionar y revisar los objetos de desarrollo de modo jerárquico para permitir una navegación fácil entre los objetos y el entorno de desarrollo.
	Sistema de información del Repositorio ABAP	Navegar y buscar objetos del diccionario, objetos de desarrollo y relaciones entre objetos de desarrollo.
	Jerarquía Aplicación	Visualizar los objetos de desarrollo desde un punto de vista organizativo y de aplicación.
	Data Browser	Navegar y visualizar los contenidos de las tablas de la base de datos.
Debugging	Trace SQL	Seguir y rastrear los accesos y llamadas a la base de datos desde los programas y transacciones del sistema
	Análisis Tiempo Ejecución	Analizar el rendimiento de las llamadas al sistema.
	Debugger en línea	detener un programa y analizar el resultado de la ejecución de cada sentencia del programa.
	Lock del Sistema	Seguimiento de los errores y mensajes que se producen durante la ejecución de los

		programas.
Organización del desarrollo	Workbench Organizer	Controlar y seguir el trabajo de desarrollo y los proyectos en equipo y para gestionar las versiones de los objetos de desarrollo.
	Sistema de Transporte	Realizar y gestionar los transportes de los objetos de desarrollo entre distintos sistemas SAP.

Concepto de Mandante (cliente).- Muchas veces se entiende de manera equivocada el este concepto, en realidad es el nombre del sistema SAP R/3 al que nos conectamos, la mayoría de las compañías cuentan con un cliente para cada tarea específica.

Se define como una unidad independiente dentro del sistema R/3, desde el punto de vista fiscal, legal y organizativo. Por ejemplo un mandante se puede decir que representa a una empresa dentro de una corporación. Técnicamente se puede decir que un mandante se comporta dentro de SAP como una base de datos lógica independiente, es decir, los datos de una tabla en un mandante no pueden ser modificados ni visualizados desde otro mandante.

Por ejemplo una compañía podría tener la siguiente configuración:

El sistema cliente de formación 400, que se utiliza para la formación de nuevos usuarios.

El sistema cliente de desarrollo 100, que se utiliza para nuevos desarrollos asi como etapa de prueba de los nuevos desarrollos.

El sistema de producción 100 es le sistema activo utilizado para dirigir la empresa. Ésta es un área nada recomendable para practicar.

Es importante conocer el cliente o mandante en que se requiera realizar la tarea específica esto debido a los posibles problemas que se pueden generar. Cada cliente o mandante tiene especificadas autorizaciones para realizar tal o cual tarea. La realización de las tareas en cada mandante son restringidas y asignadas según el nivel del usuario.

Por ejemplo un usuario final no siempre tiene autorización para manipular todos los menús de SAP R/3, asi como un programador puede tener un límite dentro del sistema de desarrollo (no estar autorizado para actualizar bases de datos, eliminar elementos de SAP, etc.), un funcional no tendra acceso a los recursos del sistema base (administración de la base de datos)…

Concepto de Transacción.- De un modo genérico una transacción es una operación que permite a un usuario realizar cambios en la base de datos. Todo el sistema R/3 se puede considerar como un sistema de proceso de transacciones de negocios.

4 Aplicaciones del ABAP/4

¿ Qué es ABAP/4 ?

ABAP/4 es el lenguaje de programación de cuarta generación propio de SAP, su iniciales nos indican:

A - Advanced
B - Business
A - Aplication
P - Programming

Las aplicaciones del ABAP/4 son:

- **Reporting** (Clásico e interactivo)
- **Programación de diálogo o transacciones** (diseño de pantallas)
- **Otras aplicaciones** (Batch Input, 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 se requiere un formato muy completo.

El **Reporting Clásico** se caracteriza por listados voluminosos o muy frecuentes con mezcla de información detallada y resumidas.

El **Reporting Interactivo** esta orientado a pantallas, listados cortos y ventanas controladas por teclas de función.

Ambos **reporting** se pueden ejecutar en **Online** (tiempo real) mientras que únicamente el clásico se puede ejecutar en **Batch** (diferido).

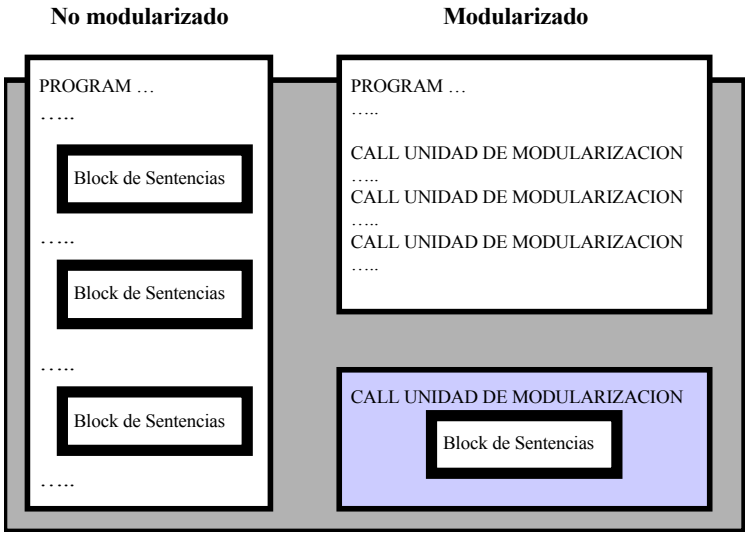
La **programación en diálogo** se caracteriza por estar enfocado a pantallas que estarán controladas por módulos ABAP/4.

Otras aplicaciones posibles en lenguaje de programación son la generación de Batch Input 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 en batch la introducción de datos en el sistema vía transacción online.

Modularización

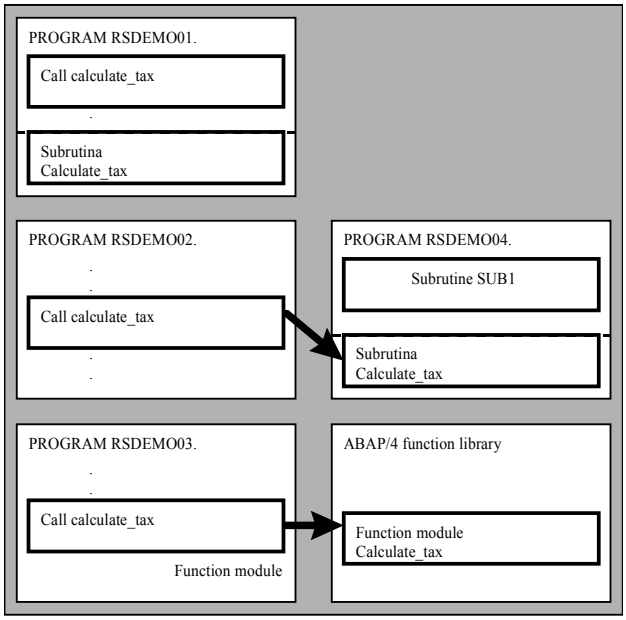
Por modularización dentro de los programas ABAP/4, entendemos hacer más fácil la lectura y mejorar su estructura. Modularizar programas facilita el mantenimiento y la actualización de los mismos a comparación de los que no están debidamente modularizados. (estructurados)

Principio de Modularización



Técnicas para facilitar la modularización:

Modularización



Definir subrutinas internas y externas para evitar secuencias de sentencias similares o idénticas, o sea evitar redundancia. Las subrutinas mejoran la estructura de tu programa (esto significa modularización) y hace más fácil la lectura. Una secuencia de sentencias definida dentro de una subrutina puede ser llamada desde varios puntos de un programa.

Para mayor claridad debes colocar las subrutinas al inicio o al fin del programa.

- Puedes definir programas include dentro de la biblioteca.

  Sí tu quieres usar la misma secuencia de sentencias en varios programas, tu puedes realizar el código dentro de un programa include.

- Definir y realizar llamados a módulos de función. Los cuales son almacenados dentro de la biblioteca de funciones donde son asignados a un grupo de función.

  El sistema R/3 proporciona diversos módulos de función predefinidos, los cuales pueden ser llamados desde cualquier programa ABAP/4, además estos módulos de función pueden ser creados por el propio programador.

  A diferencia de las subrutinas los módulos de función cuenta con una interface; además de estandarizar el pase de parámetros.

- Interfaces abiertas (llamadas a programas externos)

Introducción a ABAP/4

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, esto es una vez escrita la sentencia puede ir un comentario a continuación.

Podemos combinar sentencias consecutivas del mismo formato. Esto significa agrupar sentencias y evitar el escribir más líneas de código.

Por ejemplo**:**

```
WRITE LFA1-LIFNR.
WRITE LFA1-NAME1.
WRITE LFA1-ORTO01.
```

es equivalente a :

```
WRITE:  LFA1-LIFNR,
        LFA1-NAME1,
        LFA1-ORTO01.
```

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

  Ejemplo:

```
DATA: VENTAS-TOTALES TYPE P.
```

5.3 Estructura de un programa.

REPORT <nombre> ————	Nombre programa
TABLES: ————	Tablas que se utilizan
DATA: ————	Variables internas
TOP-OF-PAGE. ————	Por inicio de página ejecutar
<Sentencias>	las instrucciones que se indiquen.
END-OF-PAGE. ————	Por fin de página ejecutar las
<Sentencias>	instrucciones que se indiquen.
START-OF-SELECTION. ———	Por inicio de programa
<Sentencias>	ejecutar las instrucciones
	indicadas.
END-OF-SELECTION. ————	Por Fin de programa
<Sentencias>	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	1E-307→1E+308	0.0E+00	Punto flotante
I	4	$-2^{31} \rightarrow 2^{31}-1$	0	Entero
N	1	1→32000	'0000'	Texto numérico
P	8	1→ 16	0	Núm. 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 indicar la variable con un valor distinto al que tienen 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-L.IFNR,
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 10,
                NOMBRE LIKE  LFA1-NAME 1,
                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 SOCIEDADE  OCURRS  1 0.
        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:  VARI(15) VALUE  'RIVERLAND BCN.',
         VAR2(15) VALUE  'HOLA'.
       MOVE VARI+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 desean 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 variables 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 del reg1 al 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 convesión 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 +, -, / y *,

COMPUTE var1 = <Exp. Aritmética>.

donde la palabra COMPUTE es opcional.

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

- Con las instrucciones : **ADD TO, SUBSTRACT FROM, MULTIPLY BY y DIVIDE BY.**

También dispondremos de funciones matemáticas para los números de coma flotante: **EXP, LOG, SIN, COS**, **SQRT, DIV, MOD. STRLEN.**

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 blanco por el lado opuesto.

Ejemplo:

H	O	L	A	

SHITF campo BY 2 PLACES.

L	A			

SHITF <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> WITII <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 'cadena 1'.

- 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 la 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 líneas 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 utilizados 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-T'ABIX que contiene la línea de proceso de una tabla interna.

En **el Anexo 1** 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.>.          IF <Cond.>.          IF <Cond.>.
   …                    …                    …
ENDIF.               ELSE.                ELSEIF.
      …                                         …
   ENDIF.                              ELSEIF.
                                                …
                                             ELSE.
                                                …

                                             ENDIF.
```

En las condiciones utilizamos los clásicos operadores:

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

Además existen operadores adicionales para comparar cadenas de caracteres:

<f1> **CO** <f2> (Contains Only) : f1 sólo contiene caracteres de f2. En caso de ser cierta **SY-FDPOS** contiene la longitud de f1, y si es falsa contiene el offset del primer 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 primer 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 **SY-FDPOS** es cierta 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. Cómo 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 ST'OP 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** se tratará 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 Cómo 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 <intal> (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**:**

AT FIRST.
…
ENDAT.

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

AT LAST.
…
ENDAT.

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

AT NEW <campo>.
…
ENDAT.

Realiza las instrucciones que hay a continuación del AT NEW para cada inicio de nivel de ruptura.

AT END OF <campo>.
…
ENDAT.

Realiza las instrucciones que hay a continuación del AT END para cada final de nivel de ruptura.

Si utilizamos la instrucción **SUM** dentro de un AT ... ENDAT, realizará la suma de todos los campos P,I,F de ese nivel de ruptura (para el cálculo de subtotales).

El resultado lo encontraremos en el área de trabajo de la tabla.

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

Así la utilización conjunta de todas estas instrucciones será:

```
SORT <intab> BY <c1> <c2>.
LOOP AT <intab>.
        AT FIRST ... (SUM) ... ENDAT.
        AT NEW <cl>.
        ... (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> WHIT KEY <clave>.
```

No necesita llenar la línea de cabecera. Buscará desde el inicio de la tabla qué 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 sobreescribir 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) el 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 Subrutinas

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> Llamada a un procedimiento o subprograma.

FORM <modulo> Subprograma.

ENDFORM

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

```
…
   PERFORM <modulo> USING varl var2 ...
…
   FORM <modulo> USING varl 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 <varl> <var2> ...

FORM <modulo> TABLES <intab> ...
USING <varl> ...

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 <módulo> USING <reg>.

FORM <modulo> USING <reg> STRUCTURE <estructura>.

ENDFORM.

Es decir con la cláusula STRUC'I'URE 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> S'I'RUCTURE <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 EXPORTNG especificamos los parámetros de entrada, en el IMPOR'I'ING (opcional) el resultado o retorno de la función y en TABI.ES (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 SHO\\' 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.**

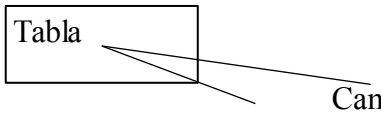
Para recuperar de memoria:

IMPORT <campo>... **FROM 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: SKBI,SKMI ...

CAMPO: STEXT

ELEM.DATOS: STEXI' - SKBI
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, carnpos, dominios, programas, …etc.

Existen diversos tipos de tablas-.En el código del include no utilizaremos la sentencia REPORT...

-

Tablas **TRANSP** (transparentes): Tablas nominales **relacionales** (SQL).

- 'I'ablas **POOL**: 'I'ablas 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.

Éstas son:

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

Además de las variables del sistema:

SY -SUBRC: Código de retorno de una operación.
SN-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.

- 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 TABI,E <intab> (WHERE <cond»)

Llena la tabla interna <intab> machacando los registros que pudiera tener ésta. 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 <campol> <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 finalizara sino que continuará con el siguiente registro.

Si **SY-SUBRC = 0** Todos los registros modificados.
Si **SY-SUBRC = 4** No todos los registros han sido modificados.
En **SY-DBCNT** Tendremos la cantidad de registros modifacados.

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 modifcarlo 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 **SY-SUBRC** = 0

Todos los registros han sido borrados.
Si **SY-SUBRC** = 4

No todos los registros han sido borrados.
En **SY-DBCNT**

Tendremos la cantidad de registros borrados.

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 Locking 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 a 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.**

AUTORITY-CHECK OBJECT <objeto_de_autorizaci>
ID <Campo1> FIELD-<f1>
ID <Catupo2> FIELD <f2>
ID <Campo3> DIJMMY.

Donde <Campo(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.
En **SY-DBCNT**

Tendremos la cantidad de registros borrados.

Ejemplo:

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

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

Para obtener una documentación más exhaustiva sobre el funcionamiento del AUTORITY-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 F1

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 <tabla BDD1>.
<sentencias evento>
```

……

GET <tabla BDD2>.
<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.

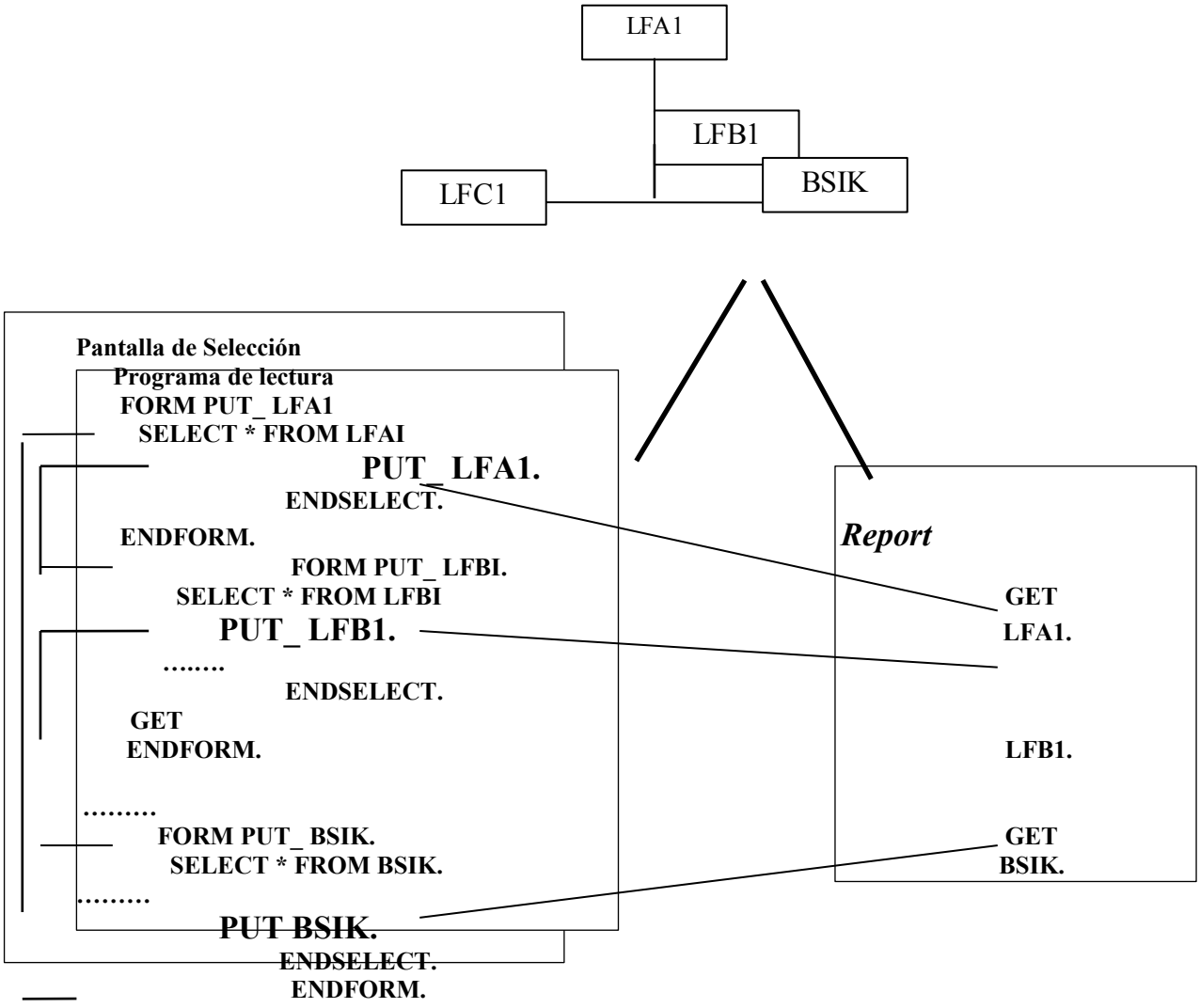
Para comunicar el programa de lectura con nuestro report se utiliza el **PUT**, que suministra el registro de la BDD que especifiquemos, previamente habrá realizado el SELECT.

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).

KDF



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. Fields 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 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 querernos 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.
      EXTRACTIMPORTES.

*---------------------------------------------------------------------
```

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

EXTRACT DIRECCION

EXTRACT DIRECCION

	PROVEEDOR1	RIVERLAND DIAGONAL 618 BARCELONA

EXTRACT IMPORTE

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-BELNNR, 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>(<Iong>) "<datos a visualizar>".

ULINE /<offset>(<Iong>) "<datos a visualizar>".

SKIP <n>.

NEW-PAGE.

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

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

WRITE <campo2> UNDER <campo 1>.

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

POSITION <columna>.

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

SKIP TO LINE <n>.

- Cuando utilizamos la instrucción WRITE con números empaquetados, el sistema trunca por la izquierda en caso de ser necesario (deja un * como indicador de que ha truncado) y rellena con blancos si sobra espacio. Tenemos que tener 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:**

WRITE <campo_importe> CURRENCY<moneda>.

- Si se desea formatear la salida de un campo según una cierta máscara utilizaremos el parámetro **USING EDIT MASK "<mascara>"** 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 mascara). mascara.
"RR" : justifica por la derecha. (Al principio de la mascara).

Ejemplo:
WRITE /(8) SY-UZEIT IJSING EDIT MASK "_:_:_".

- Si queremos suprimir los ceros iniciales de una cadena de caracteres haremos:

WRITE <campo_Caracter> NO-ZERO.

- Para formatear fechas es posible realizar:

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

- Podemos modificar los atributos de pantalla para un campo.

FORMAT INTENSIFIED ON/OFF.
FORMAT INVERSE OFF/ON.
FORMAT INPUT OFF/ON.
FORMAT COLOR n.
FORMAT RESET.

Ver la **documentación Online** del editor ABAP/4 para obtener información mas 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 :

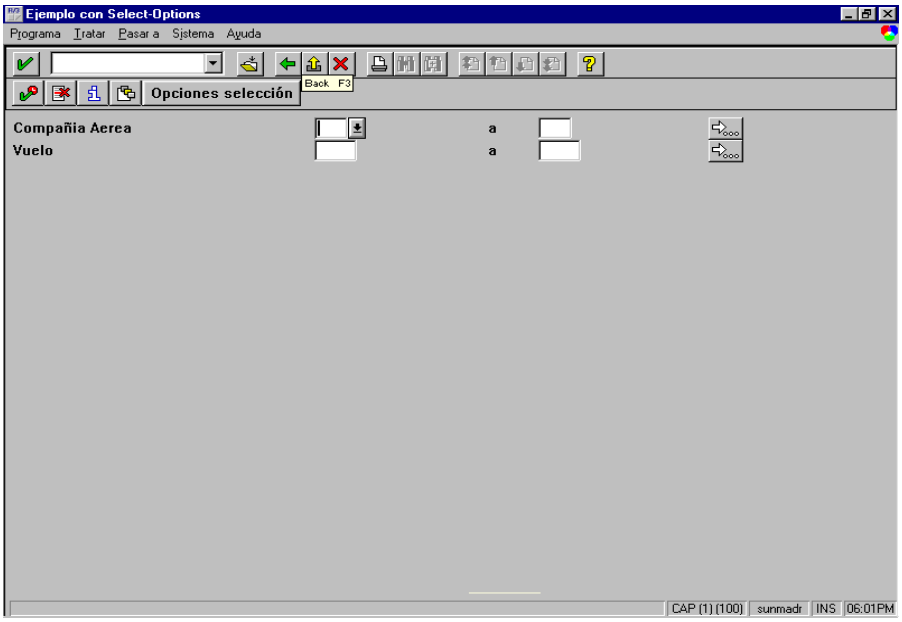
SELECT-OPTIONS <var> FOR <campo_tabla>.

<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**>**-SIGN**, <**var**>**-OPTION**.
El contenido de cada registro será respectivamente: el valor inferior, el superior, el signo (**I**ncluido/**E**xcluido) y el operador.
En la pantalla de selección si queremos realizar una selección compuesta de más de una condición (más de un registro en la tabla interna), tendremos que hacer un 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 fecha).

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 14.4).
Si además queremos que al pulsar Fl (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 parametros 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-xxx,** donde **xxx** son tres caracteres cualquiera. 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 <área>.

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

Para visualizar un mensaje utilizamos la sentencia MESSAGE.

MESSAGE Tnnn.

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

A = Cancelación o "Abend" del proceso.

E = Error. Es necesaria una corrección de los datos.

I = Información. Mensaje meramente informativo.
El proceso continuará con un ENTER.

S = Confirmación. Información en la pantalla siguiente.

W = Warning. Nos da un aviso.
Podemos cambiar los datos o pulsar "intro" para continuar.

Si se emiten mensajes del tipo W o E en eventos START-OF-SELECTION o 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 cómo 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>.  ————————  "ZPRUEBA"
MOVE "ZPRUEBA" TO <F>.
WRITE TRDIR-NAME.
```

2.-

```
FIELD-SYMBOLS <F>.
TEXTO="ABCDEFGH".
INICIO = 2.                ————————  "CDEFG"
LONGITUD =5.
ASSIGN TEXTO+INICIO (LONGITUD) TO <F>
WRITE <F>
```

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>
        WIHILE <PUNTERO> EQ SPACE.
                        SHIFT NUMERO RIGHT.
                        WRITE “O”"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>.
        “ZPRUEBA”
MOVE”TRDIR-NAME" TO CAMPO .

ASSIGN (CAMPO) TO <F>.
WRITE <F>.
```

Para información adicional sobre los Field Syrnbols 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 dalos 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 cómo 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 en 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:

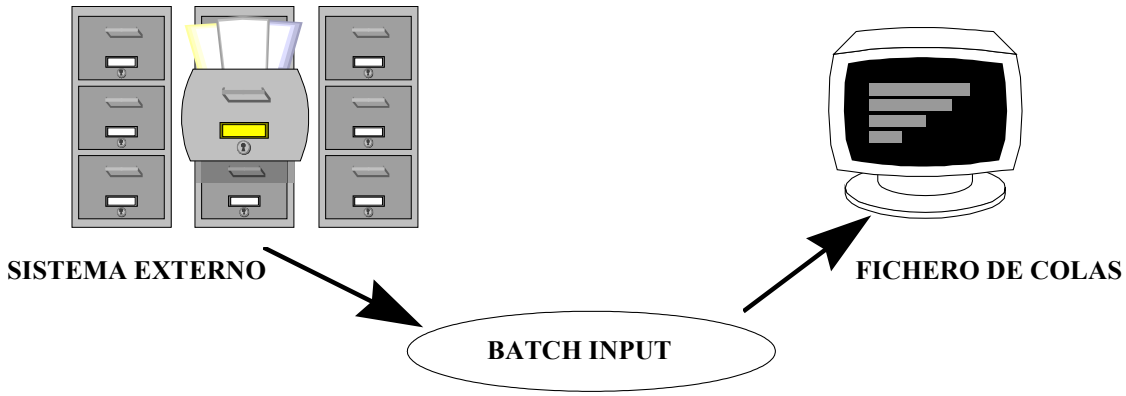
Fase de Generación: A partir de una fuente de información como puede ser un fichero de entrada, donde estarán todos los datos que queremos cargar en SAP, se transformaran estos datos en un formato determinado, para almacenarlo en una estructura de SAP que llamaremos fichero de colas.

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 ésta.

Cómo 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 fondo 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 estarnos 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 Fl 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 en 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 querernos 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 estido 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 TOO1 (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 cómo 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> 0CCURS <n>.
                INCLUDE STRUCTURE BDCDATA.
      DATA: 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 **Fl** (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 tab_B_I, 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: FSSI
Programa: SAPMF02H
Dynpro : 0102

```
tab_B_  I-PROGRAM =”SAPMF02H”.
tab_B_  I- DYNPRO =”0102”.
tab_B_  I- DYNBEGIN =”X”.
APPEND tab_B_I.
```

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 tab_B_I.
tab_B_I -FNAM =”RF02H-SAKNR”.
tab_B_I -FVAL = VAR_CTA.
APPEND tab_B_I.
```

```
CLEAR tab_B_I.
tab_B_I -FNAM =”RF02H-BUKRS”.
tab_B_I -FVAL = VAR_SOC.
APPEND tab_B_I.
```

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 tab_B_I.
tab_B_I -FNAM =”campo(índice)”.
tab_B_I -FVAL =”valor”.
APPEND tab_B_I.
```

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 tab_B_I.
tab_B_I -FNAM = BDC_OKCODE.
tab_B_I-FVAL = “/13”.                    “F13= Grabar.
APPEND tab_B_I.
```

También utilizamos el campo BDC_OKCODE para ejecutar funciones que aparecen en la barra de menús. Para saber el código de la función, Pulsar Fl 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 tab_B_I.
tab_B_I -FNAM = BDC_CURSOR.
tab_B_I-FVAL = “RF02H-BUKRS”
APPEND tab_B_I.
```

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

1.- Abrir la sesión de Batch Input utilizando el modulo de función **BDC_OPEN_GROUP.**

2.- Para cada transacción de la sesión:

2.a.- Llenaremos la tabla **tab_B_I** 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

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

                    CLIENT_INVALID          = 01
                    DESTINATION_INVALID     = 02
                    GROUP_INVALID           = 03
                    HOLDDATE_INVALID        = 04
                    INTERNAL_ERROR          = 05
                    QUEUE_ERROR             = 06
                    RUNNING                 = 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 Batch 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 las 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 <transacción>
                 USING       <tabint>
                 MODE        <modo_ejec>
                 UPDATE      <tipo_actual>.
```

Donde:

<tabint> Tabla interna (con estructura BDCDATA)

<modo_ejec> Modo 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-MSGID
    TYPE SY-MSGTY NUMBER SY-MSGNO
    WITH SY-MSGVI 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.

  Cómo 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 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.

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

OPEN DATASET <fichero>.
FOR OUTPUT / (INPUT) ——————— Escritura / Lectura (por defecto).
IN BINARY MODE / IN TEXT MODE —— Binario (por defecto) / Texto.

Si SY-SUBRC = 0 Fichero abierto correctamente.

SY-SUBRC = 8 Fichero no se ha podido abrir.

- Para cerrar un fichero utilizamos **CLOSE.**

  CLOSE DATASET <fichero>.

- Si queremos leer de un fichero utilizamos **READ.**

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

Ejemplo:

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

OPEN DATASET "/usr/test".

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

Notas: - Se puede leer de hasta 4 ficheros simultaneamente.
- Si SY-SUBRC = 0 Fichero leído correctamente.
SY-SUBRC = 4 Fin de fichero encontrado.

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

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

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

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

Nota:

Este tratamiento es válido solo sobre ficheros UNIX, es necesario comparar la declaración de ficheros UNIX como de DOS y otros.

Ficheros UNIX se identifican por:

file(30) default '/tmp/TELXX'., su tratamiento es con las sentencias anteriores: open dataset, read dataset, transfer.

A diferencia de los no exclusivos de UNIX.

File(30) default 'c:\prueba\prueba02.txt', su tratamiento se realiza mediante llamadas a funciones (módulos de función), como por ejemplo: UPLOAD y DOWNLOAD

A n e x o 1

ABAP/4 EDITOR

Se pueden especificar los comandos del editor en cualquiera de la formas siguientes:

- Como comandos de cabecera (en la línea de comandos sobre las líneas
- Comandos de línea sobreescribiendo los números de líneas
- Pulsando las teclas de función o seleccionando las opciones de menú.

Comandos de Cabecera

A(TTACH) n Visualiza el texto desde la línea n

B(OT'I'OM) Ir al final.

T(OP) Ir al principio.

\+ Siguiente página.

- Página anterior.

FIND c Buscar la cadena c desde la posición del cursor, el cursor se posiciona en la línea relevante.

Si la cadena contiene blancos o caracteres especiales, se deberán acotar entere caracteres especiales no contenidos en la cadena a localizar.

Ejemplo: FIND /vacío- /

el comando no distingue entre mayúsculas y minúsculas.

N(EXT) Busca y se desplaza a la siguiente ocurrencia de la cadena solicitada,desde la posición actual del cursor.

R(EPLACE) cl c2 Reemplaza la cadena c1 por la cadena c2 en todo el texto. c1 y c2 pueden tener distintas longitudes. Si una de las cadenas contiene blancos o caracteres especiales se deberán acotar, ambas entre caracteres especiales, ver lo indicado en FIND.

Ejemplo; R/empty- /blanks/

F(ETCH) prog Realiza la edición del programa indicando abandonando el programa actual.

S(AVE) Guarda el contenido del editor en un almacenamiento intermedio.

El comando UPDATE borra cualquier texto del almacenamiento intermedio.

Si un se produce una caída de sistema, normalmente el texto es recuperado del almacenamiento intermedio.

RES(TORE) Restaura el texto desde el almacenamiento intermedio, sobreescribiendo el existente.

RES(TORE) AKTIV Restaura la versión activa

en el DLIB.

SAVEAS prog Salva el programa con otro

nombre.

U(PDATE) Salva el contenido del editor

CHECK Cheque la sintaxis del programa

PCF(ETCH) Carga un fichero contenido en el PC.

PC(DOWN) Escribe el contenido del editor en un fichero de PC

HELP word Visualiza la ayuda sobre la palabra indicada.

I(NSERT) n Inserta n líneas al final del texto.

IC word Inserta la estructura de la sentencia indicada, esto es valido para las sentencias: CASE, DO, FORM, IF, LOOP, MESSAGE, MODULE, SEILECT, SHIFT, SORT, TRANSFER, WHILE y WINDOW.

IC FUNCTION func Inserta la estructura de un CALL FUNC'I'ION para la función indicada.

IC SELECT tab Inserta la estructura del SELECT para la tabla indicada.

IC... Inserta en la posición del cursor...

*f - FORM bloque de comentario
*m - MODULE bloque de comentario
. - Línea de comentario *.....text.....*
- - Línea de comentario *----------*
*-*1 - Área de comentario con línea en blanco.
** - Línea de comentario **********
**n - Área de comentario con n líneas en blanco (1<=n<=5)

PP Pretty Print del programa.

PRINT Imprime el contenido del editor.

RENUM(BER) Renumera líneas.

SHOW tab Visualiza los campos de la tabla indicada.

SHOW FUNCTION func Visualiza el módulo de la función indicada.

Comandos de Línea:

***** Considera la línea como primera línea en la pantalla.

T+ Ir a la primera línea.
B- Ir a la última línea.

> Inserta las líneas de programa del include..

< Elimina las líneas de código del include y restaura éste.

u Escribe el bloque incluido en el fichero de INCLUDE e inserta la sentencia include correpondiente.

A Línea de destino de una operación de copia o movimiento, el texto seleccionado se incluirá en la línea posterior.

B Línea de destino de una operación de copia o movimiento, el texto seleccionado se incluirá en la línea anterior.

O Ovrelay el contenido de C o M sobre la línea indicada.

C Copia esta línea.

CC **...CC** Copia el bloque de líneas.

M Mueve la línea indicada.

MM ... MM Mueva las líneas indicadas.

I Inserta una nueva línea.

In Inserta n líneas.

N Inserta un área de comentario.

D Borra la línea.

DD...DD Borra el bloque de líneas.

R Repite la línea.

Rn Repite la línea n veces.

RR...RR Repite el bloque de líneas.

J Junta la línea actual y siguiente.

S Parte la línea a la posición del cursor.

SH...SH Desplaza el bloque de líneas a la posición del cursor.

WW...WW Marca el bloque de líneas en el archivo intermedio general.

W Copia el contenido del archivo intermedio general.

XX...XX Copia el bloque indicado en el archivo intermedio X.

YY...YY Ver XX.

Y Ver Y

ZZ...ZZ Ver XX.

Z Ver X.

CLEAR Borra los buffers X, Y, Z...

PR...PR Imprime el bloque de líneas.

Variables del Sistema

Nombre de Variable	Descripción
SY-INDEX	Cantidad de repeticiones de bucles
SY-PAGNO	RUNTIME: Página actual en creación de lista
SY-TABIX	RUNTIME: Línea actual de una tabla interna
SY-TFILL	Cantidad actual de entradas en la tabla interna
SY-TLOPC	Utilización interna
SY-TMAXL	Cantidad máxima de entradas en la tabla interna
SY-TOCCU	Parámetro occurs en tablas internas
SY-TTABC	Número de la última línea de tabla interna leída
SY-TSTIS	Utilización interna
SY-TTABI	Offset de tablas internas en el área de roll

SY-DBCNT	Cantidad elementos en conjunto tratado para operaciones BD
SY-FDPOS	Lugar de hallazgo de un string
SY-COLNO	Columna actual en la creación de la lista
SY-LINCT	Cantidad de líneas de lista
SY-LINNO	Línea actual en la creación de una lista
SY-LINSZ	Longitud de línea de la lista
SY-PAGCT	Límite de página de lista en instrucción REPORT
SY-MACOL	Cantidad de columnas de instrucción SET MARGIN
SY-MAROW	Cantidad de líneas de instrucción SET MARGIN
SY-TLENG	Tamaño de la línea de una tabla interna
SY-SFOFF	Utilización interna
SY-WILLI	Número de la línea de ventana actual
SY-LILLI	Número de la línea de lista actual
SY-SUBRC	Valor de retorno tras determinadas sentencias ABAP/4
SY-FLENG	Utilización interna (longitud de campo)
SY-CUCOL	Posición del cursor (columna)
SY-CUROW	Posición del cursor (línea)
SY-LSIND	Número de la lista de bifurcación
SY-LISTI	Número de la línea de lista actual
SY-STEPL	Número de la línea LOOP en step dynpro
SY-TPAGI	Indicador para almacenar tabla interna en bloque paging
SY-WINX1	Coordenada de ventana (columna izquierda)
SY-WINY1	Coordenada ventana (línea izquierda)
SY-WINX2	Coordenada ventana (columna derecha)
SY-WINY2	Coordenada de ventana (línea derecha)
SY-WINCO	Posición de cursor en la ventana (columna)

Nombre de Variable	**Descripción**
SY-WINRO	Posición de cursor en la ventana (línea)
SY-WINDI	Indice de la línea de ventana actual
SY-SROWS	Líneas en la pantalla
SY-SCOLS	Columnas en la pantalla
SY-LOOPC	Cantidad de líneas LOOP en steploop de dynpro
SY-FOLEN	Utilización interna (longitud de salida de campo)
SY-FODEC	Utilización interna (campo posiciones decimales)
SY-TZONE	Diferencia de tiempo con 'Hora media de Greenwich' (UTC)
SY-DAYST	¿ Horario de verano activo ?
SY-FTYPE	Utilización interna (tipo de campo)
SY-APPLI	Aplicaciones SAP
SY-FDAYW	Día de semana en el calendario de fábrica
SY-CCURS	Tipo cambio/Campo resultado CURRENCY CONVERT

SY-CCURT	Tipo de cambio en tabla de aplicación CURRENCY CONVERSION
SY-DEBUG	Utilización interna
SY-CTYPE	Tipo de cambio 'M','B','G' de CURRENCY CONVERSION
SY-INPUT	Utilización interna
SY-LANGU	Clave de idioma para entrar al Sistema SAP
SY-MODNO	Cantidad de modos alternativos
SY-BATCH	Batch activo (X)
SY-BINPT	Batch input activo (X)
SY-CALLD	Call modo activo (X)
SY-DYNNR	Número de la imagen en pantalla actual
SY-DYNGR	Grupo de dynpros del dynpro actual
SY-NEWPA	Utilización interna
SY-PRI40	Utilización interna
SY-RSTRT	Utilización interna
SY-WTITL	Indicador para cabecera estándar de página
SY-CPAGE	Número de página actual
SY-DBNAM	Base de datos lógica en report ABAP/4
SY-MANDT	Número de mandante para acceder al Sistema SAP
SY-PREFX	Prefijo ABAP/4 para jobs batch
SY-FMKEY	Menú de códigos de funciones actual
SY-PEXPI	IMPRIMIR: Tiempo de permanencia en SPOOL
SY-PRINI	Utilización interna
SY-PRIMM	IMPRESION: Salida inmediata

Nombre de Variable	Descripción
SY-PRREL	IMPRESION: Borrar tras salida
SY-PLAYO	Utilización interna
SY-PRBIG	IMPRIMIR: Portada de selección
SY-PLAYP	Utilización interna
SY-PRNEW	IMPRESION: Nueva orden SPOOL (lista)
SY-PRLOG	Utilización interna
SY-PDEST	IMPRIMIR: Dispositivo de salida
SY-PLIST	IMPRESION: Nombre de la orden SPOOL (nombre de lista)
SY-PAUTH	Utilización interna
SY-PRDSN	IMPRIMIR: Nombre del set de datos SPOOL
SY-PNWPA	Utilización interna
SY-CALLR	IMPRIMIR: ID para funciones de diálogo
SY-REPI2	Utilización interna

SY-RTITL	IMPRIMIR: Título de report del programa de impresión
SY-PRREC	IMPRIMIR: Destinatario
SY-PRTXT	IMPRIMIR: Texto para portada
SY-PRABT	IMPRIMIR: Departamento en la portada
SY-LPASS	Utilización interna
SY-NRPAG	Utilización interna
SY-PAART	IMPRESION: Edición
SY-PRCOP	IMPRIMIR: Cantidad de ejemplares
SY-BATZS	SUBMIT batch: Inmediatamente
SY-BSPLD	SUBMIT fondo: Salida de lista en SPOOL
SY-BREP4	SUBMIT fondo: Nombre de raíz del report de llamada
SY-BATZO	SUBMIT fondo: Unico
SY-BATZD	SUBMIT fondo: Diario
SY-BATZW	SUBMIT fondo: Semanal
SY-BATZM	SUBMIT fondo: Mensual
SY-CTABL	Tabla de tipo de cambio en CURRENCY CONVERSION
SY-DBSYS	SYSTEM: Sistema de base de datos
SY-DCSYS	SYSTEM: Sistema de diálogo
SY-MACDB	PROGRAM: Nombre del fichero para el acceso con matchcode
SY-SYSID	SYSTEM: Identificador del Sistema SAP
SY-OPSYS	SYSTEM: Sistema operativo
SY-PFKEY	RUNTIME: Status de teclas-F actual
SY-SAPRL	SISTEMA: Release SAP

Nombre de Variable	Descripción
SY-TCODE	SESSION: Código de transacción actual
SY-UCOMM	INTERACT.: Indicar función en el código OK
SY-CFWAE	Utilización interna
SY-CHWAE	Utilización interna
SY-SPONO	RUNTIME: Número SPOOL para salida de una lista
SY-SPONR	RUNTIME: Número SPOOL de instrucción TRANSFER
SY-WAERS	T001: Moneda de sociedad tras leer segmento B
SY-CDATE	Fecha de tipo de cambio de CURRENCY CONVS.
SY-DATUM	SYSTEM: Fecha del día
SY-SLSET	Nombre de SELECTON-SETS
SY-SUBTY	ABAP: Forma de llamada en SUBMIT
SY-SUBCS	INTERNO: Status call del report
SY-GROUP	INTERNO: Concatenación
SY-FFILE	INTERNO: Flatfile (USING/GENERATING DATASET)
SY-UZEIT	SYSTEM: Hora
SY-DSNAM	RUNTIME: Nombre del set de datos para salida en

	SPOOL
SY-REPID	PROGRAM: Nombre de un programa ABAP/4
SY-TABID	Utilización interna
SY-TFDSN	RUNTIME: Nombre del set de datos para extractos de datos
SY-UNAME	SESSION: Nombre de usuario según entrada a SAP
SY-LSTAT	INTERACT.: Información de status por nivel de lista
SY-ABCDE	CONSTANT: Alfabeto (A,B,C,...)
SY-MARKY	Letra de línea actual para MARK
SY-SFNAM	Sin utilizar
SY-TNAME	Nombre de la tabla interna después de un acceso
SY-MSGLI	INTERACT.: Línea de mensaje (línea 23)
SYTITLE	PROGRAM: Título del programa ABAP/4
SY-ENTRY	Utilización interna
SY-LISEL	INTERACT.: Línea seleccionada
SY-ULINE	CONSTANT: Línea de subrayado (_____...)
SY-XCODE	Código OK ampliado
SY-CPROG	RUNTIME: Programa principal
SY-XPROG	Utilización interna (programa SYSTEM-EXIT)
SY-XFORM	Utilización interna (form SYSTEM-EXIT)
SY-LDBPG	PROGRAM: Programa ABAP/4 de base de datos para SY-DBNAM
SY-TVAR0	RUNTIME: Var. de texto para elementos de texto ABAP/4

Nombre de Variable	Descripción
SY-TVAR1	RUNTIME: Variable de texto para elementos de texto ABAP/4
SY-TVAR2	RUNTIME: Variable de texto para elementos de texto ABAP/4
SY-TVAR3	RUNTIME: Variable de texto para elementos de texto ABAP/4
SY-TVAR4	RUNTIME: Variable de texto para elementos de texto ABAP/4
SY-TVAR5	RUNTIME: Variable de texto para elementos de texto ABAP/4
SY-TVAR6	RUNTIME: Variable de texto para elementos de texto ABAP/4
SY-TVAR7	RUNTIME: Variable de texto para elementos de texto ABAP/4
SY-TVAR8	RUNTIME: Variable de texto para elementos de texto ABAP/4
SY-TVAR9	RUNTIME: Variable de texto para elementos de texto ABAP/4
SY-MSGID	ID de mensaje

SY-MSGTY	Tipo de mensaje (E,I,W,etc.)
SY-MSGNO	Número del mensaje
SY-MSGV1	Variable en mensaje
SY-MSGV2	Variable en mensaje
SY-MSGV3	Variable en mensaje
SY-MSGV4	Variable en mensaje
SY-ONCOM	INTERNO: On Commit Flag
SY-VLINE	CONSTANT: raya vertical
SY-WINSL	INTERACT.: Línea en ventana seleccionada
SY-STACO	INTERACT.: Lista visualizada a partir de la columna
SY-STARO	INTERACT.: Lista visualizada a partir de línea
SY-DATAR	Indicador: Datos recibidos
SY-HOST	Nombre de la máquina
SY-LOCDB	Existe base de datos local
SY-LOCOP	Operación local en base de datos
SY-DATLO	Fecha local, en relación con el usuario
SY-TIMLO	Hora local, en relación con el usuario
SY-TSTLO	Cronomarcador (fecha y hora), en relación con el usuario
SY-ZONLO	Huso horario del usuario
SY-DATUT	Fecha global, en relación con UTC
SY-TIMUT	Hora global, en relación con UTC
SY-TSTUT	Cronomarcador (fecha y hora), en relación con UTC

ABAP/4

Conceptos Avanzados

1 Reporting Interactivo

1. 1 Introducción al reporting interactivo.

Los informes (Reports) SAP se pueden clasificar claramente en: **Reporting Clásico y Reporting Interactivo.**

El **Reporting Clásico** como su nombre indica es el informe típico, que trabaja con gran cantidad de información y un nivel de detalle muy elevado.

En el **Reporting Interactivo** se aprovecha la tecnología SAP para ofrecer informes que van detallando la información bajo la interacción del usuario, es decir se realizan listados generales que se visualizan por pantalla y mediante teclas de función o seleccionando posiciones de pantalla, se irá desarrollando aquella información que solicite el usuario.

En contraste con el reporting clásico, que es asociado con procesos de entornos Batch, el reporting interactivo es un desarrollo que toma todas las ventajas del entorno online.

Un ejemplo de listado interactivo podría ser una lista de clientes, que permita visualizar sus datos de dirección, datos bancarios, partidas... etc. en pantallas diferentes , que van apareciendo conforme vamos seleccionando un cliente y/o solicitamos una función.

1.2 Generando listados interactivos.

Podemos encontrarnos con dos tipos de salida distintos :

- Una **lista básica.**
- Diversas **listas secundarias** (o de ramificación) , por ejemplo información detallada sobre datos de la lista básica.

La **lista inicial (básica)** la visualizaremos con la sentencia WRITE.

Desde esta lista inicial, el usuario será capaz, de dirigirse a **una lista secundaria** mediante una tecla de **función** o posicionándose con el cursor. Al igual que en las listas básicas, se implementan el listado con la instrucción WRITE.

Los datos de las listas secundarias aparecerán en una ventana, por encima de la lista principal, solapando una parte u ocultándola totalmente.Se podrán tener varias listas cada una de ellas en ventanas distintas.

Para controlar el flujo del report interacitivo, tendremos una serie de eventos como :

- **AT LINE-SELECTION** (si lo que se selecciona es una línea),
- **AT PFn** y
- **AT USER-COMMAND**(si se pulsa una tecla).

Un report interacivo puede tener como máximo 9 listados secundarios. Si el usuario selecciona la función **BACK** el sistema vuelve al listado anterior. ABAP/4 numera los listados a medida que se van generando, empezando desde 0 (listado básico).

El número del listado en proceso estará en la variable del sistema **SY-LSIND.**

En el listado básico podemos visualizar cabeceras con TOP-OF-PAGE o con la cabecera estándar, pero en los listados de ramificación no podemos utilizar cabeceras estándares, utilizaremos el evento **TOP-OF -PAGE DURING LINE-SELECTION.**

1.3 La interacción con el usuario.

El usuario tiene dos formas de interactuar con el sistema:

- Seleccionando una línea del listado (con doble-click) o utilizando la función Seleccionar. Para controlar la entrada del usuario utilizamos el evento de ABAP, **AT LINE-SELECTION.**

- Seleccionando una función, pulsando su botón correspondiente o una tecla de función. Para controlar la entrada del usuario utilizamos el evento de ABAP, **AT USER-COMMAND.** El código de la función solicitada lo podremos obtener con la variable **SY-UCOMM.**

Podemos conseguir que un campo de pantalla esté preparado para la entrada de datos con la cláusula **INPUT** de la instrucción WRITE.

WRITE <campo> INPUT.

El sistema sobreescribirá en pantalla el contenido del campo y guardará el nuevo valor en la variable **SY-LISEL,** después de un evento AT LINE-SELECTION o AT USER-COMMAND.

- **SELECCIÓN DE LINEA.**

La selección de una línea del listado interactivo , ya sea mediante un doble-click o mediante la función Seleccionar **(**con Código de función **'PICK'),** activa el evento **AT LINE -SELECTION**.

El proceso de este evento creará una nueva lista, que aparecerá por encima de la anterior. Para facilitar la orientación del usuario, podemos crear una ventana, (instrucción WINDOW), donde aparecerá los datos de esta nueva lista, solapándose parcialmente con la lista anterior.

En el evento AT LINE-SELECTION, utilizaremos los datos de la línea seleccionada, para leer directamente de base de datos y obtener los datos que necesitemos para la nueva lista. En la lista anterior**,** tendremos que guardar los datos clave que necesitamos para procesar la siguiente lista con la sentencia **HIDE.**

En el momento de procesar este evento el sistema llena una serie de variables del sistema. Ver Anexo 1 : **'Variables del sistema para reporting interactivo'** para más información.

Ejemplo:

```
TABLES LFAI.
GET LFA 1.
        WRITE LFA1-LIFNR.
        WRITE LFA1 -NAME1.
        HIDE LFAL-LIFNR.
AT LINE-SELECTION.
        IF SY-LSIND = 1.
                SELECT * FROM LFBK
                        WHERE LIFNR = LFA 1 -LIFNR.
                                WRITE LIFBK-BANKL,...
                ENDSELECT.
        ENDIF.
```

- **SELECCIÓN DE FUNCIÓN.**

La selección de una función (botón) o tecla de **función** activa el evento, **AT USER-COMMAND.** El código de la función solicitada se guarda automáticamente en la variable
SY-UCOMM. Si además el usuario selecciona una línea, el contenido de esta, podrá ser utilizado en el proceso.

Existe una serie de funciones que excepcionalmente no pueden activar el evento AT USER-COMMAND .Ni podrán ser utilizadas para realizar funciones propias:

PICK	**:**	Seleccionar (se utiliza AT LINE SELECTION)
PFn	**:**	Tecla de función(se utiliza AT PFn).
PRI	**:**	Imprimir
BACK	**:**	Volver pantalla anterior
RW	**:**	Cancelar
P…	**:**	Funciones de Scroll.

Ejemplo:

```
TABLES LFAI.
  GET  LFAI.
        WRITE LFA1-LIFNR.
         WRITE LFA1-NAME1.
         HIDE LFA1-LIFNR.

AT USER-COMMAND.
        CASE SY-UCOMM.
        WHEN 'BANK'.
                IF SY-LSIND = 1.
                        SELECT * FROM LFBK
                                WHERE LIFNR = LFA1-LIFNR.
                                WRITE LIFBK-BANKL,...
```

```
                ENDSELECT.
            ENDIF.
        ENDCASE.
```

Si el usuario pulsa una tecla de función asignada a un código de función PFn, se deberá utilizar el evento **AT PFn,** para procesar la entrada del usuario.

```
Ejemplo:  TABLES LFA1.
          GET LFA l.
              WRITE LFAl-LIFNR.
              WRITE LFA1-NAME.
              HIDE LFA1-LIFNR.

          AT PF5.        "              F5 =  Datos Bancarios
              IF SY-LSIND = 1.
                  SELECT * FROM LFBK
                      WHERE LIFNR = LFA1-LIFNR.
                      WRITE LIFBK-BANKL...
                  ENDSELECT.
              ENDIF.
```

En cualquier caso es mejor utilizar AT USER-COMMAND que AT PFn, ya que este último es menos flexible. Con el AT USER-COMMAND, podemos asignar una función a otra tecla de función sin tener que cambiar el programa y además, es posible que en diferentes listas la misma función se utilice para diferentes funciones.

1.4 Otras herramientas del reporting interactivo

- Para recuperar información seleccionada en un listado hay dos posibilidades

  - Utilizar el comando HIDE. (La más elegante).

    HIDE <campo>.

    HIDE guarda el contenido de <campo> de la línea de salida. Cuando visualicemos la siguiente lista, el sistema automáticamente llena el contenido de <campo> con el valor guardado en la instrucción HIDE.

  - Utilizar la variable del sistema SY-LISEL. (La más sencilla).

    En el momento que el usuario selecciona una línea, el contenido de esta se guardará en la variable del sistema SY-LISEL

- Podemos obtener la posición del cursor en el report con **GET CURSOR.**

GET CURSOR FIELD <campo>. :
Obtenemos el nombre del campo donde está situado el cursor

GET CURSOR LINE <línea>. :
Obtenemos la línea sobre la que está el cursor.

- También es posible situar el cursor en una posición determinada con:

  SET CURSOR <columna> <línea>.

  SET CURSOR FIELD <campo> LINE <línea>.

- Cambio de 'Status'.

  Un **Status** (GUI Status) describe que funciones están disponibles en un programa y como se pueden seleccionar, vía menús, teclas de función o 'pushbuttons'. Como ya veremos en el siguiente capítulo, podemos definir un Status mediante el '**Menu Painter**'.

Si un report genera diversas listas, cada una puede utilizar sus propias combinaciones de teclas. Con la instrucción **SET PF-STATUS** podemos cambiar la interface de cada lista.

SET PF-STATUS <status>.

Hay que tener en cuenta que existen diversas teclas de función reservadas por el sistema:

F1	:Help.
F2	:Seleccionar.
F3	**:**BACK.

F4	:Valores posibles.
F10	:Ir a la Barra de Menús.
F12	:Cancel.
F15	**:**Fin.
F21	**:**Scroll (lª. Página).
F22	**:**Scroll (Página anterior).
F23	**:**Scroll (Página siguiente).
F24	**:**Scroll (última Página).

En la variable **SY-PFKEY** tendremos el status del listado en curso.

- Hasta ahora todas las listas secundarias que creamos, aparecían cubriendo totalmente la lista anterior. La referencia a la lista anterior no es aparente. Para facilitar la orientación al usuario, podemos visualizar las listas secundarias en ventanas mediante la instrucción **WINDOW.**

WINDOW STARTING AT <columna> <línea>
(ENDING AT <columna> <línea>).

Especificamos las coordenadas _superior-izquierda en el **STARTING** e inferior-derecha en el **ENDING.**

Utilizaremos esta instrucción justo antes de los WRITE's de las listas secundarias.

- Manipular de listas

Para leer una línea determinada de la lista en visualización, utilizaremos la instrucción **READ LINE** después de un evento AT LINE-SELECTION o AT USER COMMAND.

READ LINE <línea>.

También podemos leer una línea de cualquier listado ramificado.

READ LINE <línea> INDEX <índice>.

Donde <índice> es el número de listado.

También es posible modificar una línea de un listado con el contenido de SY-LISEL

con el contenido de SY-LISEL con la instrucción **MODIFY LINE.**

MODIFY LINE <línea>.

Para obtener información detallada sobre la utilización de READ LINE y MODIFY LINE, ver la **documentación Online del editor de ABAP/4.**

- Es posible conectar reports interactivos con otros reports o con transacciones.

Podemos llamar a otro report con la instrucción SUBMIT.

SUBMIT <rcport> (WITH <datos>).

En este caso no se ejecutará la pantalla de selección del report que llamamos y asignaremos el criterio de selección con la cláusula **WITH.**

Para visualizar la pantalla de selección utilizamos la cláusula **VIA SELECTION SCREEN** del SUBMIT.

Después del SUBMIT, el programa que es llamado es quien tienen el control y el programa donde está el SUBMIT es liberado.

Si queremos volver al programa original en la siguiente instrucción al ,SUBMIT, utilizaremos la cláusula AND RETURN, es decir el programa original no será liberado.

Podemos llamar a un transacción con :

LEAVE TO TRANSACTION <cod_Tran>.

En este caso la transacción toma el control del proceso y el report es liberado.

Si queremos que una vez ejecutada la transacción el report recupere el control tendremos que hacer:

CALL TRANSACTION <cod_Tran>.

2 Programación de Diálogo

2.1 Introducción.

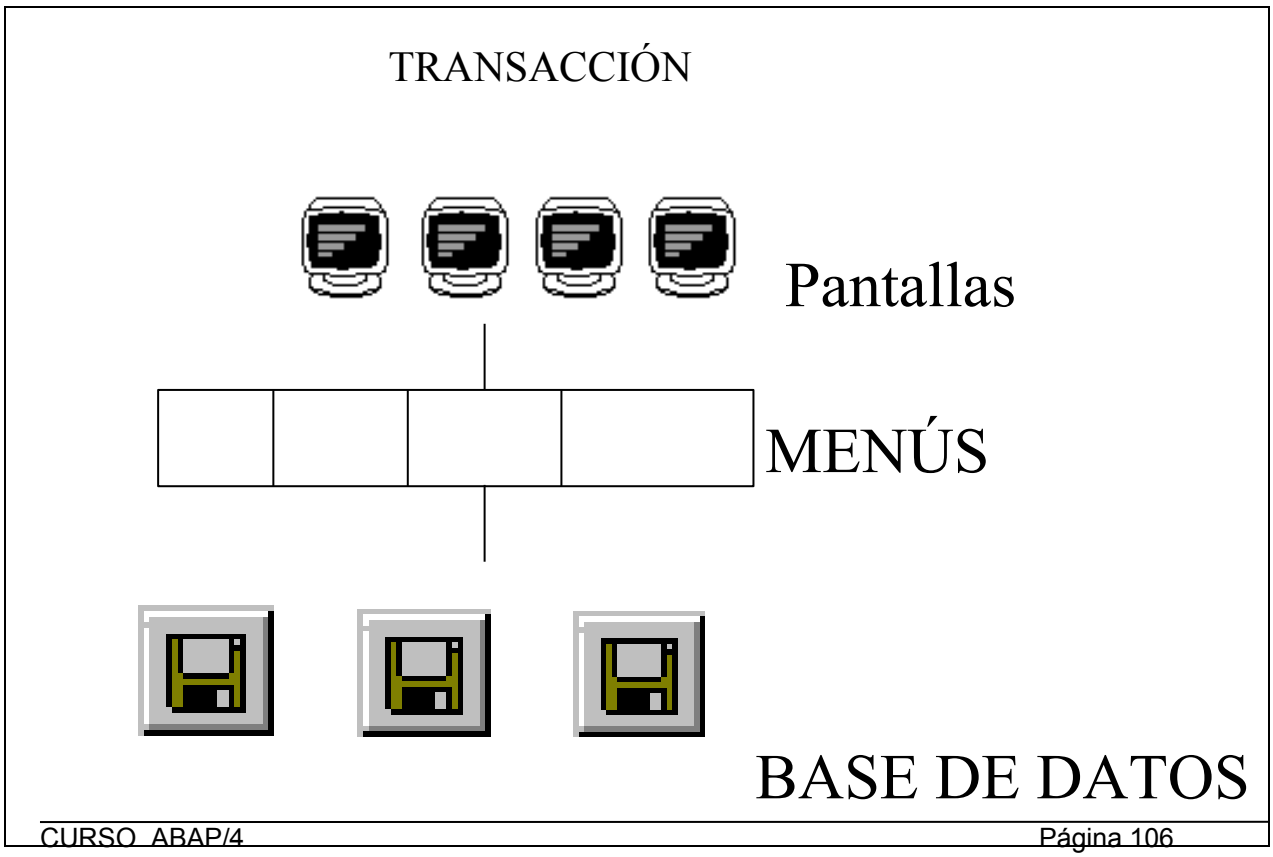
Hasta el momento, tanto en el capítulo anterior como en el manual **'Introducción al ABAP/4',** hemos estado describiendo el diseño de aplicaciones de tipo Reporting, ya sea Clásico o interactivo.

En este capítulo empezaremos a ver los fundamentos de la programación de diálogo para el diseño de **Transacciones** SAP.

La programación de diálogo necesita técnicas especiales de codificación en ABAP/4, además de herramientas específicas, como son un editor de pantallas **(Screen Painter)** y un editor de superficies **(Menú Painter).**

2.2 Pasos en la creación de transacciones.

Para crear una transacción, será necesario la generación de una serie de objetos de desarrollo. Cada transacción puede dividirse en varias pantallas, cada una de las cuales puede utilizar distintos menús y todo ello controlado por un programa en ABAP/4 denominado **Module Pool,** que controla el flujo de la transacción y realiza las acciones necesarias para cumplir la funcionalidad de la transacción.



Por lo tanto los pasos a seguir para el desarrollo de transacciones será :

1º. Crear el **código de transacción.**

Herramientas -> Workbench ABAP/4 -> **Desarrollo -> Transacciones.**

Indicándole : El tipo de transacción, la descripción de la transacción, El nombre del programa ABAP/4 (Module Pool), el número de la primera pantalla y opcionalmente un objeto de verificación para ejecutar la transacción.

2º. Crear el programa ABAP/4 **(Module Pool).**

3º. Definir las **pantallas** que intervienen en la transacción con el **Screen Painter.**

Especificando que datos aparecen en pantalla y de que forma, además de una lógica de proceso de cada pantalla.

4º. Definir los **menús** con el **Menú Painter.**

Especificando el contenido de los menús Pop-up, las teclas de función y los botones de comandos que se pueden utilizar.

5º. Definir el **Flujo de pantallas** en el **Module Pool.**

6º. Programar, en el Module Pool, los **módulos** de cada pantalla, es decir lo que debe hacer cada pantalla. Programando las acciones a realizar en tiempo de PBO **('Process Before Output'),** antes de que aparezcan los datos de la pantalla y en tiempo de **PAI ('Process After Input'),** después de que se hayan introducido los datos en los campos de entrada.

3 Diseño de Menús (MENU PAINTER). (Release 3.0)

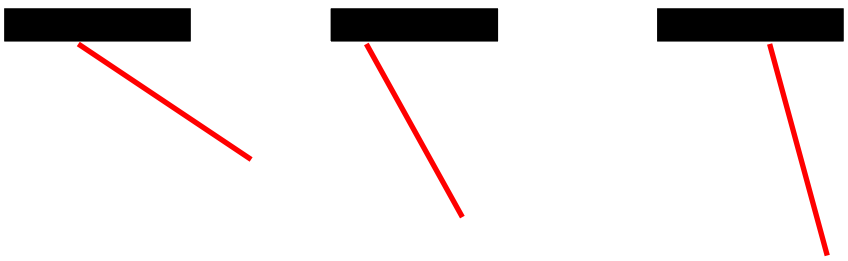
3.1 Introducción.

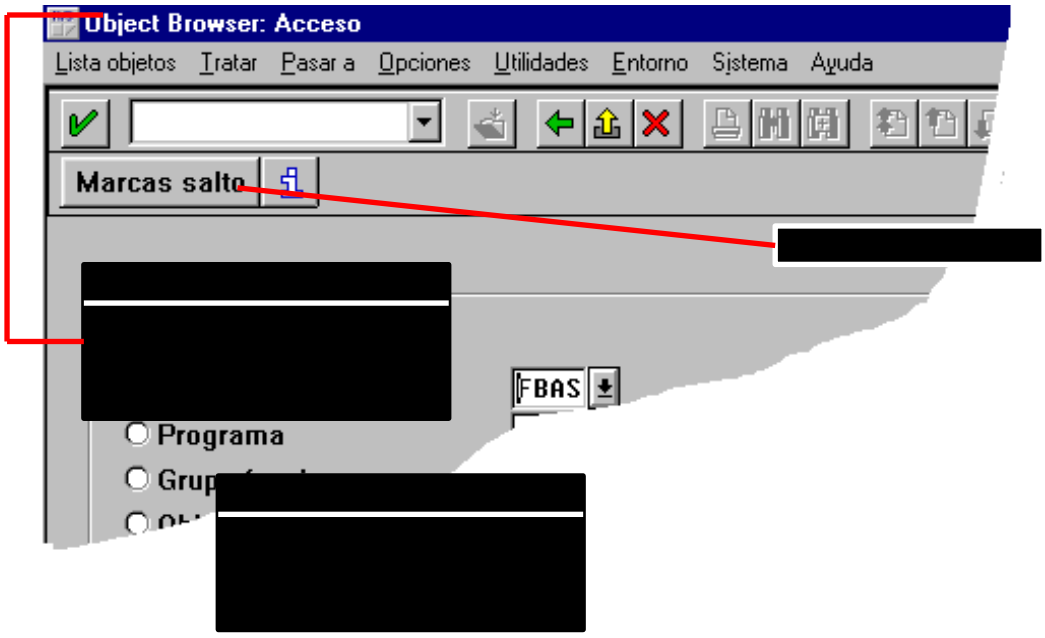
Con el **Menu Painter** diseñaremos las superficies GUI, (Grafical User Interface), sobre las que correrán las transacciones SAP.

Una GUI contiene todos los menús, teclas de función, pushbuttons, etc... disponibles para el usuario, durante la ejecución de una transacción.

Podremos indicar el status que utilizamos en una pantalla o el título en un módulo PBO de la pantalla con las instrucciones :

SET PF-STATUS <cod_ status>.
SET TITLEBAR <cod_título>.





Identificamos las diferentes interfaces GUI de una transacción mediante los **Status** Una transacción tendrá muchos status diferentes. No será necesario redefinir todos los objetos de los status, ya que muchos objetos definidos en un status podrán ser utilizados en otro. Por ejemplo es posible crear una barra de menús igual para toda la transacción.

Para iniciar el Menú PAINTER**,** seleccionar : **'Tools -> ABAP/4 WORKBENCH -> Desarrollo -> Menu Painter. (SE41).** Es posible mantener tanto un status de un determinado programa, como los diferentes objetos de una GUI que forman parte de los status (barras de menús, teclas de función, títulos de menú.…).

3.2 La Barra de Menús.

En primer lugar introduciremos las distintas opciones de la barra de menús, que las iremos desarrollando en funciones o en otros submenús, entrando los nombres en la parte superior de la pantalla. Se pueden incluir hasta 6 menús en la barra de menús. Además de los menús de usuario, elsistema añadirá automáticamente **System y Help**.

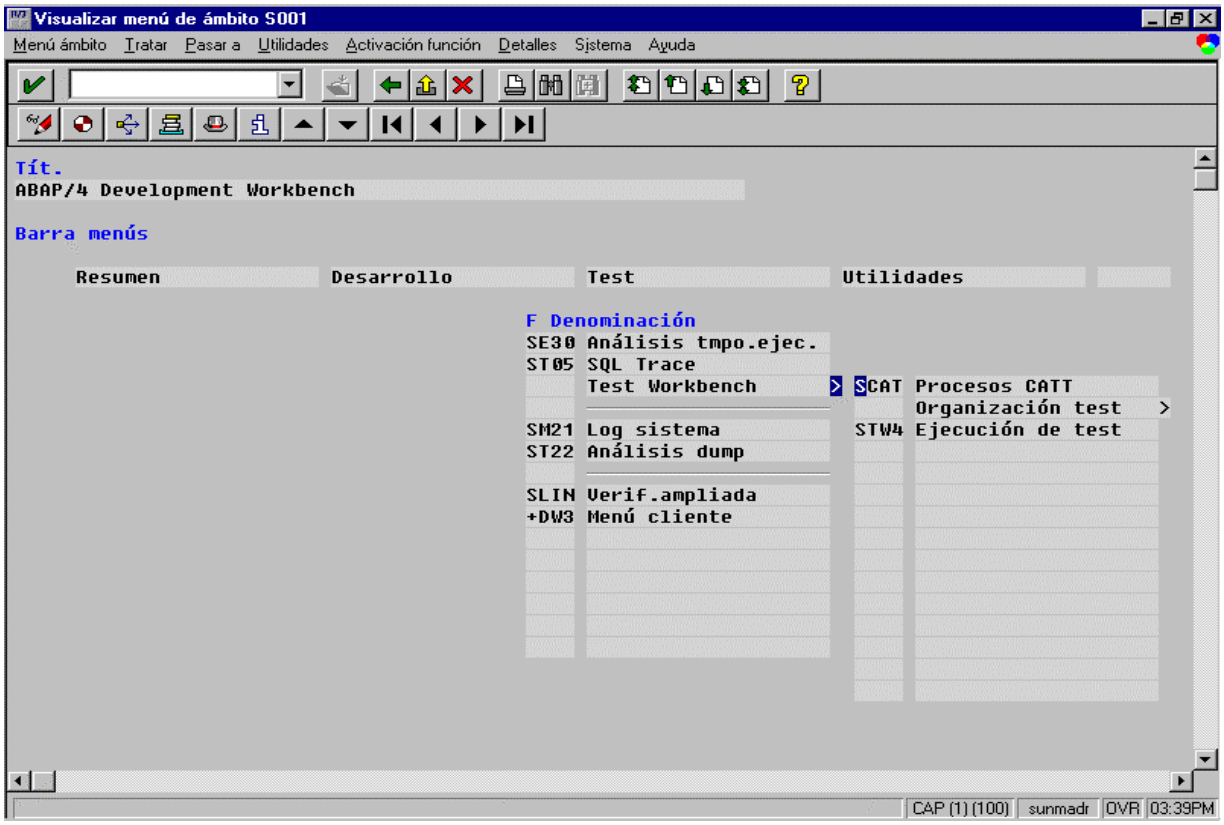
Cada menú puede tener hasta 15 entradas. Cada una de las cuales puede ser otro menú en cascada o una función.

Para abrir un menú o submenú, hacer un Doble-Click sobre el nombre. Cada entrada estará compuesta de un código de función y un texto de función o un texto de menú. Con F4 podemos ver una lista de las funciones que podemos utilizar.

Se pueden anidar hasta 4 niveles de submenús.

En el caso de las funciones, bastará con indicar el código de la función para que el texto de esta aparezca automáticamente, si esta ya existe previamente. Podemos definir los atributos de una función nueva con Doble-Click sobre la nueva función definida.

En el caso de un menú en cascada, no será necesario indicar el código, y con Doble-Clíck podemos desarrollar las opciones del submenú.



3.3 Los 'Pushbuttons'.

Los Pushbuttons son botones tridimensionales que aparecen debajo de la barra de herramientas estándar.

Previamente a definir un Botón será necesario definir la función deseada como una **tecla** de función (apartado **3.4).**

Para ver que funciones se pueden utilizar, nos situaremos sobre **'Aplication toolbar'** y pulsaremos **F4,**

Indicaremos el código de función que deseamos que aparezca en la barra de herramientas de aplicación. Podemos especificar si queremos que aparezca un texto corto o únicamente un icono que identifique la función.

No será necesario definir las funciones de la barra de herramientas estándar, **'Standard Toolbar'.**

Para definir iconos para visualizarlos en la barra de herramientas de aplicación será necesario :

Seleccionar: **Edit -> Insert -> Function with icon.**

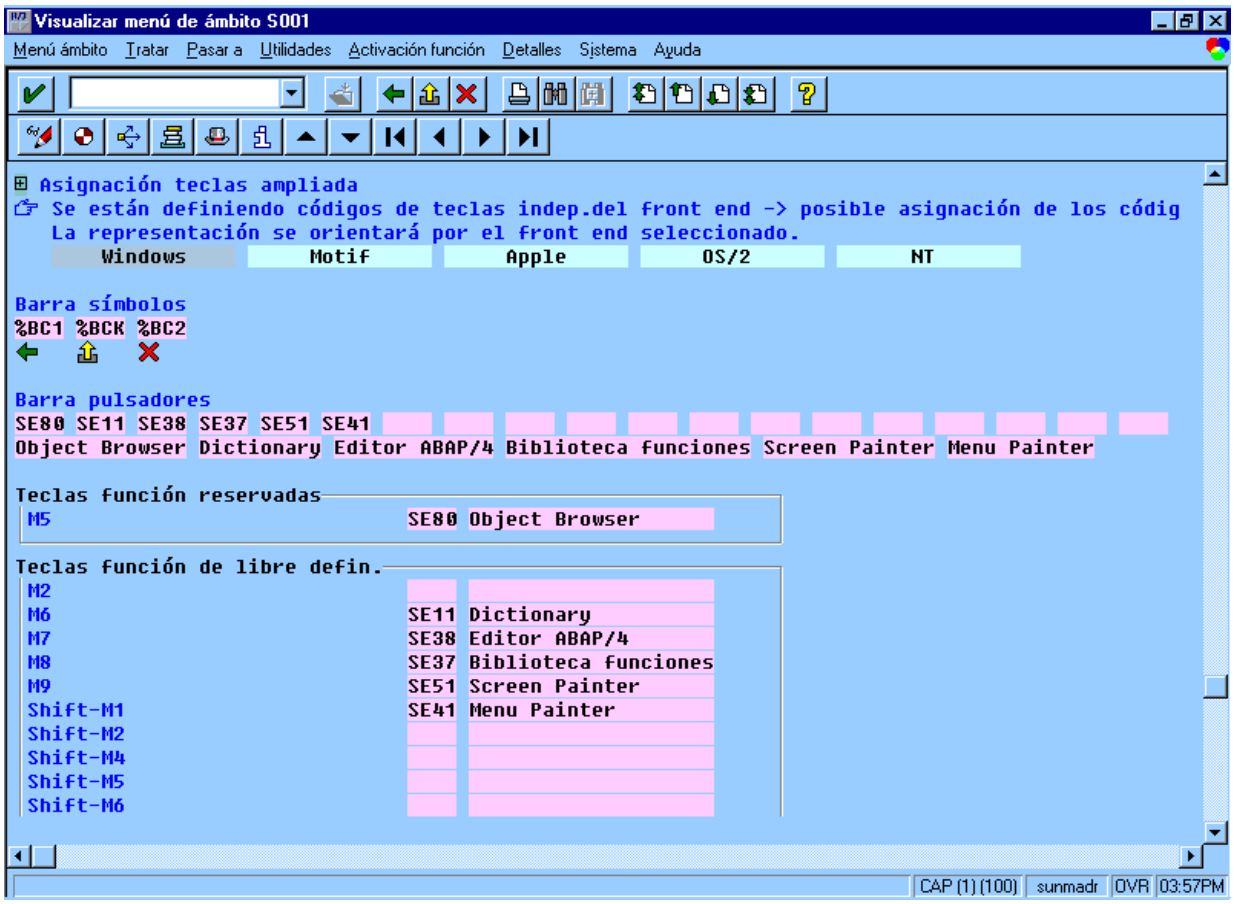
Entrar el código de función

Introducir el nombre del icono y el texto del menú.

3.4 Teclas de Función.

Para definir las teclas de función utilizamos el espacio destinado para ello. Indicando el código de la función en la línea correspondiente a la tecla que deseamos utilizar. El texto de la tecla de función aparecerá automáticamente, pero podrá ser modificado en caso de desearlo.

SAP no recomienda definir nuevas teclas de función en el espacio reservado para teclas de función estándar.



3.5 Otras utilidades del Menú Painter.

3.5.1 Activación de funciones.

Podemos hacer que las funciones de la barra de menús estén en modo **activo** o **inactivo.** En caso de estar inactivas, se visualizarán en la barra de menús en baja intensidad y su selección no implicará efecto alguno, en cambio las funciones activas serán completamente ejecutables.

Para activar o desactivar funciones seleccionar **'Function activation'.**

3.5.2 'FastPaths'

Un 'FastPaths ' (Camino rápido), es una manera rápida de escoger funciones en un menú, asignado a cada función una tecla.

Podemos mantener 'FastPaths' seleccionando:

Goto → Further options → Fastpath.

Indicaremos para las funciones deseadas una letra, normalmente la primera, teniendo cuidado de no especificar la misma letra para distintas funciones.

3.5.3 Títulos de Menú.

Es posible mantener distintos títulos para un menú.

Goto -> Title list

Cada título se identificará con un código de título de tres dígitos.

Introduciremos el texto del título, pudiendo utilizar variables de la misma forma que lo hacíamos con los mensajes en ABAP/4, es decir utilizando el símbolo &. Posteriormente será en el programa ABAP/4 donde le indiquemos que título vamos a utilizar con la instrucción:

SET TITTLEBAR <cod-título> WITH <varl> <var2> ....

En tiempo de ejecución el título del menú se guardará en la variable del sistema SY-TITLE.

3.5.4 Prueba, chequeo y generación de Status.

Podemos probar el Status simulando la ejecución de la interface con :**User Interface -> Test Status** , e introduciendo los datos: número de pantalla, y,código del título.

Antes de usar la interface podemos comprobar que la hemos definido correctamente, realizando un proceso de chequeo con: **User Interface -> Check Syntax**. Posteriormente realizaremos un proceso de generación de la interface que incluye e1 chequeo y la grabación de la misma.

3.6 Menus de Ambito o de área.

Un Menú de ámbito es una agrupación de transacciones en forma de menú.Es una manera de agrupar las transacciones más frecuentemente utilizadas por un usuario bajo un mismo menú. A diferencia de una transacción de diálogo, el menú de ámbito sólo llama a otras transacciones, no pudiendo incorporar otro tipo de funciones propias.

Podemos crear los menús de ámbito con una versión simplificada del Menu Painter.

Herramientas →Workbench ABAP/4 → Desarrollo → Más herramientas→Menus ámbito

Únicamente será necesario introducir los códigos de transacción (Tabla TSTC) y el texto del menú.

Para más información sobre las posibilidades del Menú Painter, ver el capítulo **Menu Painter** del manual 'BC **ABAP/4 Workbech tools'.**

4 Diseño de Pantallas (SCREEN PAINTER). (Release 3.0)

4.1 Introducción al diseño de pantallas.

El Screen Painter nos permite 'pintar' las pantallas de SAP y diseñar su comportamiento (lógica de proceso).

En SAP, al conjunto de pantalla y lógica de proceso de la misma se le denomina Dynpro ('Dynarnic Program').

Existen dos modos de funcionamiento del editor de pantallas : El **modo gráfico** y el **modo alfanumérico,** dependiendo del interface gráfico sobre el que funcione SAP.

En este capítulo se describe el uso del Screen Painter en modo gráfico (versión 3.0), ya que se le considera como el más cómodo y avanzado, siendo además soportado por los interfaces gráficos más extendidos (MS WINDOWS y XI l/MOTIF UNIX). De cualquier modo la funcionalidad de ambos modos del editor es la misma.

Para activar o desactivar el modo gráfico del editor de pantallas ir a :

Settings -> Graphical Fulllscreen

4.2 Diseño de pantallas.

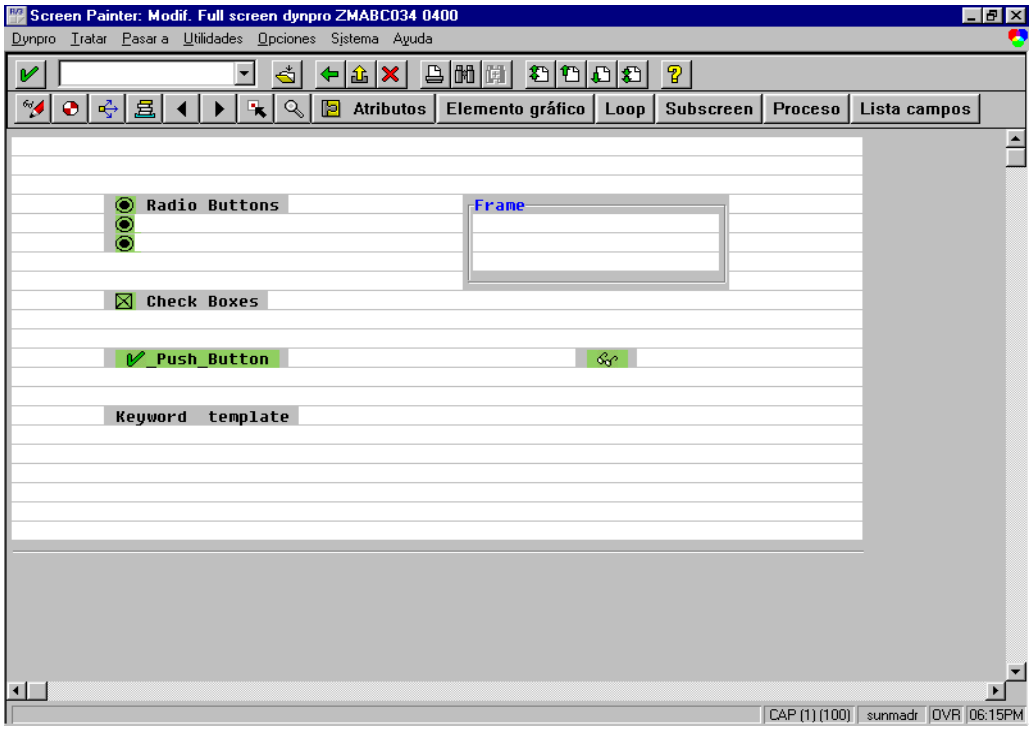
4.2.1 Utilizando el Screen Painter.

Una pantalla SAP se identifica por el nombre del programa module pool de la transacción a la que pertenece, más un número de pantalla.

Así tras acceder al Screen Painter desde el ABAP/4 Workbench, tendremos que introducir el programa y el número de pantalla que deseamos mantener. Una vez hecho esto aparecerá el Editor de Pantallas **'Fullscreen Editor'**.

Si estamos creando el dynpro por primera vez, nos pedirá los atributos de pantalla:

Descripción
Tipo de pantalla (normal, de selección, modal, Subscreen).
Código de la siguiente pantalla.
Campo donde se situará el cursor.
Grupo de pantallas
Tamaño máximo de la pantalla.



En el editor de pantallas podemos observar 3 áreas diferenciadas :

- La **cabecera :** Con datos sobre la pantalla y el campo se está manteniendo en ese preciso instante**.**

- La **barra de objetos** (columna izquierda) : Lista de los objetos que se pueden crear en la pantalla : Textos, entrada de datos, 'checkboxes', 'frames', 'subscreens' ...

- El **área de dibujo**: Es el área donde se dibuja la pantalla que estemos diseñando.

4.2.2 Creando objetos en 1a pantalla.

Para dibujar un objeto en la pantalla tendremos que colocarlo en el área de trabajo y posteriormente **definir** sus características (atributos). Para ello tendremos que
pulsar el botón correspondiente en la barra de objetos y marcar el área donde vamos a
situar el objeto.

Si queremos cancelar la creación de un objeto pulsaremos el botón **Reset** de la misma barra de objetos.

Objetos disponibles :

- **Textos :** Textos, literales,... que son fijos en pantalla y no pueden ser manipulados por el usuario. Para considerar varias palabras como un mismo texto tendremos que colocar un símbolo '-' entre ellas, ya que de otro modo las considerará como objetos de texto completamente distintos.

- **Objetos de entrada/ salida ('Templates') :** Son campos para introducir o visualizar datos.Pueden hacerse opcionales o obligatorios.Los caracteres de entrada se especifican con el símbolo '_', pudiendo utilizar otros caracteres para dar formato a la salida. Por ejemplo una hora podemos definirla como:
  __:__:__.

- **Radio-Buttons :** Son pequeños botones redondos, que permiten una entrada de dos valores sobre una variable (marcado o no marcado). Los podemos agrupar, de forma que la selección de uno implique que no se pueda seleccionar ningún otro.

- **Check-Boxes :** Es como un radio-button pero de apariencia cuadrada en vez de redonda. La diferencia respecto los radio-buttons deriva en su utilización en grupos, ya que se pueden seleccionar tantos checks-boxes como se quiera dentro de un grupo.

- **Pushbuttons :** Es un botón de tipo pulsador. Se le asocia a una función, de forma que en el momento que se pulsa el Pushbutton se ejecute la función.

- **Frames (Cajas) :** Son Cajas que agrupan grupos de objetos dentro de una pantalla como por ejemplo un conjunto de radio-buttons.

- **Subscreen:** Son áreas de la pantalla sin ningún campo que se reservan para la salida de otras pantallas (subscreens) en tiempo de ejecución. Para definir este área, nos colocaremos el punto de la pantalla donde queremos situar el ángulo superior izquierda de la Subscreen, seleccionaremos **Edit -> Subscreen**, indicándole el nombre que le vamos a dar, y finalmente señalaremos con doble-click, el ángulo inferior derecha de la ventana.

  Posteriormente será en los módulos PBO y PAI cuando le indicaremos con la instrucción **CALL SUBSCREEN,** qué pantalla aparecerá en el área de Subscreen que hemos definido.

  Una vez situados los objetos sobre el área de trabajo, podremos modificar su tamaño, moverlos o borrarlos.

  Todos los textos, pushbuttons... pueden incorporar iconos en su salida por pantalla. Los iconos tienen una longitud de dos caracteres y están representados por símbolos estándares. El icono será un atributo más de los campos y por tanto se definirán junto al resto de atributos del objeto.

4.2.3 Creando objetos desde el diccionario de datos.

En la pantalla que estamos diseñando, podemos utilizar campos que están guardados en el diccionario de datos o declarados en el module pool- Para ello tendremos que seleccionar: **Goto -> Dict / Program fields.**

Aparecerá una pantalla de selección de datos en la que indicaremos el campo o la tabla de la cual queremos obtener datos. Además se deberá seleccionar, si queremos ver la descripción de cada campo (indicando la longitud) y si queremos realizar una entrada de dalos ('template') de dicho campo por pantalla. Finalmente pulsaremos el botón correspondiente a crear desde el diccionario de datos o desde un programa. Marcaremos el campo que queremos incorporar a nuestra pantalla y los copiaremos sobre el área de trabajo, situándolos en la posición que creamos más conveniente.

4.2.4 Definiendo los atributos individuales de cada campo.

Los atributos de los objetos definen las características de estos.

Podemos mantener los atributos desde el mantenimiento de atributos de campo o desde listas de campos.

Podemos distinguir entre atributos generales, de diccionario, de programa y de visualización

Atributos Generales:

- **Matchcode:** Permite especificar un matchcode para la entrada de un campo.
- **References :** Especificamos la clave de la moneda en caso de que el campo sea de tipo cantidad (CURR o QUAN).
- **Field type:** Tipo de campo.
- **Field Name:** Nombre del campo. Con este nombre se identificarán desde el programa.
- **Field text:** Texto del campo. Si queremos utilizar un icono en vez de texto dejaremos este valor en blanco.
- **With icon :** si queremos utilizar iconos en entrada de datos ('templates').
- **Icon name :** Identifica el nombre de un icono para un campo de pantalla.
- **Rolling (Scrolling):** Convierte un campo en desplegable, cuando su longitud real es mayor que su longitud de visualización.
- **Quick Info :** Es el texto explicativo que aparece cuando pasarnos por encima de un icono con el ratón.
- **Line :** Especifica la línea donde el elemento aparecerá. El sistema completa este valor automáticamente.

- **CI:** Especifica la columna donde el elemento aparecerá. El sistema completa este valor automáticamente.

- **Ht:** Altura en líneas. El sistema completa este valor automáticamente.

- **Dlg :** Longitud del campo.
- **Vlg:** longitud de visualización.

- **FctCode:** Código de función (código de 4 dígitos). Atributo sólo para pushbottons.

- **FctType:** Especifica el tipo de evento en el cual el campo será tratado.

  Ltype: Tipo de step loop (fijo o variable). El tipo variable significa que el tamaño del step loop se ajusta según el tamaño de la pantalla, mientras que fijo no ajusta el step loop.

- **Lent :** Número de líneas de un step loop.

- **Groups:** Identifica grupos de modificación para poder modificar varios campos simultáneamente. Podemos asignar un campo a vanos (4) grupos de modificación.

Atributos de Diccionario:

- **Format:** Identifica el tipo de datos del campo. Determina el chequeo que realiza el sistema en la entrada de los datos.

- **Frm DICT.:** El sistema rellena este atributo en el caso de que el campo lo hayamos creado a partir de un campo del diccionario de datos.

- **Modific.:** El sistema rellena este campo si detecta alguna diferencia entre la definición del campo en el diccionario de datos y su utilización en pantalla.

- **Conv. Exit:** Si queremos utilizar una rutina de conversión de datos no estándar, especificamos aquí el código de esta.

- **Param. ID:** Código del parámetro SE]7GET. (Ver siguiente atributo)-

- **SET paramGET param :** Los parámetros **SPA** (Set Parameter) y **GPA** (Get Parameter), nos permiten visualizar valores por defecto en campos. Si marcamos el atributo SET **param,** el sistema guardará en un parámetro ID lo que entremos en este campo. Si marcamos el atributo **GET param**, el sistema inicializa el campo, con el valor del parámetro ID que tenga asignado en el atributo anterior.

- **Up../lower:** El sistema no convierte la entrada a mayúsculas.

- **W/o template:** Marcamos este atributo si queremos que los caracteres especiales se traten como textos literales.

- **Foreign key:** Si queremos que sobre el campo el sistema realice un chequeo de clave externa. (Hay que definir previamente las claves externas en el diccionario de datos).

Atributos de programa :

- **Input field :** Campo de entrada.

- **Output field:** Permite visualización. Se puede utilizar en combinación con el anterior.

- **Output only :** Sólo visualización.

- **Required field :** Atributo para campos obligatorios. Se distinguen con un **?**.

- **Poss. Entry:** El sistema marca este atributo si hay un conjunto de valores para el campo. No es posible modificar el contenido del atributo.

- **Poss. Entries:** Indica como podemos ver la flecha de entradas posibles en un campo.

- **Right-justif:** Justifica cualquier salida del campo a la derecha.

- **Leading zero:** Rellena con ceros por la izquierda en el caso de salidas numéricas.

- ***-entry :** Permite la entrada de un asterisco en la primera posición de un campo. Si se introduce un * se podrá hacer un tratamiento en un módulo:

FIELD MODULE ON *-INPUT.

- **No reset:** Cuando activamos este atributo, la entrada de datos no podrá ser cancelada mediante el carácter **!**.

Atributos de visualización:

- **Fixed font** : Visualizar un campo de salida en un tamaño fijo (no proporcional). Sólo se puede utilizar en campos Output only.

- **Briglit :** Visualiza un campo en color intenso.

- **Invisible:** Oculta un campo.

- **2-dimens**: Visualiza un campo en dos divisiones en vez de en tres.

4.3 Lógica de proceso de una pantalla.

4.3.1 Introducción a la lógica de proceso.

Una vez hemos definido gráficamente las pantallas, será preciso escribir una lógica de proceso para cada una de ellas, pasándose a denominar dynpros.
Para introducir la lógica de proceso de las pantallas, utilizaremos una versión especial del editor de ABAP/4. Goto -> Flow logia.
La lógica de proceso de las pantallas tienen una estructura determinada, y utilizan comandos y eventos propios de manejo de pantallas, similares a los utilizados en ABAP/4.

Consistirá en dos eventos fundamentales:

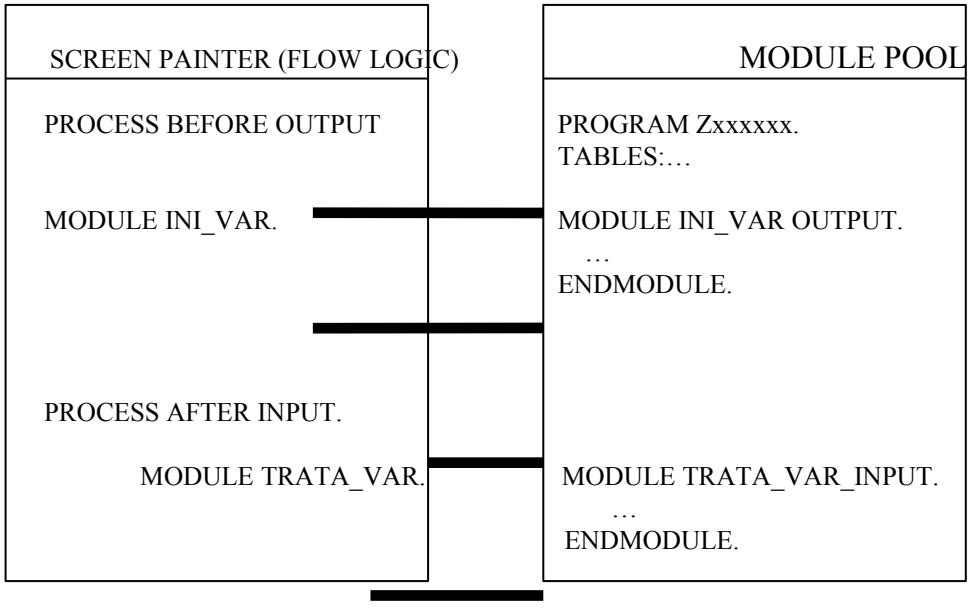
PROCESS BEFORE OUTPUT (PBO).

PROCESS AFTER INPUT (PAI).

El **PBO**, será el evento que se ejecutará previamente a la visualización de la pantalla, mientras que el **PAI**, se ejecutará después de la entrada de datos del usuario en la pantalla.

Además de estos dos eventos obligatorios, existen eventos para controlar las ayudas, **PROCESS ON HELP-REQUEST (POH),** y para controlar los valores posibles de un campo **PROCESS ON VALUE-REQUEST (POV).**

Desde la lógica de proceso de las pantallas no se actualizan datos, únicamente se llamara a los módulos del module pool que se encargan de esta tarea.



4.3.2 Process Before Output (PBO).

En el Módulo PBO, indicaremos todos los pasos que queremos realizar antes de que la pantalla sea visualizada, como por ejemplo inicializar los campos de salida, Esta inicialización se realizará en un módulo independiente dentro del module pool-

Para llamar a un módulo utilizaremos la sentencia **MODULE.**

MODULE <nombre modulo>.

Ejemplo :

```
PROCESS BEFORE OUTPUT.
*
      MODULE inicializar_campos.
*
PROCESS AFTER INPUT.
```

En el Module pool el código del módulo empezará con la sentencia:

MODULE <nombre_modulo> OUTPUT.

En el caso de ser un módulo llamado desde el PAI será:

MODULE <nombre-inodulo> INPUT.

Dentro del module pool declararemos los campos de pantalla que vayamos a utilizar y en el módulos del PBO los inicializamos al valor que queramos. Si no inicializamos explícitamente, el sistema le asignará su valor inicial por defecto, a no ser que lo hayamos definido como parámetro SPA / GPA.

Además de inicializar los campos de entrada de datos, el evento PBO, puede ser utilizado para :

- Seleccionar el Status (menú) o el título de los menús con:

  SET PF-STATUS <GUI_status>.

  SET TITLEBAR <titulo> WITH <p1> <p2> <p3> <p4>.

- Desactivar funciones de un menú con **:**

  SET PF-STATUS <GUI-status> EXCLUDING <function-code>.

Si se quiere desactivar más de una función, se le indicará en el EXCLUDING, una tabla interna con los códigos de función que queremos desactivar. La estructura de la tabla interna será :

```
DATA: BEGIN OF tabint OCCURS 20,
            FUNCTION (4),
```

END OF tabint.

- Modificar los atributos de la pantalla en tiempo de ejecución. Para ello disponemos de la tabla **SCREEN,** donde cada entrada se corresponde a un campo de la pantalla que puede ser leída y modificada con LOOP's y MODIFY'S.

La estructura de la tabla SCREEN es :

NAME	30	Nombre del campo de pantalla.
GROUP1	3	Campo pertenece al grupo de campo 1.
GROUP2	3	Campo pertenece al grupo de campo 2.
GROUP3	3	Campo pertenece al grupo de campo 3.
GROUP4	3	Campo pertenece al grupo de campo 4.
ACTIVE	1	Campo visible y listo para entrada de datos.
REQUIRED	1	Entrada de datos obligatoria.
INPUT	1	Campo apto para entrada de datos.
OUTPUT	1	Campo sólo visualización.
INTENSIFIED	1	Campo en color intensificado
INVISIBLE	1	Campo invisible.
LENGTH	1	Longitud de salida reducida.
REQUEST	1	Campo con Valores posibles disponibles
VALUE HELP	1	Campo con Help disponible.
DISPLAY_3D	1	Campo tridimensional

- Inicializar radio-buttons y check-boxes. Los declararemos como caracteres de longitud 1 y los inicializamos como:

```
Radio1 = 'X'
Check1 = 'X'.
```

En el caso de los radio-buttons, que pueden funcionar en grupo, de forma que la activación de un botón desactive otro, no será necesario codificar este comportamiento manualmente, ya que lo realizará el sistema automáticamente.
Para definir un grupo de radio-buttons, seleccionamos los botones del grupo y marcar: **Edit -> Radio Button Group.** En caso de utilizar el editor Alfanumérico : Seleccionamos el primer objeto de grupo, escoger .**Edit -> Graphical element,** Seleccionar el último objeto del grupo y pulsar : **Define Graph group.**

- Para utilizar subscreens será necesario realizar las llamadas a las mismas en tiempo de PBO y de PAI.

```
PROCESS BEFORE OUTPUT.

CALL SUBSCREEN <área> INCLUDING <programa> <screen>.
```

PROCESS AFTER INPUT.
CALL SUBSCREEN <área>.

Donde: **<área> :** es el nombre qlie le hemos dado al área de sul)screcii en el Screen Painter.

<programa> : Es el nombre del programa que controla el subscreen.

<scrcen> : Es el número de pantalla.

La subscreen se ejecutará como una pantalla normal y corriente con su PBO y PAI correspondiente que son llamados desde el PBO y el PAI de la pantalla principal.

- Para manipular la posición del cursor.
  Por defecto el sistema sitúa el cursor en el primer campo de entrada de la pantalla, pero es posible que queramos situarlo en otro campo:

SET CURSOR FIELD <nombre_campo>.

También es posible que en algún momento sea interesante conocer en qué campo está situado el cursor. Para hacer esto utilizamos:

GET CURSOR FIELD <var_campo>.

4.3.3 Process After Input (PAI).

El PROCESS AFTER INPUT se activa cuando el usuario selecciona algún punto de menú, pulsa alguna tecla de función o pulsa ENTER. Si alguno de estos eventos ocurre, el PAI de la pantalla necesitarán responder apropiadamente a la función seleccionada por el usuario.

4.3.3.1 La validación de los datos de entrada.

Una de las funciones más importantes del Proces After Input, es la de **validar** los **datos de entrada** de la pantalla antes de ser usados. Existen dos tipos de validación de los datos de entrada: Un **chequeo automático** realizado por el sistema y un **chequeo manual** programado con el comando **FIELD** de la lógica de proceso de dynpros.

- **Chequeo automático.**

El sistema realiza automáticamente una serie de chequeos de los datos de entrada, antes de procesar el evento PAI. Por ejemplo el sistema valida que se introduzcan aquellos campos que sean obligatorios, que el formato de los datos sea el correcto y

que el usuario introduzca un valor correcto en el campo (si en el diccionario de datos hay claves externas o valores fijos para un campo).
Si el chequeo automático detecta alguna inconsistencia, el sistema obliga al usuario a introducir nuevamente un valor en el campo erróneo.

Es posible que en alguna ocasión el usuario quiera salir de la pantalla sin necesidad de pasar las validaciones automáticas (Por ejemplo utilizando las funciones estándares BACK, EXIT o CANCEL). En ese caso utilizaremos la cláusula **AT EXIT COMMAND** de la instrucción MODULE.

MODULE <modulo-ABAP> AT EXIT-COMMAND.

Con AT EXIT COMMAND, podemos ir a un módulo de ABAP/4 antes de que el sistema realice las validaciones automáticas para un campo**.**

Para poder utilizar un AT EXIT COMMAND en un botón un campo, será necesario asignar el valor **E** en el atributo de campo **FctType** del editor de pantallas o en las funciones del Menú Painter.

En el módulo que llamamos incluiremos las instrucciones necesarias para poder salir de la transacción o de la pantalla en proceso, por ejemplo para salir de la transacción utilizaremos

LEAVE TO SCREEN 0.

- **Chequeo Manual.**

Además del chequeo automático es posible realizar una validación más extensa de los valores de entrada a los campos con las instrucciones **FIELD** y **CHAIN** de la lógica de proceso del Screen painter.

Con FIELD podemos validar un campo y con CHAIN … FIELD f1 … FIELD f1 … ENDCHAIN podemos validar más de un campo en el mismo procedimiento de chequeo.

Con **FIELD** podemos validar individualmente cada campo, de forma que en caso de error, la siguiente entrada de datos sólo permitirá introducir el campo erróneo sobre el que estamos utilizando la instrucción FIELD.

Dependiendo del tipo de sentencia FIELD que utilicemos, el mecanismo de chequeo se realizará en la lógica de proceso del Screen painter o en un módulo ABAP/4.

Es posible realizar distintas validaciones de un campo de entrada dependiendo de la fuente con la que contrastamos los valores posibles. Así podemos chequear el contenido de un campo comparándolo con una tabla de la base de datos, con una lista de valores, o realizando la validación en un módulo del module pool.

Para **chequear un campo contra la base de datos** utilizamos :

FIELD <campo_pantalla> SELECT FROM <tabla>
WHERE <campo_tabla> = <entrada_campo_pantalla>.

Si no se encuentran registros en el diccionario de datos el sistema emite un mensaje de error estándar.

Existe una versión ampliada de la instrucción anterior que permita enviar mensajes o warnings en caso de que encuentre o no registros:

FIELD <campo_pantalla> SELECT * FROM <tabla>
WHERE <condición>
WHENEVER (NOT) FOUND SEND
ERRORMESSAGE / WARNING <numero>
WITH <campo_texto>.

Para **chequear un campo respecto una lista de valores** utilizamos:

FIELD <campo_pantalla> VALUES (<lista_valores>).
Donde **<lista de valores>** puede ser :

('<valor>')
(NOT '<valor>')
('<valorl>', '<valor2>',...NOT '<valorn>')
(BETWEEN'<valorl>'AND'<valor2>')
(NOT BETWEEN'<valorl>'AND'<valor2>')

Si el valor entrado por el usuario no corresponde a ningún valor de la lista, el sistema emite un mensaje de error.

Para **chequear un campo en un modulo de ABAP/4** utilizamos :

FIELD <campo_pantalla> MODULE <modulo ABAP/4>.

También es posible condicionar la ejecución de los módulos ABAP/4 :

. FIELD... MODULE.... **ON INPUT :** Se ejecuta el módulo si el campo tiene un valor distinto del inicial (distinto de blancos o ceros).

. FIELD... MODULE.... **ON CHAIN INPUT :** Se ejecuta el módulo si algún campo del CHAIN tiene un valor distinto del inicial.

. FIELD...MODULE....**ON REQUEST :**Se ejecuta el módulo si el campo ha sido cambiado desde su visualización en pantalla. Aunque le demos el valor inicial.

. FIELD... MODULE.... **ON CHAIN REQUEST :** Se ejecuta el módulo si el algún campo a del CHAIN ha sido cambiado desde su visualización en pantalla.

.FIELD…MODULE…**ON *-INPUT**: Se ejecuta el módulo si el usuario introduce un * en el campo y el campo tiene el atributo ***-entry.**

.FIELD…MODULE… **AT CURSOR SELECTION**: Se ejecuta el módulo si el campo ha sido seleccionado.

- Con CHAIN podemos chequear múltiples campos simultáneamente, combinándolo con la instrucción FIELD.

La instrucción CHAIN ... ENDCHAIN encierra un conjunto de instrucciones FIELD, en caso de error en la entrada de alguno de ellos todos los campos del CHAIN se podrán modificar, mientras que los que no pertenezcan al CHAIN estarán bloqueados para la entrada de datos.

Ejemplos:

```
CHAIN.
      FIELD <campo1>,<campo2>, <campo3>.
      MODULE <mod1>.
      MODULE <mod2>.
  ENDCHAIN.

CHAIN
      FIELD <campo1>,<campo2>.
      MODULE <mod1>.
            FIELD <campo3> MODULE <mod2> ON CHAIN
INPUT
      ENDCHAIN.
```

4.3.3.2 Respondiendo a los códigos de función.

Cuando el usuario de una transacción, pulsa una tecla de función, un punto de menú, un pushbutton, un icono o simplemente la tecla ENTER, los datos introducidos en la pantalla se pasan a los módulos del PAI para ser procesados junto a un código de función que indicará qué **función** ha solicitado el usuario.

En el Screen Painter, será necesario crear un campo de tipo código de función, OK, (de longitud 5), que normalmente aparece al final de la lista de campos de cada pantalla. Tradicionalmente a este campo se le denomina **OK_CODE**, y será declarado en nuestro module Pool como cadena de caracteres de 4 posiciones:

```
DATA: OK_CODE(4).
```

En la lógica de proceso de cada pantalla, tendremos que realizar el tratamiento del OK_CODE. Para ello utilizaremos un módulo que deberá ser el último del evento PAI, es decir que se ejecutará una vez que todos los datos de entrada han sido validados correctamente.

Ejemplo:

LOGICA DE PROCESO

```
PROCESS BEFORE OUTPUT

      ….

PROCESS AFTER INPUT.


CHANGE_DATA.
      FIELD …
                MODULE …
DISPLAY_DOC.
                MODULE…

SCREEN 0200.
                MODULE OK_CODE.
```

MODULE POOL

```
MODULE OK_CODE.
   CASE OK_CODE.
      WHEN 'BACK'.
         SET SCREEN 0.
         LEAVE SCREEN.
      WHEN 'CHNG'.
         PERFORM

      WHEN 'DISP'.
               PERFORM

         WHEN 'COPY'.
            CALL

      ENDCASE.
      CLEAR OK_CODE.
      ENDMODULE.
```

Una vez procesado el código de función, borraremos el contenido del OK-CODE, inicializándolo para la próxima pantalla. Podemos guardar el contenido del OK-CODE en una variable intermedia e inicializarlo inmediatamente.

4.3.3.3 Procesando Step loops.

Un **Step loop** es un conjunto de datos idénticos que son copiados repetidamente, e interactúan con el usuario como una tabla.

Será necesario crear el Step loop con el Screen Painter, seleccionando los datos que queremos que formen parte del Step Loop y seleccionando la opción 'Loops' del menú 'edit'.

Dentro de los atributos de campos pertenecientes al step loop, podemos actualizar **'Ltype'** con el tipo del step loop (fijo o variable), donde el tipo variable significa que el tamaño del step loop se ajusta según el tamaño de la pantalla, mientras que fijo no ajusta el step Loop y **'Lcnt'** con el número de líneas de step loop.

Posteriormente podremos acceder al contenido de los campos del step loop con las instrucciones **LOOP... ENDLOOP o LOOP AT... ENDLOOP**.

Entre el LOOP y el ENDLOOI', es posible utilizar las instrucciones de lógica de proceso : FIELD, MODULE, VALUES, CHAIN, aunque lo más normal es utilizar MODULE para llamar a un módulo ABAP/4 que trate cada línea del step loop.

Será necesario codificar un LOOP en los eventos PBO y PAI por cada step loop de la pantalla, ya que permitirá la comunicación entre el programa y la pantalla.

Sintaxis :

- **LOOP. . . ENDLOOP.**

Permite moverse entre las líneas del step loop. En un evento PBO, permite copiar los campos del loop a la pantalla, mientras que en un PAI, los campos de pantalla son copiados en un área de trabajo del programa (como un registro de cabecera).

Utilizaremos esta instrucción si queremos utilizar nuestro propio método de scrolling.

En la variable del sistema **SY-STEPL** tendremos el índice de la línea que estamos procesando actualmente.

Ejemplo :

***** SCREEN PAINTER, LOGICA DE PROCESO *****

```
PROCESS BEFORE OUTPUT.
        LOOP.
                MODULE LEER_TABLA_INTERNA.
        ENDLOOP.

PROCESS AFTER INPUT.
        LOOP.
                MODULE MODIFICA_TABLA.
        ENDLOOP.
```

***** MODULE POOL*****

```
MODULE LEER_TABLA_INTERNA OUTPUT
        IND = BASE + SY-STEPL.-1
                READ TABLE INTTAB INDEX IND.
        ENDMODULE

MODULE MODIFICA_TABLA INPUT.
        IND=BASE + SY-STEPL-1
                MODIFY INTTAB INDEX IND.
        ENDMODULE
```

- **LOOP AT <tab interna>... ENDLOOP**.

Permite moverse entre las líneas del step loop, transfiriendo datos entre una tabla interna y el step loop. Con este loop, aparecen automáticamente las barras de scrolling.

En el PBO, podemos utilizar:

LOOP AT <tab -interna>

CURSOR <índice> (FROM <LI> TO <L2>). ...
ENDLOOP.

Con **CURSOR** indicamos cual es el primer registro de la tabla **interna** que queremos visualizar. Si indicamos una variable-,de programa, el sistema la irá actualizando conforme nos vayamos moviendo por el step loop.

Con **FROM y TO** podemos controlar que porción de la tabla interna visualizaremos. Por defecto será toda la tabla.

En el PAI, podemos utilizar la instrucción MODIFY (dentro de un módulo ABAP/4), para modificar la tabla interna con los valores del área de trabajo.

Ejemplo:

***** SCREEN PAINTER, LÓGICA DE PROCESO *****

```
PROCESS BEFORE OUTPUT.
        LOOP AT tab_int FROM desde TO hasta CURSOR indice.
        ENDLOOP.

PROCESS AF'FER INPUT.
        LOOP AT tab_int
                MODULE modif_tab_int.
        ENDLOOP.
```

***** MODULE POOL ABAP/4 *****

```
MODULE modif_tab_int
        MODIFY tab_int INDEX indice.
ENDMODULE
```

- **LOOP AT <tabla_BDD>... ENDLOOP**.

Permite moverse entre las líneas del step loop, transfiriendo datos entre una tabla de base de datos y el step loop. En el screen painter tendremos que definir los campos del step loop como campos de base de datos.

Si queremos actualizar la base de datos con contenido del step loop, tendremos que utilizar la sentencia : **MODIFY <tabla>.** de la lógica de proceso dentro de un LOOP AT.

Ejemplo:

```
PROCESS AFTER INPUT.
LOOP AT <tab_BDD>.
        MODIFY < tab_BDD>.
ENDLOOP.
```

4.3.4 El Flujo de la transacción.

Desde una transacción podemos **ir** controlando el flujo de pantallas de la misma, llamar a otras transacciones o a reports.

- **Controlando la secuencia de pantallas.**

Existen dos formas de saltar de pantalla en pantalla.

Por defecto, cuando acaben todos los módulos del evento PAI, el sistema saltará a la pantalla que indique el atributo **Next Screen** de la pantalla en ejecución**.** Es posible modificar el atributo de próxima pantalla con la instrucción **SET.**

SET SCREEN <no._pantalla>.

Existe una fórmula dinámica para llamar a otras pantallas, interrumpiendo así la secuencia de pantallas descrita mediante el atributo *Next Screen* Para ello utilizaremos la instrucción **CALL SCREEN.**

CALL SCREEN < no_pantalla >.

O

CALL SCREEN < no_pantalla >.
STARTING AT <columna_inicio> <linea_inicio>
ENDING AT <columna_fin> <linea_fin>

Con esta última versión del CALL SCREEN, podemos utilizar ventanas de tipo popup, indicándole las coordenadas de inicio y final de la ventana, siempre que sea más pequeña que una ventana normal.

Tendremos en cuenta que cuando finalice la pantalla o conjunto de pantallas, que hemos incorporado mediante CALL SCREEN, la secuencia normal de la transacción continuará ejecutándose desde el punto en que se dejó.

La manera más normal de teminar el proceso de una pantalla e ir directamente a ejecutar otra, es usando la instrucción **LEAVE.**

LEAVE TO SCREEN < no._pantalla>.

SET SCREEN <no._pantalla>.
LEAVE SCREEN.

Así, una manera de volver a la secuencia de pantallas anterior a la utilización de una instrucción CALL SCREEN puede ser:

LEAVE TO SCREEN 0. 0 **SET SCRFEN 0.**
LEAVE SCREEN.

En caso de ningún CALL SCREEN, estas instrucciones finalizarán la ejecución
de la transacción.

- **Llamando a un report** desde una transacción.

Existen dos posibilidades para incluir reports en una transacción. Dependiendo de la cantidad de información que la transacción tenga que suministrar al report utilizaremos una u otra posibilidad.

En el caso de que la transacción suministre la mayoría de los datos del report utilizaremos la técnica del **LEAVE TO I,IST-PROCESSING,** consistente en cambiar el tipo de ejecución de modo diálogo (transacciones) a modo lista (reports interactivos).

LEAVE TO LIST-PROCESSING.

El listado interactivo se ejecutará mientras que el usuario no indique un comando EXIT o BACK en la lista básica o no se encuentre con una instrucción para abandonar el listado interactivo, como por ejemplo **LEAVE SCREEN.** El sistema automáticamente volverá al PBO de la pantalla que estaba procesando.

```
Ejemplo:    CASE OK_CODE
                WHEN 'DISP'.
                    LEAVE TO LIST PROCESSING.
                    WRITE: /....
                    WRITE: /....
                    LEAVE SCREEN.
                .WHEN….
            ENDCASE.
```

Si entre el report y la transacción hay pocos datos en común o el proceso de reporting lo tenemos identificado en un programa independiente, utilizaremos la instrucción **SUBMIT.**

SUBMIT <nombre_report> (AND RETURN).

Con el SUBMIT, el report y la transacción no tienen el mismo área de trabajo (LUW), por tanto no comparten datos y el único intercambio de datos se producirá mediante los parámetros de entrada del report.

Podemos intercambiar datos con:

SUBMIT <nombre_report> WITH...

Ver la **Ayuda Online del editor de ABAP/4** para obtener más información sobre como pasar parámetros a un report desde una transacción.

- **Llamando a un módulo de función** desde una transacción.

Desde el modulo pool, también es posible ejecutar un módulo de función de la misma forma que lo hacemos para un report, es decir, utilizando la instrucción **CALL FUNCTION**.

Existen muchos módulos de función especialmente diseñados para transacciones.

Por ejemplo:.

```
CALL FUNCTION 'POPUP_TO _ CONFIRM_LOSS_OF_DATA'
                    EXPORTING
                         TEXTLINE1 = TEXT-001
                         TEXTLINE2 = TEXT-002
                  IMPORTING
                         ANSWER = REPLY.
```

Esta función te solicita confirmación antes de abandonar una transacción sin haber grabado previamente los datos introducidos.

- **Llamando a otra transacción** desde una transacción.

De la misma forma que existían dos métodos para cambiar las pantallas de una transacción, **existen** dos formas para llamar a otra transacción **independiente** de la transacción que se está ejecutando.

Si queremos que cuando **finalice** la transacción, el sistema ejecute otra utilizaremos :

LEAVE TO TRANSACTION 'cod_transacción'.

Si en cualquier momento queremos ejecutar otra transacción para posteriormente continuar la ejecución de la primera, utilizaremos :

CALL TRANSACTION'cod_transacción'.

En este caso el sistema creará una LUW independiente a la anterior y en el caso

de producirse un comando BACK, el sistema retornará para ejecutar la primera transacción.

Un caso típico es querer ejecutar una transacción pero evitando introducir los parámetros de entrada de esta manualmente (es decir, saltando la primera pantalla). En ese caso usaremos la instrucción :

CALL TRANSACTION'cod_trans'AND SKIP FIRST SCREEN.

Para lo cual será imprescindible utilizar parámetros almacenados en memoria, ya sea mediante los atributos **SET** (en la primera transacción) y **GET** (,en la transacción que llamamos) o codificando explícitamente las instrucciones SET-GE'I' respectivamente:

SET PARAMETER ID'param'FIEI,D <campo>.

GET PARAMETER ID'param'FIEI,D <campo>.

4.3.5 Actualizando la base de datos en una transacción.

Todas las operaciones que se realizan sobre la base de datos no se reflejan inmediatamente en base de datos hasta que no se produce un proceso de actualización (**COMMIT**). En el ámbito de las bases de datos. Se dice que el conjunto de todas las acciones referentes a base de datos incluidas entre dos commits se le denomina LUW (Logical Unit Work)

Existen dos clases de actualizaciones **Commits internos** (que se producen de forma transparente al programador de la transacción) y **Commits SAP** (actualizaciones forzadas por el programador en el momento que lo crea necesario, por ejemplo en la última pantalla de una transacción si no se ha producido ningún error)

Cuando se realizan actualizaciones de la base de datos en una transacción, se deberá escoger entre una **actualización síncrona** y una **actualización asíncrona.**

Con una actualización síncrona, la actualización se produce en el mismo momento que el usuario lo solicita.

En una actualización asíncrona, el sistema graba en una cola las operaciones que se deban realizar, y en un segundo paso, este conjunto de operaciones se procesará para ir realizando las actualizaciones. En este caso, los tiempos de respuesta serán inmediatos, ya que la tarea de actualización, que es la más costosa en tiempo, se pospondrá para otro momento.

Las instrucciones de actualización de la base de datos son:

- **COMMIT WORK.**

  Realiza un update físico de la LUW sobre la base de datos.

- **PERFORM <rutina> ON COMMIT.**

  Permite ejecutar la <rutina> cuando el sistema encuentra un COMMIT WORK. Esto nos permite concentrar todas las actualizaciones de la base de datos en un único procedimiento, codificando todas las instrucciones de base de datos en la rutina.

- **ROLLBACK WORK.**

  Deshace todas las operaciones realizadas sobre la base de datos hasta el último COMMIT WORK. lJtlll7,aremos el ROI,LBACK WORK cuando ocurra algún error o cuando el usuario abandona una transacción con un BACK, CANCEL o END y ya hemos realizado alguna operación en base de datos que debamos deshacer.

- **CALL FUNCTION <módulo_función> IN UPDATE TASK**

  Permite ejecutar un módulo de función en el momento que se quiera actualizar físicamente en la B.D.D. El módulo de función deberá estar marcado como módulo de update.

Podemos distinguir entre diversos métodos de actualización:

1.- El más sencillo es codificar directamente las instrucciones de base de datos: INSERT, DELETE, MODIFY,... en el programa principal de la transacción y finalmente realizar un COMMIT WORK o esperar a que finalice una pantalla para que el sistema realice un commit interno (aunque esto último no es muy fiable)

Será únicamente viable en transacciones de una única pantalla.

2.- Utilizar **PERFORM <rut> ON COMMIT** y utilizar la rutina <rut> para
codificar las instrucciones de base de datos (INSERT, UPDATE, MODIFY, DELETE...)

3.- Utilizando **CALL FUNCTION <mod_función> IN UPDATE TASK** los cambios sobre la base de datos se realizan en una fase asíncrona (update task), en el módulo de función indicado cuando se produce un COMMIT WORK.

Los datos que se actualizan son los valores en el momento de hacer la llamada al módulo de función y no los valores de cuando se ejecute.

4.- Utilizar una función en UPDATE TASK que se llama desde una rutina que a su vez es llamada 'ON COMMIT'. (método asíncrono). En este caso los valores de actualización son los que existan previamente a la actualización física.

Ver el Capítulo **'Writing Dialog programs in ABAP/4 : Database Interface'** del manual **'BC ABAP/4 User handbook',** para obtener más información sobre como realizar los procesos de actualización en las transacciones SAP.

4.3.6 **El bloqueo de datos en SAP**.

Para coordinar el acceso de distintos usuarios a los mismos datos, y para evitar posibles inconsistencias en las actualizaciones, SAP dispone de un método interno de bloqueo, de forma que únicamente un usuario podrá procesar un objeto de datos de la primera pantalla de una transacción a la última.

Para implementar los bloqueos es necesario utilizar objetos de bloqueo, que pueden estar compuestos de un registro de una cierta tabla o de una colección de registros de una o varias tablas.

Los objetos de bloqueo se definen y se activan desde el diccionario de datos. En el momento que se activan, el sistema creará dos módulos de función que permiten bloquear o desbloquear los objetos que se han especificado.

ENQUEUE_<objeto_de_bloqueo>.

DEQUEUE_-<objeto-de-bloqueo>.

En el desarrollo de transacciones, para **bloquear** un objeto, llamaremos a la función **ENQUEUE-_<objeto-de-bloqueo>** en el evento PAI de la primera pantalla.

Es posible comprobar el resultado del bloqueo comprobando si están activas las excepciones de este módulo de función.

FOREIGN_LOCK : Si el objeto está bloqueado por otro usuario.
SYSTEM_FAILURE : Error del sistema.

Mientras que para **desbloquear** un objeto bastará con utilizar la función **DEQUEUE_<objeto_de_bloqueo>.**

4.3.7 Ayudas programadas. Eventos POH y POV.

Es posible programar nuestras propias ayudas para la entrada de datos en los campos**.** Para ello utilizaremos los eventos **PROCESS ON HELP-REQUEST (POH)** para las ayudas con **F1** y **PROCESS ON VALUE-REQUEST (POV)** para las ayudas de **F4** entradas posibles.

Si el usuario pulsa Fl o F4, el sistema **no** activa el evento PROCESS AFTER INPUT, pero si el correspondiente evento de ayuda.

- **PROCESS ON HELP-REQUEST(POH).**

Indicamos en una instrucción FIELD, el elemento de datos del diccionario, del cual queremos obtener la ayuda sobre un campo de pantalla.

FIELD <campo> WITH<elem_datos>".

FIELD <campo> WITH<var>. Donde <var> es una variable que contiene un elemento de datos.

- **PROCESS ON VALUE-REQUEST (POV).**

Indicamos en una instrucción FIELD, el módulo donde vamos a tratar los valores posibles, que sustituye a la ayuda de SAP, para un campo de pantalla.

FIELD <campo> MODULE<módulo>

5 Creación de nuevas tablas en el Diccionario de Datos

5.1- El proceso de creación de una tabla.

El proceso de creación de una tabla de base de datos en el diccionario de datos consta de una serie de pasos :

Crear el objeto TABLA
Especificar los campos de la tabla.
Definir los elementos de datos y dominios de la tabla.
Especificar los índices, claves externas y parámetros técnicos.
Activar la tabla.

- Entrar en al **ABAP/4 Dictionary,** indicar el nombre de la tabla que vamos a crear y seleccionar **'Create'.** (Empezará por Z).

- Introducir una descripción breve de la tabla.

- Introducir un **'delivery class',** que indicará el grado de responsabilidad que tiene SAP o el cliente sobre la tabla. Normalmente utilizaremos el tipo A , que corresponde a la tablas de aplicación (datos maestros o de transacción, es decir que se generan vía transacciones SAP). Podemos ver la utilidad del resto de posibilidades, (C,G,L,E,S,W) pulsando Fl sobre este campo.

- Entrar el **nombre de los campos** de la tabla en la columna **'Field name'.**

- Seleccionar los campos que queremos que formen parte de la **clave primaria** de la tabla.

- Introducir el **elemento** de datos de cada campo :

⇒ **Si** este ya existe en el Diccionario de datos, la información del mismo, aparecerá automáticamente.

⇒ Si el elemento de datos no existía en el diccionario de datos, con doble-click podemos ir al mantenimiento de elementos de datos para crearlo en ese momento.

- Seleccionar el flag `**initial**´ , si los campos deben ser inicializados con sus valores iniciales.

- Si queremos permitir el mantener la tabla desde la transacción de mantenimiento de tablas (SM31), lo indicaremos activando el flag **'table maintenance allowed'.**

- Si queremos que la tabla que estamos creando dependa del mandante, incluiremos el campo **MANDT** como el primer campo de la tabla.

- Es posible especificar parámetros de tipo técnico como el tipo de datos que almacena la tabla, el tamaño aproximado, ...etc. desde : **Goto -> technical Settings.**

- Finalmente, grabaremos los datos introducidos.

5.2. Las claves foráneas.

Si un campo de la tabla tiene un dominio cuyo rango de valores está definido por una tabla de valores, será necesario definir las claves foráneas (externas) con dicha tabla de valores e indicar con un * en la columna **'Check table'.**

Para ello :

- Seleccionamos el campo con tabla de chequeo.
- Seleccionamos **Goto -> Foreign Key**.
- Introducir
    - Texto de la clave foránea.
    - Código de la tabla de chequeo.
    - Asignación del campo (Genérico o Constante).
    - Chequeo del campo en las pantallas.
    - Mensaje de error que sustituye al mensaje estándar de error cuando se introduce un valor que no existe en la tabla de chequeo.
    - Cardinalidad. Describe la relación con respecto al número de registros dependientes entre las tablas -(1:C, 1:CN, 1:N, 1: 1)-
    - Tipo de campo de clave foránea.

Será necesario definir las claves externas si queremos que el chequeo automático de las pantallas funcione correctamente.

5.3 Otras posibilidades en la creación de tablas.

- Si hemos marcado la posibilidad de poder mantener el contenido de la tabla desde la transacción SM31, será necesario que ejecutemos un proceso de generación de dicho mantenimiento.

Para hacerlo debemos seleccionar .- **Environment -> Gen. Maint. Dialog.**.

Introduciremos:

El grupo de funciones para el mantenimiento de tablas.
El grupo de autorizaciones para el mantenimiento de tablas.
El tipo de mantenimiento que queremos:
De una pantalla.
De dos pantallas. Una pantalla con lista de registros y otra con valores individuales.
El número de Dynpro(s) que queremos asignar a la(s) pantalla(s).

- El índice primario se crea cuando una tabla es activada. Bajo ciertas circunstancias es posible que sea recomendable ayudar a la selección de datos con **índices secundarios.**

Para definir índices secundarios ir a: **Goto -> Indices.**

- Finalmente, antes de poder utilizar una tabla creada por nosotros, será necesario realizar un proceso de **ACTIVACION.**