ÍNDICE

1 INTRODUCCIÓN.

1.1 ¿Qué es ABAP/4 ? (Advanced Business Application Programming 4th Generation).

El ABAP/4 es un lenguaje de programación de 4a. Generación (4GL) orientado tal como su definición especifica, al desarrollo de aplicaciones de negocios. Todos los módulos disponibles en SAP han sido programados en este lenguaje de programación. Además podemos escribir nuevas aplicaciones en ABAP/4 como complemento a las ya existentes o como apoyo a la configuración del sistema.

1.2 Propósito de R/3

El propósito de un sistema R/3 es proveer una suite firmemente integrada, para aplicaciones de negocios a gran escala. El conjunto estándar de aplicaciones utilizadas en el sistema R/3 son las siguientes:

Modulos de SAP

- PP (Production Planning)
- MM (Materials Management)
- SD (Sales and Distribution)
- FI (Financial Accounting)
- CO (Controlling)
- AM (Fixed Assets Management)
- PS (Project System)
- WF (Workflow)
- IS (Industry Solutions)
- HR (Human Resources)
- PM (Plant Maintenance)
- QM (Quality Management)

Estas aplicaciones son llamadas *áreas funcionales*, *áreas de aplicación* o *módulos funcionales* de R/3. Todos estos términos son sinónimos unos de otros.

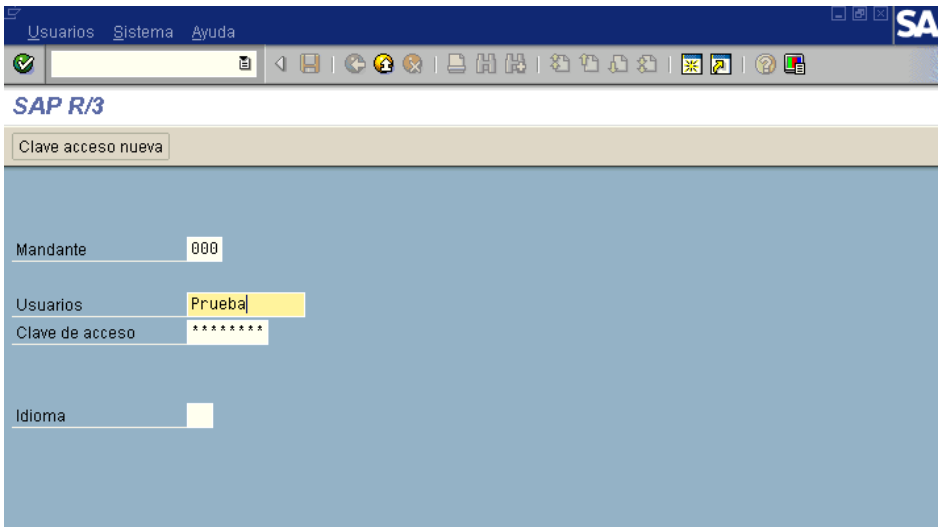
Tradicionalmente, los negocios ensamblan en una suite de aplicaciones de datos por la evaluación individual de los productos y por la compra separada de estos productos. Entonces las interfaces se necesitan entre ellas. Por ejemplo, MM necesitará ligarse a SD y FI, y WF necesitará alimentarse de HR.

R/3 viene con las aplicaciones de negocio, bases necesarias por todas las grandes empresas. Estas aplicaciones existen en un ambiente homogéneo. Y son diseñadas para su funcionamiento usando una sencilla base de datos y un (gran) conjunto de tablas. La producción actual del tamaño de una base de datos está entre el rango de 12 gigabytes a 3 terabytes. Alrededor de 8,000 tablas son enviadas con la entrega estándar del producto R/3 (el número exacto depende de la versión de R/3 y de los módulos que se instalen).

1.3 Conociendo el ambiente de R/3.

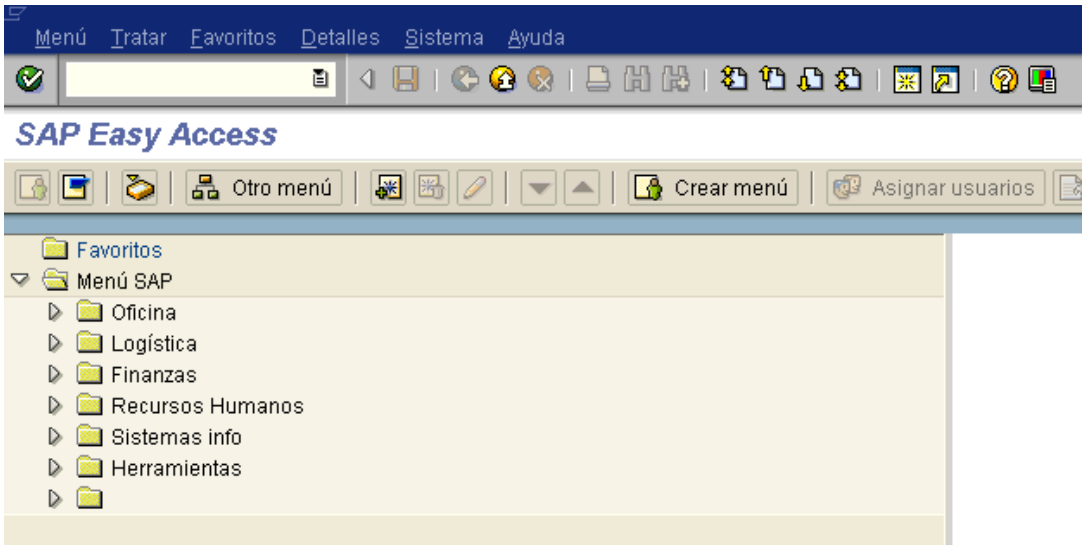
En un ambiente Windows, se puede entrar a R/3 escogiéndolo desde el menú principal (Start) o por medio de un doble click en el icono de R/3, el cual se muestra a continuación.

El sistema R/3 requiere un mandante (cliente), un usuario, un password y el lenguaje. La pantalla logon aparece en la siguiente figura. Cuando sean introducidos estos campos solo bastará presionar ENTER y se podrá accesar al sistema.



Pantalla de Acceso

El sistema R/3 desplegará una pantalla de derechos reservados, y cuando presione ENTER el sistema desplegará el menú principal como se muestra en la figura siguiente.



Menú principal

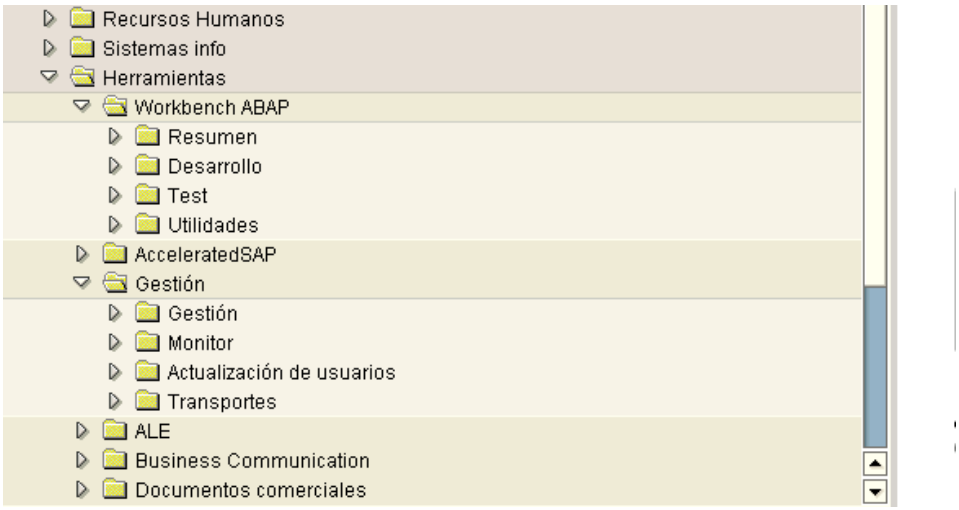
Desde el menú principal, se puede acceder a tres áreas conceptuales en el sistema R/3:

- El área de Aplicaciones.
- El área de Basis
- El área de Desarrollo Workbench

En el **área de Aplicaciones**, se inician transacciones para las áreas funcionales dentro de R/3. Para acceder a éste, desde el menú principal de R/3 se escoge una de los siguientes: Oficina, Logística, Finanzas, Recursos Humanos o Sistemas Info.

En el **área de Herramientas**, se puede ejecutar una transacción que monitorea el sistema R/3. Al acceder al área de Basis, desde el menú principal, se escoge la opción Herramientas → Gestión. Aquí se encontrarán algunas funciones y herramientas para la administración de base de datos.

El **área de desarrollo Workbench** es usada para crear y probar programas en ABAP/4. Para acceder al desarrollo Workbench, se escoge del menú la opción Herramientas →Worbench ABAP.



1.4 Interfaz del usuario.

Cada pantalla de R/3 contiene estos elementos:

- Barra de título: contiene el título de la pantalla actual.
- Barra de menú: El contenido de la barra de menú cambia en cada pantalla. Pero las opciones **System** y **Help** se presentan en todas las pantallas.
- Campo de comando: En este lugar se introduce un comando para ser ejecutado. Por ejemplo, se puede realizar un *log off* si se introduce: */nex* en este campo y se presiona al tecla ENTER.
- Barra de Herramientas Estándar: Contiene el campo de comando y una serie de botones. Estos pueden cambiar en apariencia, posición, o función y se encontrarán en cada pantalla. Algunas veces pueden presentarse deshabilitados.
- Barra de Herramientas Aplicaciones: Cambia en cada pantalla. Despliega los botones que dan un rápido acceso a opciones del menú para esa pantalla.
- Menú Interfaz: Habilita las opciones de cambiar las características de la interfaz para el usuario, accede al clipoard de Windows, y genera gráficos.
- Área de la pantalla: Ésta es un área en al mitad de la pantalla que despliega el reporte de datos o una pantalla desde un programa de dialogo.
- Barra de estado: Despliega mensajes, número de sesión, número del cliente, indicador del modo insert/overtype, y la hora actual.

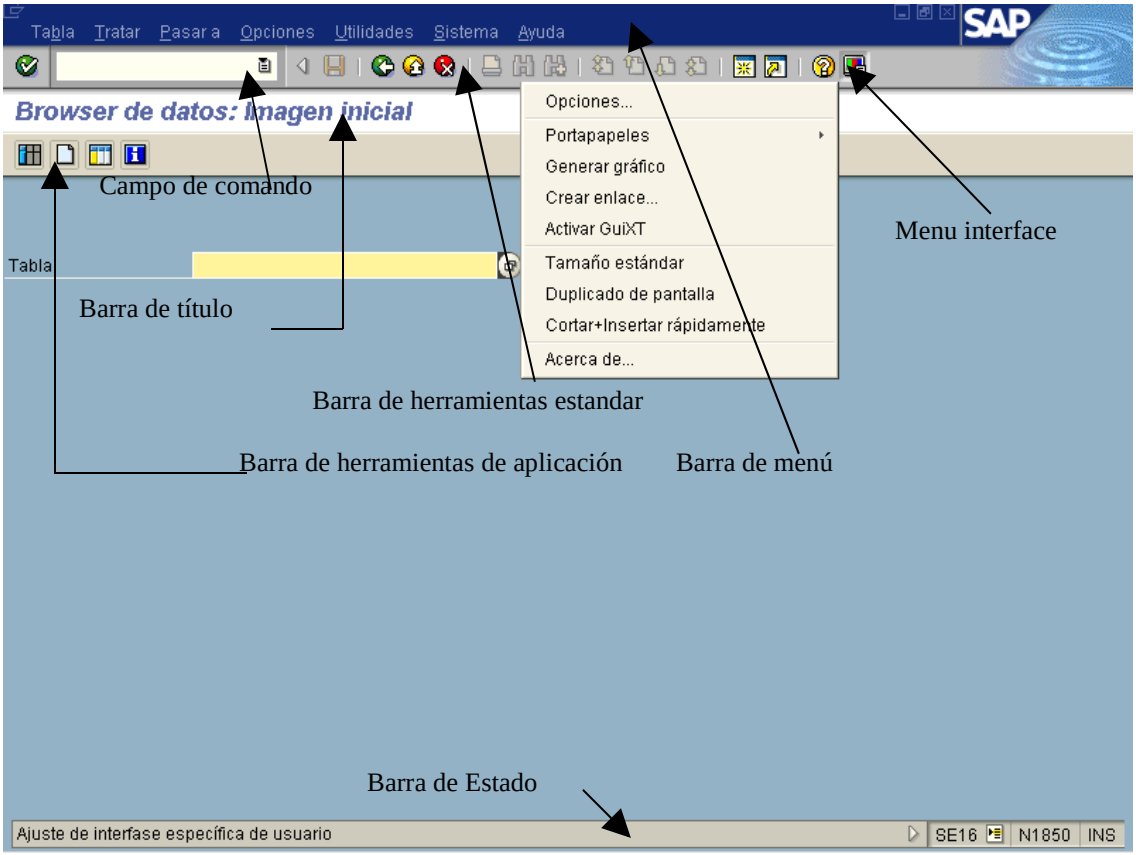
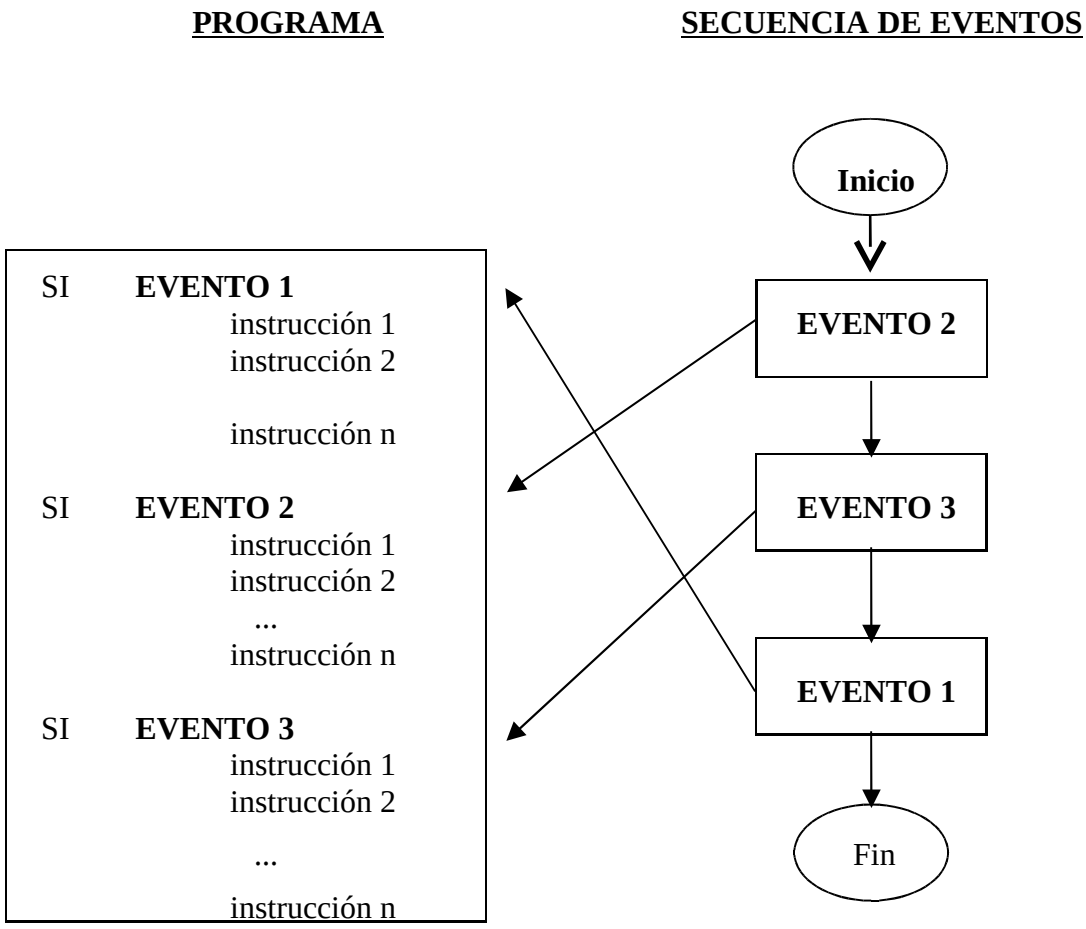


Tabla Tratar Pasar a Opciones Utilidades Sistema Ayuda
SAP
Browser de datos: Imagen inicial
Opciones...
Portapapeles
Generar gráfico
Crear enlace...
Activar GuiXT
Tamaño estándar
Duplicado de pantalla
Cortar+Insertar rápidamente
Acerca de...
Tabla
Campo de comando
Barra de título
Menu interface
Barra de herramientas estandar
Barra de herramientas de aplicación
Barra de menú
Barra de Estado
Ajuste de interfase específica de usuario
SE16
N1850
INS

2 CARACTERÍSTICAS DE ABAP/4.

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



Estos son algunos **conceptos y definiciones** que se deben conocer para comprender mejor la filosofía de trabajo en desarrollos Abap/4.

1. Configuración de SAP: SAP se configura en tres capas:

- Desarrollo: Es donde se realizan todos los desarrollos de programación y parametrización.
- Test: Donde se realizan las pruebas del sistema. En esta capa no se puede desarrollar.
- Producción: Donde el cliente está trabajando con datos reales. En esta capa no se puede desarrollar.

Esto es la estricta teoría. Muchas veces la capa de desarrollo y test se integran en una sola. Estas capas pueden estar configuradas las tres en una sola máquina o en máquinas distintas.

Los desarrollos en ABAP sólo se pueden hacer en la capa o sistema de desarrollo. Cuando se quieren pasar a Test o a Producción el administrador del sistema debe hacer una operación que se denomina transporte, donde pasa el programa de una capa a otra.

Un programa en ABAP/4 sólo se puede modificar en el mismo sistema o capa en el que se haya creado. Es decir no se puede modificar un programa en producción. Obligatoriamente se deben hacer los cambios o modificaciones en desarrollo y pasarlos a test y producción una vez que se haya probado, con una orden de transporte.

2. Perfiles de usuario.

Desde la parte de administración del sistema (BASIS) se pueden definir perfiles de usuario donde se le dan o deniegan a un usuario o grupo de usuarios los privilegios para acceder a ciertas aplicaciones o utilidades que SAP permite. Una de estas características es lo que se conoce como clave de desarrollador, que permite que un usuario pueda hacer nuevos desarrollos o modificar los existentes en el sistema.

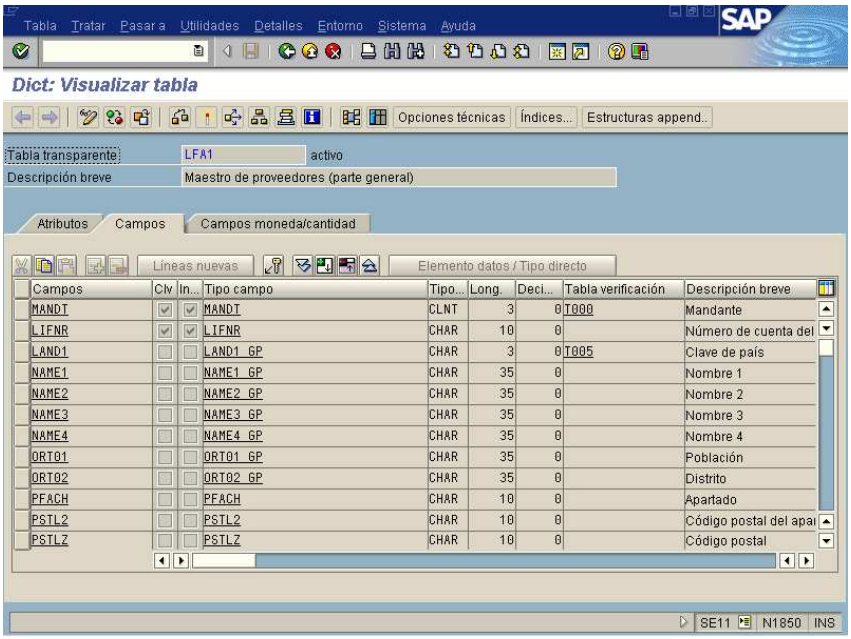
3. Concepto Cliente – Servidor.

SAP se configura como una arquitectura cliente-servidor. El sistema con las bases de datos está instalado en el Servidor. Los usuarios (o clientes) se conectan al servidor a través de un software instalado en cada máquina. El fichero que hay que ejecutar para ello es el sapgui.exe. Esto instala una aplicación en la máquina cliente conocido como SAPGUI, que una vez configurado adecuadamente nos permite acceder al servidor.

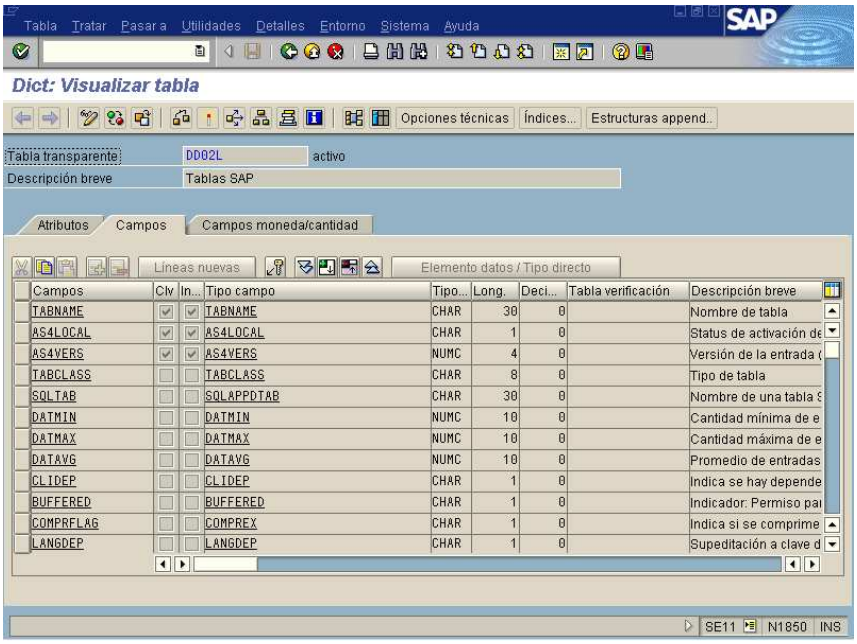
4. Concepto de Mandante.

En SAP hay objetos dependientes de Mandante u otros que son independientes de Mandante. Una tabla es dependiente del Mandante cuando su primer campo es uno llamado mandt de tipo CLNT. Este campo siempre forma parte de la clave de la tabla. Una tabla será independiente de Mandante cuando no tiene dicho campo.

Las siguientes tablas son ejemplos de: a) una tabla dependiente de mandante y b) una tabla independiente de mandante.



a) Tabla dependiente de Mandante.



b) Tabla independiente de Mandante.

El mecanismo de Mandante sirve para dividir los datos en diferentes grupos (sistemas, empresas, etc.) usando las mismas tablas y de forma transparente a la hora de programar. El Mandante por defecto es el que seleccionamos al hacer logon en el sistema.

2.1 Clase de desarrollo.

Es un sistema de organización de todos los objetos nuevos que se crean en SAP. Todos ellos deben ir en una clase de desarrollo. En un símil, un objeto nuevo sería un archivo y la clase de desarrollo sería la carpeta donde metemos ese archivo. Así se pueden tener los objetos divididos por módulos, utilidad, usuarios, etc.

Existe una clase de desarrollo llamada $TMP. Es la clase de desarrollo temporal, que se usa para pruebas. Los objetos que se creen en esta clase de desarrollo no podrán ser transportados al sistema de producción. Es la clase que se usa en formación.

Nota: Los cinco sistemas de ayudas para aprender y desarrollar en el lenguaje de programación Abap/4 son:

1. Desde el editor de programas, pulsando F1 sobre un comando de SAP. Nos dará una ayuda bastante buena sobre dicho comando con ejemplos.
2. Usando la ayuda extendida de SAP. En todas las pantallas de SAP tenemos la opción Help desde la que podemos buscar cosas concretas de Abap y de todo el sistema en general.
3. Docuprint. Son una serie de ficheros de ayuda con ejemplos, agrupados por temas en los que viene una explicación más amplia que la ayuda extendida de cosas relacionadas con Abap y el resto del sistema.
4. Internet. Existen muchas páginas en Internet donde se pueden encontrar libros, preguntas comunes, foros de discusión, etc. muy útiles para resolver dudas acerca de Abap/4.

5. Tenemos una serie de transacciones con programas ejemplo IBLS, etc.

3 APLICACIONES DEL ABAP/4.

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

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

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

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

El reporting interactivo tiene las siguientes características: orientado a pantalla, listados cortos y con datos resumidos. Información detallada en sublistados o ventanas y controlado por teclas de función.

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

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

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

EL Menú painter o CUA (Common User Access) permite organizar los elementos de la superficie gráfica, sin necesidad de conocer los softwares de presentación (WINDOWS ...). Se especificará el contenido de la barra de menús, teclas de función y menús de acción.

Otras aplicaciones posibles del lenguaje de programación son la generación de interfaces (programas de **comunicaciones)**. Estos interfaces pueden conectar un sistema SAP con otro sistema cualquiera (también puede ser SAP). Un interface es una utilidad para transferir información de forma segura y automatizada. Hay varios tipos de interfaces: Batch input, Guis, Idocs, RFCs y Bapis.

Un batch input es simular mediante un proceso Batch la introducción de datos en el sistema vía transacción online.

Una Bapi (Bussines Application Programming Interface) es una función estándar de SAP que ya hace el interface, con lo que el programador tan sólo tiene que hacer una llamada a esa función.

Al entrar un Idoc al sistema, dispara una función que hace la entrada de datos automáticamente.

Una RFC (Remote Function Call) es una llamada en un sistema (servidor) desde otro sistema (cliente), es decir, desde SAP llamamos a un trozo de código en el sistema que queremos conectar con SAP que ejecuta, dentro de SAP, la entrada de datos.

4 ENTORNO DE DESARROLLO EN ABAP/4.

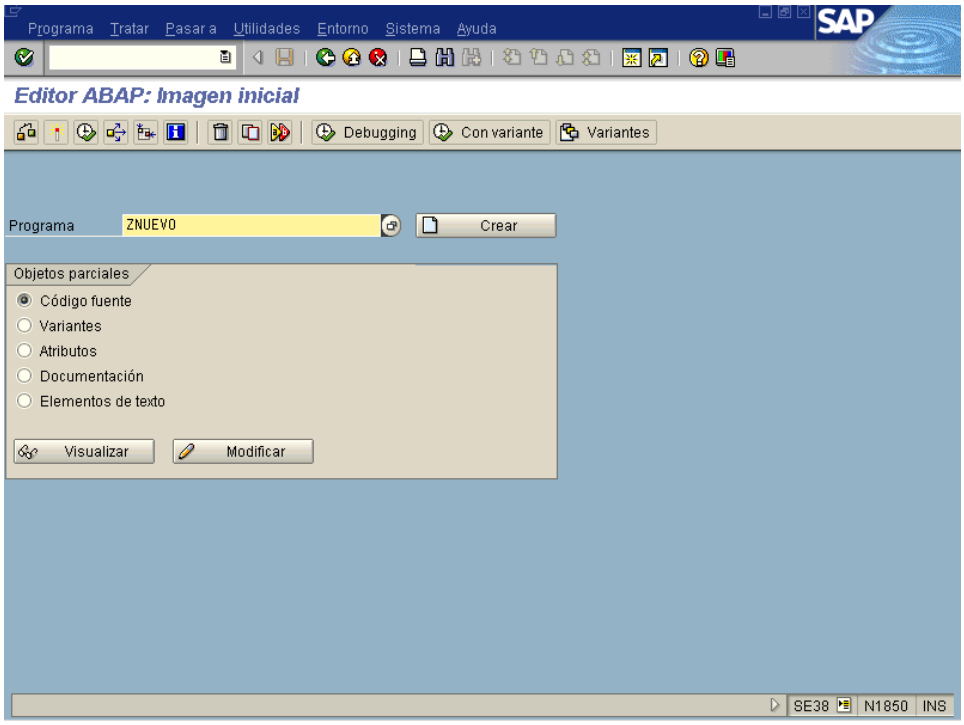
4.1 Creando un programa ABAP/4.

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

Ejemplo práctico: Creando Z_NUEVO

Herramientas →Workbench ABAP → Desarrollo → Editor ABAP. (SE38)

Introducir nombre programa, siempre deberá empezar con **Z**. Presionar el botón **Crear**.



Pantalla Inicial del Editor

- Introducir el título del programa. Este título aparecerá en la parte superior del listado.
- Tipo de programa (Obligatorio). Generalmente un 1 (REPORT).
- Status del programa (opcional).
- Módulo funcional al que hacemos referencia en el programa. Con un * especificamos que puede hacer referencia a cualquier aplicación.
- Clase del programa (opcional).
- Grupo de Autorizaciones con las que se puede ejecutar o editar y modificar un programa. (opcional).
- Pulsar ENTER
- Pantalla de selección (opcional, sólo si usamos Bases de Datos lógicas).
- **GRABAR.**

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

4.2 Trabajando con el editor de programas. (Transacción SE38)

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

También existen múltiples comandos de línea.

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

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

Una vez escrito el programa podemos **verificar** que sintácticamente no tenga ningún error y antes de poderlo ejecutar tendremos que **generar** para evitar que los cambios no sean tomados en cuenta y se llegue a ejecutar una versión anterior. En el proceso de generación SAP transfiere el estado del programa (Time Stamp) a diversas tablas del Diccionario de datos. La tabla TRDIR contiene información sobre todos los programas del sistema.

Algunos ejemplos de los comandos existentes en el sistema son:

Check.- Verifica sintácticamente que el programa creado no tenga ningún error.

Generate.- Crea una nueva versión del programa si es que ya existía alguna anteriormente para ejecutar el programa.

Save.- Salva el programa en el Servidor (NO en el Disco Duro de su máquina)

Execute.- Ejecuta el programa.

Pretty Printer.- Da formato al programa.

Cut.- Corta la(s) línea(s) señalada al buffer.

Copy.- Copia la(s) línea(s) al buffer.

Insert.- Inserta el contenido del buffer al código del programa.

Search/Replace.- Busca la(s) palabras indicadas dentro del código programa y/o las remplaza por otra(s).

Find next.- Busca la palabra indicada que se encuentra en la siguiente posición del cursor.

Undo.- Deshace la última acción dentro del código del programa.

Breakpoints.- Coloca puntos de ruptura cuando se está ejecutando el programa para poder usar el debug y conocer el contenido de algunas variables o tablas. Es útil para hacer debug a programas en sistemas en los que no se puede modificar el código. Hay que tener cuidado de quitarlos luego.

Upload.- Translada un archivo de un disco flexible o del cualquier otro dispositivo hacia el Servidor.

Download.- Baja un archivo al disco duro del Servidor o a cualquier otro dispositivo.

5 FUNDAMENTOS DE LA PROGRAMACIÓN DE REPORTS.

5.1 Tipos de instrucciones.

Un report (programa) consiste en una serie de instrucciones ABAP que empieza por una **palabra clave** y termina cada instrucción 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. (El texto quedará rojo)
2. Con comillas (“) en mitad de una línea.

Podemos combinar sentencias consecutivas de mismo formato. Por ejemplo:

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

es equivalente a :

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

5.2. Estructura de un programa.

REPORT <nombre> → Nombre programa

TABLES: → Definición de las tablas que se utilizan, tanto internas como del diccionario de datos.

DATA: → Definición de variables internas

SELECCIONES → Definición de la pantalla de selección.

EVENTOS → Es una etiqueta que identifica una sección del código. La sección de código asociado con un evento empieza con una nombre de evento y termina cuando el próximo nombre de evento es encontrado.

CÓDIGO → Ejecución de instrucciones en general dentro del programa.

SUBRUTINAS → Son una sección de código que puede ser usadas varias veces. Son como un mini-programa que pueden ser llamadas desde otro punto del programa. En ellas se pueden definir variables, ejecutar sentencias, obtener resultado y escribir las salidas.

5.3 Tipos de Eventos.

Existen diferentes eventos en ABAP/4, los principales son:

- Intialization
<sentencias>
- At selection-screen
<sentencias>
- Start-of-selection
<sentencias>
- End-of-selection
<sentencias>

- At line-selection
<sentencias>
- At user-command
<sentencias>

- Top-of page
<sentencias>
- End-of-page
<setencias>

•**INITIALIZATION.-** Se ejecutará antes que la pantalla de selección sea desplegada.

•**START-OF-SELECTION.-** Es la primera sentencia ejecutada en el programa, si no es precedido por otro evento, un rutina del sistema automáticamente inserta **star-of-selection** antes de la primera línea de código ejecutable.

•**END-OF-SELECTION**.- Es el último evento que se ejecutará en el programa.

Ejemplo:

Código

```
report  z_ejem1.
data  f1  type  i  value  1.

end-of-selection.
  write:/  '3. F1 =',    f1.

start-of-selection.
  write:/  '2.  F1 =',   f1.
  f1  =  99.

initialization.
    write:/  '1. F1=',    f1
    add 1 to f1.
```

Salida:

```
1.  F1 =   1
2.  F1 =   2
3.  F1 =  99
```

•**AT SELECTION-SCREEN**.- En realidad, es un conjunto de eventos que permiten formatear a nuestro gusto la pantalla, las opciones se verán en el capítulo 8.

•**AT LINE SELECTION**.- Sirve para controlar la entrada del usuario en un Report Interactivo, cuando se hace un doble-click en un listado.

•**AT USER-COMMAND**.- Sirve para controlar la entrada del usuario en un Report Interactivo, cuando se pulsa un botón.

•**TOP-OF-PAGE**.- Ejecuta las instrucciones al inicio de la página (Encabezados).

•**END-OF-PAGE**.- Ejecuta las instrucciones al final de la página (Pie de página).

Ejemplo:

Código

```
report z_ejem2.
parameters p1(8).

write:/  'P1 =',  p1.

initialization.
   p1 = 'INIT'.

end-of-selection.
```

```
    write:/  (14)   sy-uline, /  'Fin del programa'.

top-of-page.
    write:/  'Este es mi Título'.
    Skip.
```

Salida

```
Este es mi Título
P1 = INIT.
---------------------
Fin del programa
```

La secuencia de eventos no es relevante. Cada evento se disparará en su momento, independientemente del evento que haya antes o después en el programa.

6 DECLARANDO Y PROCESANDO DATOS

6.1. Tipos de Datos.

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

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

6.2 Tipos de Variables.

Podemos decir que existen 2 tipos de variables; las que ya están definidas por R/3 y las que el programador define para realizar sus procedimientos dentro del programa. Pero esto no quiere decir que los primeros no pueden ser accedidos por el usuario.

6.2.1 Variables del Sistema.

Estas variables han sido definidas en el sistema y se encuentran en la estructura SYST, son flags que nos indican, dependiendo de la operación que se realice, un estado de dicha operación, es decir, nos pueden

indicar si se realizó la operación correctamente, la fecha del sistema, la hora del sistema, etc. Como se puede ver en el anexo 3 existe un gran número de estas variables, aunque las más usuales son:

SY-SUBRC -> Código de retorno posterior a determinadas operaciones.
SY-INDEX -> Cantidad de repeticiones de bucles.
SY-DATUM -> Fecha del día.
SY-PAGNO -> Número de página.
SY-ULINE -> CONSTANTE: Dibuja una línea horizontal.

Todas estas variables se leen e imprimen de la misma forma que las variables que son definidas por el programador, la única diferencia consiste en que este tipo de variables no pueden ser modificadas por el programador, únicamente cambia su valor dependiendo de las operaciones efectuadas dentro del programa.

6.2.2. Variables de programa.

Se utiliza la forma básica de declaraciones de datos para definir variables en un programa, con la siguiente sintaxis:

DATA v1 [(longitud)] [**TYPE** t] [**DECIMALS** d] [**VALUE** 'xxx'].

o

DATA v1 LIKE v2 [**VALUE** 'xxx'].

Donde: **v1** es el nombre de la variable.

v2 es el nombre de la variable previamente definida en el programa o es el nombre de un campo que pertenece a una tabla o a una estructura en el Diccionario de Datos.
(longitud) es la longitud.
t es el tipo de dato.
d es el número de dígitos después del punto decimal (máximo 14, *únicamente usado con un dato de tipo p*).
'xxx' valor inicial de la variable.

Ejemplos:

```
data  f1            type   c.
data  variable2     like    f1.
data  max_valor     type   i   value   100.
data  fecha         type   d   value   '19980211'.
data  flag          type    c   value is initial*.
```

*__IS INITIAL__ indica que la variable *flag* tiene el valor inicial de una variable de tipo **c.**

Si no se indica el tipo de la variable, será tipo carácter (Texto) y si no se indica longitud, la que tenga el tipo por defecto (ver tabla en página anterior).

Ejemplo: DATA VAR_CAR. . → Creará una variable texto de longitud 1.
DATA VAR_CAR(8). → Creará una variable texto de longitud 8.

Los nombres de las variables pueden ser de longitud 1 a 20 caracteres. Pueden contener cualquier carácter excepto **( ) + . , :**

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 (con las mismas propiedades).

Ejemplo: DATA ACREEDOR **LIKE** LFA1-LIFNR.

o

DATA LIFNR **LIKE** LFA1-LIFNR.

A pesar de que el nombre de las variables puede ser el que el programador desee, es muy recomendable que cuando se utilice el comando **LIKE,** el nombre de las variables sea el mismo que el del campo de la tabla del diccionario de datos al que se hace referencia.

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

Ejemplo: DATA CONTADOR TYPE P VALUE 1.

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

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

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

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

También usaremos la instrucción DATA para declarar tablas internas.

La diferencia entre una **tabla interna** y una **tabla del diccionario de datos** es que las primeras guardan los datos en memoria y no en la base de datos, mientras que la segunda guarda la información en la base de datos.

Una **estructura** es definida en le diccionario de ABAP/4 como las tablas y pueden ser direccionadas desde los programas de ABAP/4. Cualquier campo en la definición de la estructura en el diccionario de ABAP/4 es automáticamente reflejada en todos los programas. Mientras que los datos en las tablas de datos se almacenan de manera permanente, las estructuras únicamente contiene datos durante el tiempo de ejecución del programa. La única diferencia es que no se genera ninguna tabla en la base de datos.

Ejemplo:

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

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

Las tablas internas se declaran, inicializan y referencian como un registro de datos.

También podemos utilizar la misma estructura que una tabla de base de datos, para ello utilizaremos la instrucción **INCLUDE STRUCTURE**.

Ejemplo:

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

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

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

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

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

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

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

Para inicializar las variables utilizamos la sentencia:

CLEAR <campo>.

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

La asignación e inicialización de los registros de datos funciona de la misma forma que en las variable normales. Un CLEAR inicializa todos los campos del registro. Podremos conseguir una asignación mas potente con **MOVE-CORRESPONDING**.

MOVE-CORRESPONDING <reg1> TO <reg2>.

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

6.3. Declaración de Constantes.

Para declarar una constante es igual que definir una variable, únicamente que la sentencia para definirlas es diferente como se muestra en la siguiente sintaxis:

constants c1 [(longitud)] [tipo t] [decimales d] [valor 'xxx'].

Donde: **c1** es el nombre de la constante.

longitud es la longitud.

tipo especificación del tipo de dato.

decimales número de dígitos después del punto decimal.

Ejemplos:

```
constants   myname (20)          value  ‘Martín’.
constants   birthday      type d value  ‘19760321’.
constants   zero          type i value is initial.
```

6.4. Declaración de tipos.

El parámetro **TYPES** es utilizado para crear tipos de datos y estructuras de datos.

Types t1 (longitud) [tipo t] [decimales d]

Ejemplos:

1. types: surname (20) type c,
   begin of address
   name type surname,
   end of address.
   Data: address1 type address,
   address2 type address.
2. data conts type i.
   types: company like spgf1_carrid,
   no_flights like counts
3. types: surname(20) type c,
   street(30) type c,
   cp(10) type n,
   city(30) type c,
   phone(20) type n,
   fecha like sy_datum.
   types: begin of address,
   name type surname,
   code type cp,
   town type city,
   str type street,
   end of address.
   types: begin of phone_list,
   adr type address,
   tel type phone,
   end of phone_list.
   data datos1 type phone_list.
   …
   write datos1-adr-name.

6.5 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)**

Offset es la posición apartir de la cual se moverá el dato

Longitud es la contidad de caracteres que se desean mover.

Ejemplo:

```
DATA:     VAR1(15) VALUE 'ESCADOR MADRID',
          VAR2(15) VALUE 'HOLA'.
MOVE VAR1+8(6) TO VAR2+5(6).
WRITE VAR2.
```

Resultado:

HOLA BCN.

VAR1

E	S	C	A	D	O	R		M	A	D	R	I	D	

VAR2

H	O	L	A											

MOVE VAR1+8(6) TO VAR2+5(6).

VAR2

H	O	L	A		M	A	D	R	I	D				

Si se desean utilizar variables en el offset o la longitud se usará la instrucción **WRITE TO,** la cual es la única forma de poder hacer esto.

Ejemplo:

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

6.6 Conversión de tipo.

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

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

En la siguiente tabla se ve la asignación por el ejemplo en la primera fila se leería como:

C = C, D = C, F = C, N = C, P = C, T = C, X = C.

	C	D	F	N	P	T	X
C	**A** **2**	A 2	ERROR 2,00000000000000E+00	ERROR 2	ERROR 2	A00000 200000	A0 20
D	1	**23061999**	7,29929000000000E+05	1	729.929	ERROR	49
F	* *	05010001 00000000	**3,54000000000000E+00** **-3,5400000000000E+00**	4 4	4 4-	000004 235956	04 FC
N	7	7	7,00000000000000E+00	**7**	7	700000	07

P	* *	0405001	1,23000000000000E+02 -123,000000000000E+02	3 3	**123** **123-**	000203 232557	7B 85
T	1	ERROR	6,83800000000000E+04	1	68.380	**185940**	1C
X	5	01040001	9,00000000000000E+01	0	90	000130	**5A**

Nota: para pasar un número con decimales de C a P, no debe tener puntos de mil y, además, el separador de decimales ha de ser '.', no',.

6.7 Operaciones Aritméticas en ABAP/4.

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

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

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

  donde la palabra COMPUTE es opcional.

  Por ejemplo:

  Código

```
data:   var1 type n  value  '3',
        var2 type c value  '2',
        var3 type n .
var3 = var1 + var2. 'o podría ser COMPUTE var3 = var1 +  var2., el
                                  resultado es el mismo
Write 'VAR3 =', var3.
```

  Salida

```
Var3 = 5.
```

  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**.

6.8 Procesando campos de tipo texto.

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

- Para realizar un desplazamiento del contenido de un campo utilizamos **SHIFT**.

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

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

Ejemplo:

H	O	L	A	

SHIFT campo BY 2 PLACES.

L	A			

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

Ejemplo:

H	O	L	A	

SHIFT campo BY 2 PLACES CIRCULAR.

L	A		H	O

- Podemos reemplazar el contenido de ciertos campos con la instrucción **REPLACE**.

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

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

REPLACE únicamente sustituirá la **primera** aparición de 'cadena1'.

Ejemplo: *d*ata field(10).

move 'ABCB' to field.
replace 'B' with 'string' into field.
write: field.

Salida AstringCB

- 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 a sustituir.

Ejemplo:

Código

```
data var1(20) type c value 'Escador'.
translate var1 to UPPER CASE.
write:/ var1.
```

Salida

ESCADOR.

- La instrucción **SEARCH** busca una cadena dentro de un campo o una tabla.

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

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

Ejemplo:

Tabla1

NOMBRE	*SEXO*
Ariel Martínez	M
Lorena Jimenez	F
Miriam Trinado	F
Sergio Mondragon	M

Código

```
tables:  tabla1.
data  var1(10)  type  c  value 'Lorena'.
search  tabla1  for  var1.

if sy-subrc = 0.
      Write: 'LA CADENA FUE ENCONTRADA EN LA LINEA:', sy-tabix.
else.
      Write: 'LA CADENA NO FUE ENCONTRADA'.
endif.
```

Salida.

LA CADENA FUE ENCONTRADA EN LA LINEA: 2

En este ejemplo, se puede ver que sy-subrc es igual a cero, lo que significa que si existe la cadena dentro de la tabla1, una vez que entra al if, imprime con la ayuda de la variable del sistema la línea en la que se encuentra.

- 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.

- Podemos dividir un texto en varios campos con la instrucción **SPLIT**.

Esta instrucción separa en varias campos un texto tomando como referencia un carácter central. Es decir, indicandole un carácter clave guarda todo el texto que tiene a su izquierda en un campo, y el que tiene a su derecha en otro campo.

SPLIT <campo> AT <caracter> INTO <campo1> <campo2> … <campoN>

Ejemplo:

Data: Texto(20) value 'Toni, Juan, Pedro',

Nom1(10), nom2(10), nom3(10).

SPLIT texto AT ',' into nom1 nom2 nom3.

Salida: nom1 = 'Toni', nom2 = 'Juan', nom3 = 'Pedro'.

- Podemos juntar varios campos en un texto con la instrucción **CONCATENATE.**

Esta instrucción concatena varios campos en un texto. Como ya se puede imaginar, realiza la tarea contraria a SPLIT.

CONCATENATE <campo1> <campo2> … <campoN> INTO <texto>

SEPARATED BY <carácter>.

Ejemplo: Tomamos los datos del anterior ejemplo. Nom1 = 'Toni', nom2 = 'Juan', nom3 = 'Pedro',

Concatenate nom1 nom2 nom3 into Texto separated by ','.

Salida: nombre1 = 'Toni', nombre2 = 'Juan', nombre3 = 'Pedro'.

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 que 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	<> , **NE**
Mayor	> , **GT**
Menor	< , **LT**
Mayor o igual	>= , **GE**
Menor o igual	<= , **LE**

Además existen operadores adicionales para comparar cadenas de caracteres.

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

Ejemplos:

'ABCDE' CO 'XYZ' => SY-FDPOS = 0

'ABCDE' CO 'AB' => SY-FDPOS = 5

'ABCDE' CO 'ABCDE' => SY-FDPOS = 5

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

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

Ejemplos:

'ABCDE' CA 'CY' => SY-FDPOS = 2

'ABCDE' CA 'XY' => SY-FDPOS = 5

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

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

Ejemplos:

'ABCDE' CS 'CD' => SY-FDPOS = 2.

'ABCDE' CS 'XY' => SY-FDPOS = 5.

'ABCDE' CS 'AB' => SY-FDPOS = 0.

' ABC' CS ' AB' => SY-FDPOS = 1.

'ABCDEF ' CS ' ' => SY-FDPOS = 0.

<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.

Ejemplos:

'ABCDE' CP '*CD*' => SY-FDPOS = 2.

'ABCDE' CP '*CD' => SY-FDPOS = 5.

'ABCDE' CP '++CD+' => SY-FDPOS = 0.

'ABCDE' CP '+CD*' => SY-FDPOS = 5.

'ABCDE' CP '*B*D*' => SY-FDPOS = 1.

<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 INITIAL. Para chequear valores iniciales.

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

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

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

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

Ejemplo:

```
tables T100.
select * from T100
         where sprsl = sy-langu and   msgnr  <  '010'
         order by primary key.
  on change of t100-arbgb.
      uline.
       write:/ '***', t100-arbgg, '***'.
  endon.
  Write:/ t100-msgnr, t100-text.
endselect.
```

Salida:

```
***  &H     ***
001     Program "&" does not exist.
```

002 Program “&” is not an online program.
003 The source text for program “&” is not available.
004 Macro “&” does not exist in table TRMAC.

*** 00 ***
000
001
02 Please enter a valid value.
03 Memory consuption display switched on.

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 (si no estamos usando eventos).

Ejemplo:

```
DO 4 TIMES
        CHECK SY-INDEX BETWEEN 2 AND 3.
        WRITE SY-INDEX.
ENDDO.
```

Produce la siguiente salida:

```
2       3
```

Aquí el programa termina la primera y la cuarta pasada del bucle sin procesar la sentencia WRITE porque SY-INDEX no está entre 2 y 3.

- 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.

Ejemplo:

```
DO 4 TIMES.
        IF SY-INDEX = 3.
                EXIT.
        ENDIF.
        WRITE SY-INDEX.
ENDDO.
```

Produce la siguiente salida:

```
                1       2
```

El sistema termina el procesamiento del bucle en la tercera pasada sin ejecutar el WRITE ni la cuarta pasada.

- La instrucción: **CONTINUE**

Termina un bucle incondicionalmente. Sólo se puede usar dentro de bucles.

Después de la sentencia CONTINUE, el sistema salta todas las sentencias siguientes en el bloque en el que estamos y continúa con la siguiente pasada.

Ejemplo:

```
DO 4 TIMES.
        IF SY-INDEX = 2.
                CONTINUE.
        ENDIF.
        WRITE SY-INDEX.
ENDDO.
```

Produce la siguiente salida:

1 3 4

El sistema termina la segunda pasada del Do sin procesar la sentencia WRITE.

- La instrucción : **STOP.**

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

- La instrucción : **LEAVE.**

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

8 FORMATEANDO UN LISTADO.

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

8.1 Formato de los datos de salida.

- Para visualizar un valor utilizaremos la sentencia **WRITE**.

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

donde:

/ Indica si queremos saltar una línea o no antes de empezar a imprimir en pantalla (opcional).

Offset Indica la columna donde empezará la impresión (opcional).

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

Ejemplo:

Código.

```
data:  nombre(10) type c  value 'Maribel',
       edad(2)      type n   value '25',
       telef (20)   type c   value '91-746-62-26',
       prom         type p   decimals 2 value '8.75'.
write:/10  'NOMBRE:', nombre, 20 'EDAD', edad.
write:/10  'TELEFONO', (9)telf+3.
write:/10  'PROMEDIO', prom.
```

Salida.

```
NOMBRE: Maribel   EDAD:  25
TELEFONO:  746-62-26
PROMEDIO:  8.75
```

Comandos Adicionales a **WRITE.**

WRITE .... v1 [opción(es)]

(1) **under** v2 | **no-gap**
(2) **using edit mask** m | **using no edit mask**
(3) **mm/dd/yy** | **dd/mm/yy**
(4) **mm/dd/yyyy** | **dd/mm/yyyy**
(5) **mmddyy** | **ddmmyy** | **yymmdd**
(6) **no-zero**
(7) **no-sign**
(8) **decimals** n
(9) **round** n
(10) **currency** c | **unit** u
(11) **left-justified** | **centered** | **right-justified**

donde:

.... opciones de la forma simple
v1 es una literal, variable o el nombre de algún campo.
m es una mascara.
c es una moneda.
u es una unidad.
n es un literal numérico o una variable.

Las siguientes opciones se refieren a:

- ✓ Para cualquier tipo de dato

left-justified	Salida justificada a la izquierda.
Centered	Salida centrada.
right-justified	Salida justificada a la derecha.
under v2	Salida que empieza directamente bajo el campo v2.
no-gap	El blanco después del campo v1 es omitido.
no-zero	Si el campo contiene únicamente ceros, son remplazados por blancos.

- ✓ Campo numérico

no-sign	El signo no aparece en la salida.
decimals n	*n* define el número de dígitos después del punto decimal.
round n	En el tipo P el campo es multiplicado por 10**(-*n)* y es redondeado.
exponent e	En el campo F, el exponente es definido por *e*.
Currency m	Para visualizar importes correctamente dependiendo del importe, donde m es la moneda

- ✓ Campo de fechas

mm/dd/yy	Fecha con separadores (mes/dia/año) y año de 2 dígitos.
dd/mm/yy	Fecha con separadores (dia/mes/año) y año de 2 dígitos.
mm/dd/yyyy	Fecha con separadores (mes/día/año) y año de 4 dígitos.
dd/mm/yyyy	Fecha con separadores (día/mes/año) y año de 4 dígitos.
mmddyy	Fecha sin separadores (mes/día/año) y año de 2 dígitos.
ddmmyy	Fecha sin separadores (día/mes/año) y año de 2 dígitos.
yymmdd	Fecha sin separadores (año/mes/día) y año de 2 dígitos.

using no edit mask Desactiva la opción del uso de máscaras para los datos de salida.

edit mask puede servir para:
Insertar caracteres hacia la salida.
Mover signos al principio de un campo númerico.
Insertando artificialmente o moviendo un punto decimal.
Desplegando un número punto-flotante sin usar notación científica.

En la sentencia **edit mask** el guión bajo **(_)** tiene un significado especial. Al principio de un **edit mask**, **V, LL, RR** y **==** también tienen un significado especial. Todos los demás caracteres son transferidos sin cambio a la salida.

El *guión bajo* en un **edit mask** es remplazado uno a uno con el carácter de un campo (variable). El número de caracteres se toma del campo que será igual al número de guiones bajos en el **edit mask**. Por ejemplo si son 3 caracteres en el **edit mask**, a lo más 3 caracteres del campo (variable) serán de salida.

Por ejemplo, la sentencia:

write (6) 'ABCD' **using edit mask** '_:__:_'.

Produce como salida:

A:BC:D

Los caracteres son tomados del campo uno a la vez, y los caracteres desde el **edit mask** son insertados por la máscara (mask).

La sentencia **write 'ABCD' using edit mask '_:__:_'** únicamente escribe **A:BC** porque por defecto la longitud de la salida es igual a la longitud del campo (4) debido a que va contando uno a uno los caracteres del campo y los de la máscara.

Si hay algunos guiones bajos más que en el campo, por defecto toma los *n* caracteres más a la izquierda del campo, donde *n* es igual al número de guiones bajos en el **edit mask.**

Esto se puede especificar explícitamente precediendo a **edit mask** con **LL.** Por ejemplo:

WRITE 'ABCD' using edit mask 'LL__:_' toma los 3 caracteres más a la izquierda y escribe **AB:C**. Usando **RR** toma los 3 caracteres más a la derecha, por lo que **WRITE 'ABCD' using edit mask 'RR__:_'**

Escribirá **BC:D**.

Si son más los guiones que los caracteres del campo el efecto de **LL** es justificar a la izquierda el valor de la salida, **RR** justificará el valor a la derecha.

Cuando un **edit mask** comienza con **V**, cuando es aplicado a un campo númerico (tipo I,P y F) causa que el signo sea desplegado al comienzo si se aplica a un campo de tipo carácter, **V** será la salida.

Ejemplo:

Código

```
data:   f1(4)  value  'ABCD',
        f2(5) value  '1234-'.
Write:/ '1. ', f2,
        '2. ', f2 using edit mask 'LLV_____',
        '3. ', f2 using edit mask 'RRV_____',
        '4. ', f2 using edit mask 'RRV____,_',
        '5. ',f1 using edit mask 'V____'.
```

Salida

1. 1234-
2. –1234
3. - 123
4. –123,4
5. VABC

Una conversión de salida es una llamada de subrutina que formatea la salida. Un **edit mask** que comience con == seguido por cuatro caracteres ID llama una función que formatea la salida. Esos 4 caracteres ID es conocido como una conversión de salida o conversión de rutina. El nombre de la función puede ser CONVERSION_EXIT_XXXX_OUTPUT, donde XXXX son los 4 caracteres ID que siguien a ==. Por ejemplo **WRITE** '00001000' **using edit mask '==ALPHA'** llama a la función CONVERSION_EXIT_ALPHA_OUTPUT. La sentencia write pasa el valor primero ala función, la cual cambiará algunas cosas y regresará el valor y este será escrito en la salida.

Cuando utilizamos la instrucción **WRITE** con números empaquetados, el sistema trunca por la izquierda en caso de ser necesario (deja un * como indicador de que ha truncado) y rellena con blancos si sobra espacio. Tenemos que tener en cuenta que si es negativo el signo ocupará una posición. Si se

especifican los decimales con la cláusula DECIMALS del DATA, el punto o coma decimal también ocupará una posición. El signo decimal (punto o coma) estará determinado por los valores del registro de usuario.

Ejemplo:

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

y si no cabe el número:

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

- Podemos imprimir una línea de horizontal con la sentencia **ULINE**. Tendrá las mismas propiedades que el WRITE. Nota: con algunos monitores, la línea no se ve.

  ULINE / <offset>(<long>). o **WRITE / <offset>(<long>) SY-ULINE.**

- Para imprimir una línea vertical se necesita la sentencia **WRITE.**

  WRITE / <offset>(<long>) SY-VLINE.

- Para saltar una o varias líneas utilizaremos **SKIP,** en donde *n* es el número de líneas en blanco.

  SKIP <n>. Por defecto el salto será de una única línea.

- Para saltar una página utilizaremos **NEW-PAGE**.

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

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

  WRITE <campo2> UNDER <campo1>.

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

  SKIP TO LINE <n>.

- Podemos modificar los atributos de pantalla para un campo.

  FORMAT INTENSIFIED ON/OFF. Afecta el color del background.

  FORMAT INVERSE OFF/ON. Afecta el color del background y foreground.

  FORMAT INPUT OFF/ON. Indica cuando el usuario puede introducir datos.

  FORMAT COLOR n. Color de la línea background, y *n* puede tomar los valores de la tabla1.

  FORMAT RESET. Restablece todos los formatos anteriores.

N	***Código***	***Color***	***Intensidad para***
Off o COLBACKGROUND	0	Depende del GUI	Background
1 ó COL_HEADING	1	Gris-Azul	Cabeceras
2 ó COL_NORMAL	2	Gris claro	Cuerpo de las listas
3 ó COL_TOTAL	3	Amarillo	Totales
4 ó COL_KEY	4	Azul-Verde	Columnas llaves
5 ó COL_POSITIVE	5	Verde	Valor positivo
6 ó COL_NEGATIVE	6	Rojo	Valor negativo
7 ó COL_GROUP	7	Violeta	Niveles de grupos

TABLA1

8.2 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.

Para cambiar la cabecera de una página individualmente se puede usar **TOP-OF-PAGE**. Este evento ocurre tan pronto como el sistema empieza a procesar una nueva página. El sistema procesa la sentencia siguiente de TOP-OF-PAGE antes de la salida de la primera línea de la nueva página.

Estos eventos son independientes, es decir, se puede usar uno y el otro no, o ambos según sea necesario.

Hay que recordar que para finalizar el bloque de procesos siguientes a **TOP-OF-PAGE** se puede usar **START-OF-SELECTION.**

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 ⟶ Impresora o pantalla.

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.

8.3 Selección de parámetros.

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.

PARAMETERS: p <*opcion*>

Opciones:

(1) *DEFAULT*.- Le asigna un valor al parámetro.

(2) *TYPE typ*.- Asigna el tipo *typ* a un campo interno.

Ejemplo:

parameters number(4) **type** p **default '**999'.

(3) *DECIMALS dec*.- *dec* determina los número de lugares decimales en el campo. d*ec* debe de ser numérico.

Ejemplo:

parameters number(4) **type** p **decimals** 2 **default** '123.45'.

(4) *LIKE **g**.*- Crea un campo **p** con los mismos atributos que un campo **g**, el cual ya fue definido. **g** puede ser un campo de una base de datos o una campo interno existente.

Ejemplo:

parameters fecha **like** sy-datum**.**

Salida:

(5) *LOWER CASE*.- Permite introducir minúsculas.

(6) *OBLIGATORY*.- Obliga a introducir un valor.

(7) *AS CHECKBOX*.- Despliega un parámetro como un checkbox. Si no se especifica el tipo o la longitud cuando se define el parámetro, este será de tipo C y de longitud 1.

El chechecckbox es desplegado hacia la izquierda y hacia la derecha el texto. Para definir algun orden , se debe usar **selection-screen.**

(8) *RADIOBUTTON GROUP radi*.- Despliega el parámetro en el **selection-screen** como un radio button. Todos los parámetros asignados en esta forma para el mismo grupo *radi (*el cual puede estar definido con 4 caracteres de longitud).

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

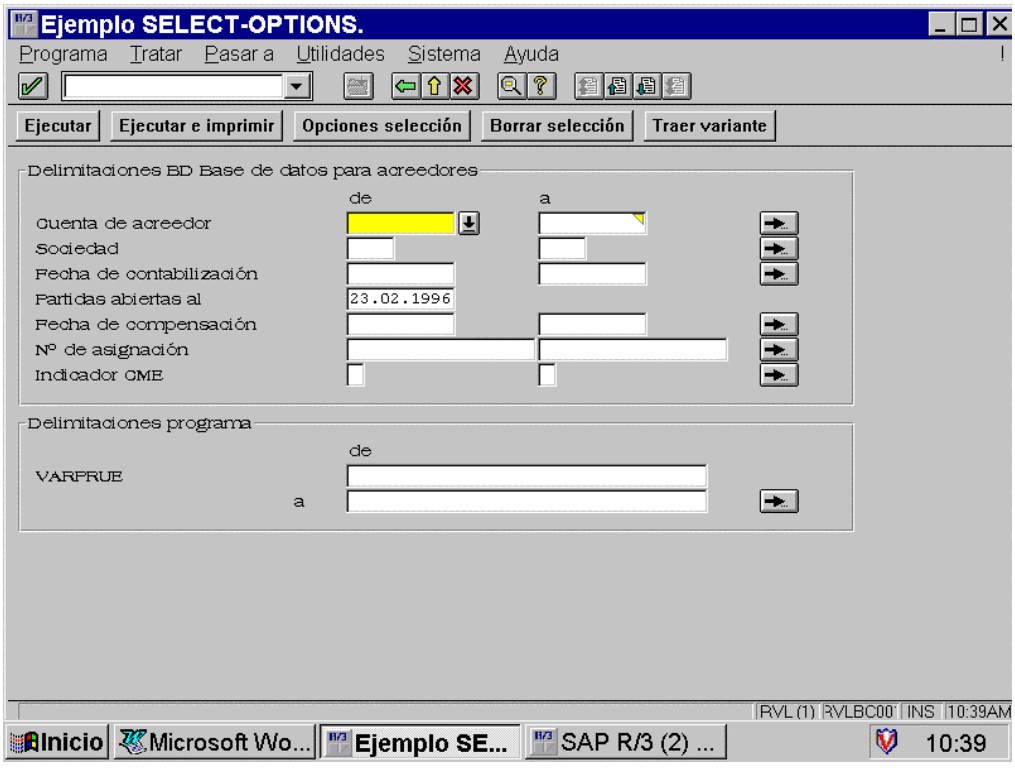
SELECT-OPTIONS <var> FOR <campo_tabla>.

<var> como mucho tendrá 8 caracteres.

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

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

Ejemplo :



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

El contenido de cada registro será respectivamente: el valor inferior, el superior, el signo (**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 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 qué otras **opciones** existen en la utilización de la instrucción **SELECT-OPTIONS.**

- Para asignar valores iniciales a un criterio de selección utilizamos la cláusula **DEFAULT**.
  SELECT-OPTIONS <var> FOR <campo> DEFAULT '<valor>'.

Si queremos inicializar un rango de valores (inferior y superior) usaremos:
SELECT-OPTIONS <var> FOR <campo> DEFAULT '<ini>' TO '<fin>'.

- Podemos hacer que se acepten valores en minúsculas.
  SELECT-OPTIONS <var> FOR <campo> LOWER CASE.
- Podemos obligar a que se introduzcan valores de selección inevitablemente.
  SELECT-OPTIONS <var> FOR <campo> OBLIGATORY.
- También es posible desactivar la posibilidad de introducir selecciones con condiciones compuestas. (Desaparecerá la flecha).
  SELECT-OPTIONS <var> FOR <campo> NO-EXTENSION.
- Otra opción es que no salga un intervalo, sino una sola caja. Es parecido a un parameter, con la diferencia que, si no se rellena un parameter, el valor es ' ', mientras, que, si no rellena un select-options, el valor es todo el rango de valores posibles.
  SELECT-OPTIONS <var> FOR <campo> NO INTERVALS.

8.4. Pantalla de selección (SELECTION-SCREEN).

También es posible formatear a nuestro gusto la pantalla de selección con **SELECTION-SCREEN** utilizando las siguientes opciones:

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

Permite la combinación de varios parámetros y comentarios especificados entre las sentencias **selection-screen begin of line** y **selection-screen end of line** y sus salidas en una línea**.** Como resultado, no hay automaticamente línea nueva para cada **parameters** y los textos de selección se despliegan.

Ejemplo:

Código

```
selection-screen begin of line.
        selection-screen comment 1(10) text-001.
        parameters: p1(3), p2(5),  p3(1).
selection-screen end of line.
```

Textos

Text-001 ‘Comment’

Salida

Comment ___ _____ _

Nota: No se puede colocar **select-option** entre **selection-screen begin of line** y **selection-screen end of line** porque se generan varios objetos en la selección de pantalla para un **select-option (**por ejemplo los campos de más bajo y más alto límites del rango**).**

- **SELECTION-SCREEN SKIP n**

Genera *n* líneas en blanco. Se puede especificar el valor para *n* entre 1 y 9. Si se quiere que solo deje una línea en blanco se puede omitir *n.*

- **SELECTION-SCREEN ULINE <col> (<long>)**

Genera una línea horizontal, se le puede especificar la posición en la que va a empezar con el parámetro **col**, y su longitud con el parámetro **long.**

- **SELECTION-SCREEN POSITION pos**

La posición de la salida de los parámetros es proporcionada por la variable **pos**, en caso de que no se coloque esta instrucción, la posición de los parámetros cero 0.

- **SELECTION-SCREEN COMMENT <col> TEXT-nnn.**

Introduce comentarios para un parametro, en donde **TEXT-nnn,** será el text simbol que contenga el comentario que se desea introducir, y con **col** se posiciona el comentario.

- **SELECTION-SCREEN BEGIN OF BLOCK** block <**WITH FRAME**><**TITLE**>**.**

Comienza un bloque lógico en la pantalla de selección, en donde **block** es el nombre de dicho bloque. Si se le coloca la opción **WITH FRAME,** se genera un marco alrededor del bloque. La opción **TITLE** coloca el nombre de ese bloque, se puede utilizar solamente si está también la opción **FRAME**.

- **SELECTION-SCREEN END OF BLOCK** block.

Cierra el bloque abierto por **SELECTION-SCREEN BEGIN OF BLOCK** block. Si el bloque tiene un frame o marco, el marco se cierra también con esta instrucción.

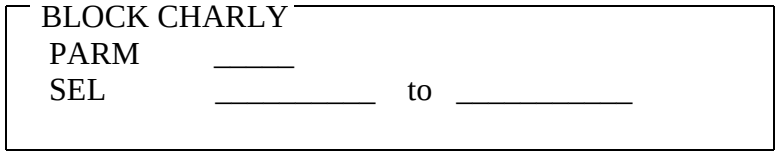
Ejemplo:

(Text-001 contiene 'BLOCK CHARLY')

Código

```
tables saplane.
selection-screen begin of block charly with frame title text-001.
        parameters parm(5).
        select-option sel for saplane-planetype.
selection-screen end of block charly.
```

Salida



- **SELECTION-SCREEN NEW-PAGE.**

Sirve para utilizar varias páginas de selección.

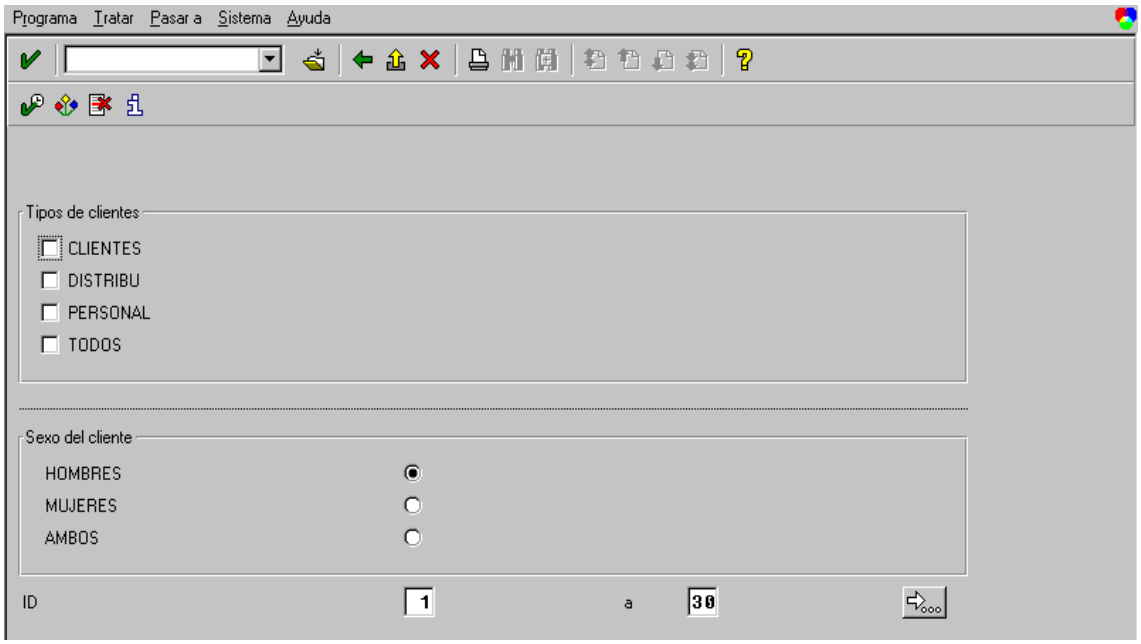
A continuación se muestra un ejemplo, en el cual se observa el uso de varias de las sentencias anteriores.

Código:

```
report zlb40.
tables: zclient.
selection-screen skip 2.
selection-screen begin of block caja1 with frame title text-001.
        parameters: clientes as checkbox,
                    distribu as checkbox,
                    personal as checkbox,
                    todos    as checkbox.
selection-screen end of block caja1.
selection-screen uline.
selection-screen begin of block caja2 with frame title text-002.
        parameters: hombres radiobutton group rad1,
                    mujeres  radiobutton group rad1,
                    ambos    radiobutton group rad1.
selection-screen end of block caja2.
selection-screen begin of block caja3.
        select-options id for zclient-cla_client default '1' to '30'
```

selection-screen end of block caja3.

Salida:



Una instrucción que nos permite verificar los datos de entrada en la pantalla de selección es la siguiente:

AT SELECTION-SCREEN ON <campo>.

…

ENDAT.

8.5 Elementos de texto y Mensajes.

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

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

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

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

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

Con los **Textos numerados** podemos utilizar constantes de tipo texto sin necesidad de declararlas en el código del programa. Los nombres de las constantes serán **TEXT-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. (SE91)**

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.
X	=	Exit. Transacción terminada con un short dump.

Ejemplo:

Código

```
report  zlb401.
parameter:  num type i.
if num > 10.
   message id 'Z1' type 'E' number '001'.
endif.
```

Salida

Este resultado se observa en la Barra de Estado, cuando se introduce un número mayor de 10.

Código

```
report  zlb401.
parameter:  num type i.
if num > 10.
   message id 'Z1' type 'A' number '001'.
endif.
```

Salida

Este mensaje se observa en la pantalla, al presionar la tecla de EXIT, el sistema regresa al pantalla principal del área de Desarrollo.

Código

```
report  zlb401.
parameter:  num type i.
if num > 10.
   message id ‘Z1’ type ‘I’ number ‘001’.
endif.
```

Salida

Este mensaje nos permite intentar nuevamente a colocar un valor y ejecutar el programa.

Código

```
report  zlb401.
parameter:  num type i.
if num > 10.
   message id ‘Z1’ type ‘S’ number ‘001’.
endif.
```

Salida

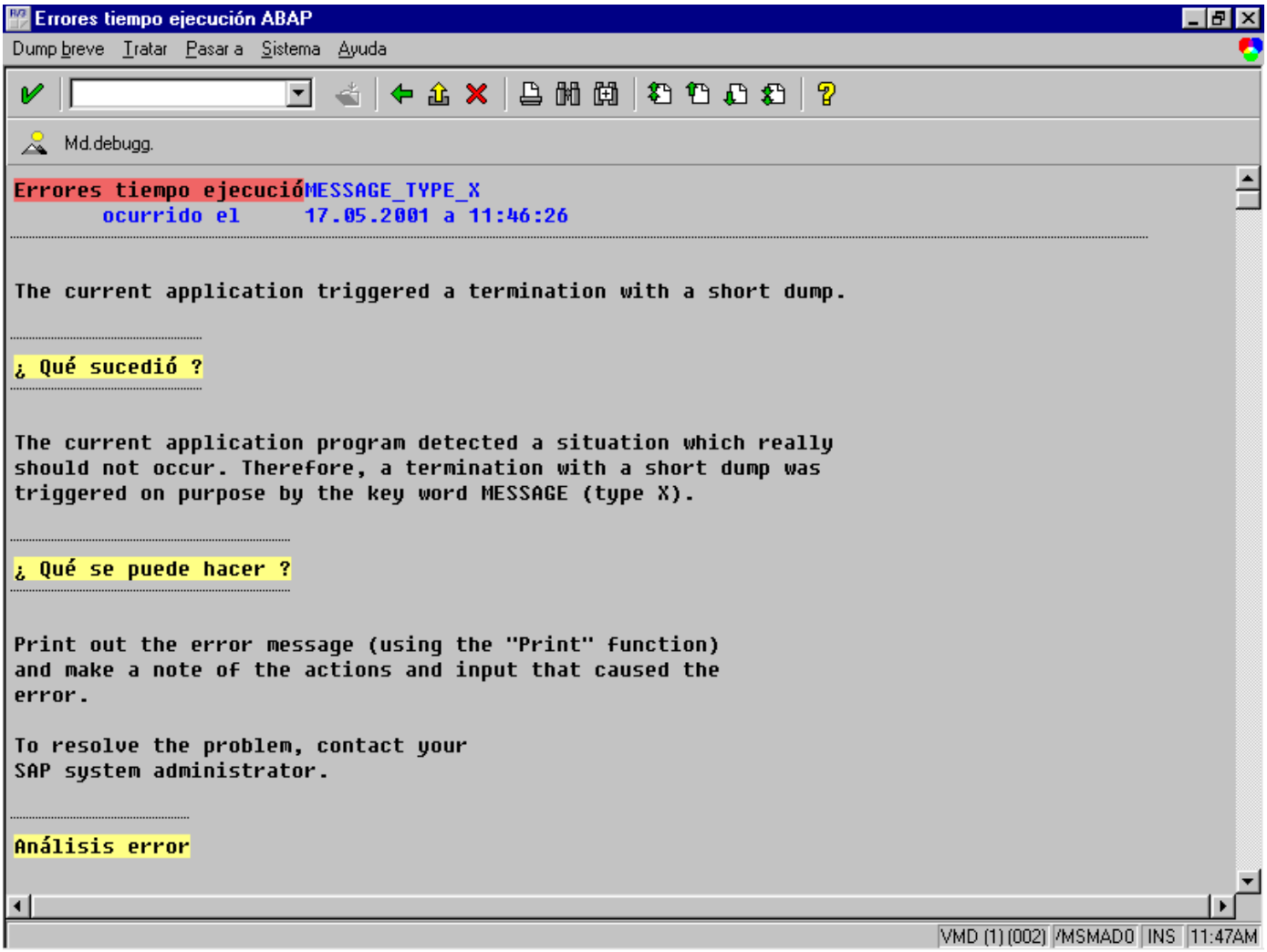
Este mensaje nos permite cambiar el valor del campo solicitado.

Código

```
report  zlb401.
parameter:  num type i.
if num > 10.
```

message id ‘Z1’ type **‘X’** number ‘001’.
endif.

Salida



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:

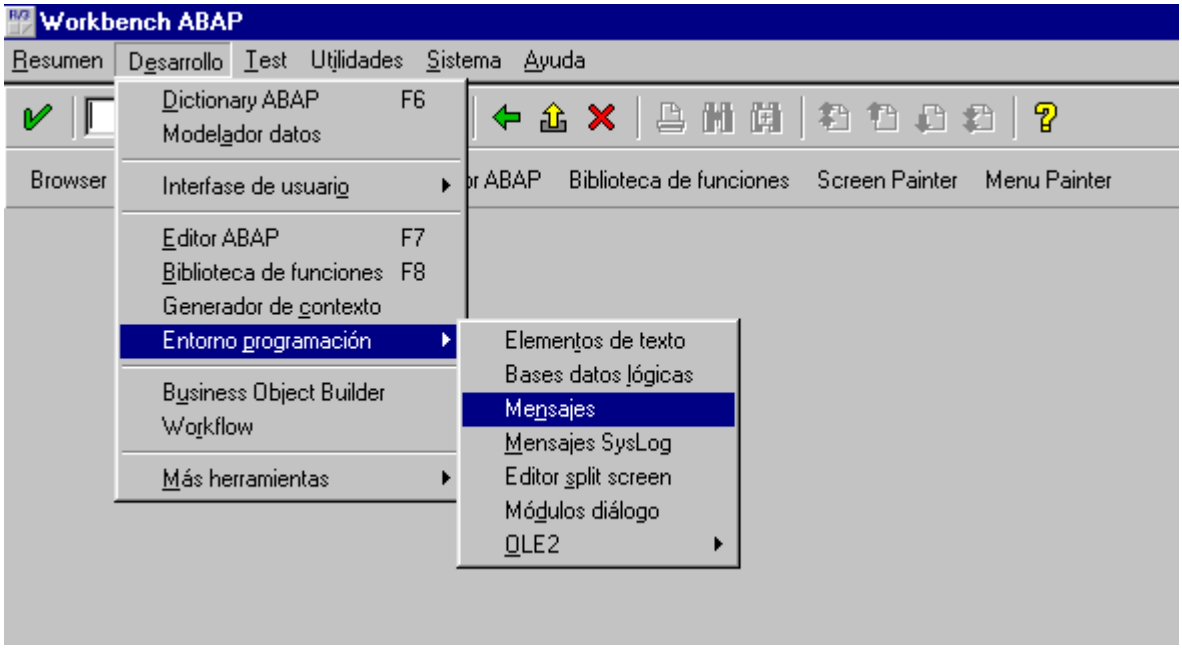
Área de mensajes ZZ.
Mensaje : 005 = Entrada &-& incorrecta.
REPORT ZPRUEBA MESSAGE-ID ZZ.
IF ....
MESSAGE A005 WITH SKA1 KTOPL.

ENDIF.

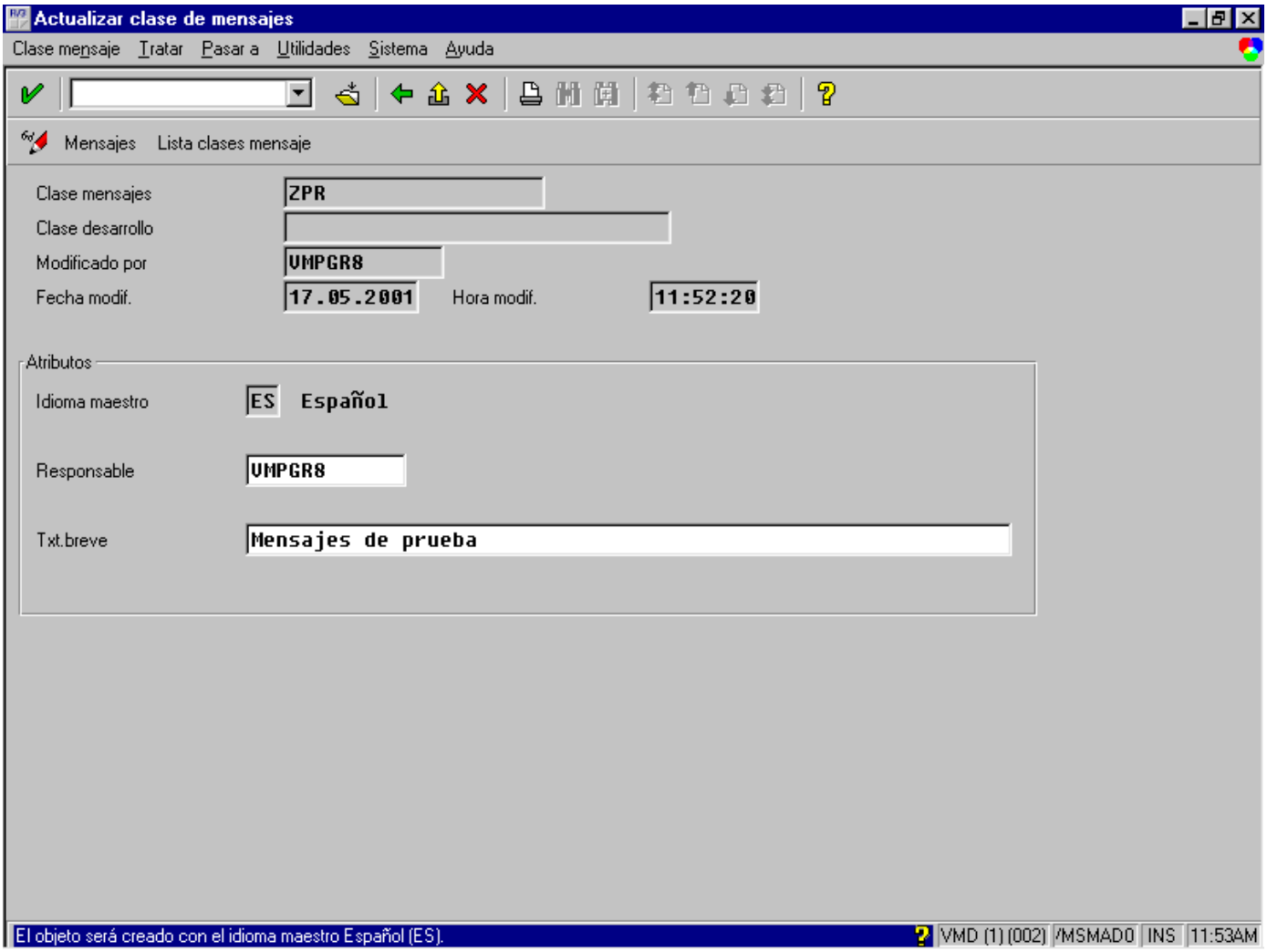
El mensaje obtenido será :

A: Entrada SKA1-KTOPL Incorrecta

Para crear un **área de mensaje**, se tiene que realizar los siguientes pasos:



1. Dentro de una sesión de ABAP Workbench se debe seleccionar del menú la opción **Desarrollo**, posteriormente la opción **Entorno de programación** y finalmente **Mensajes.** Tambien se puede acceder al mismo punto a traves de la transacción SE91.
2. En la pantalla siguiente solo se debe colocar el nombre del área de mensaje y presionar el botón **Crear**.
3. La siguiente pantalla es en donde se introducirá el título del área de mensajes.
4. Finalmente se debe de salvar.


Actualizar clase de mensajes
Clase mensaje Tratar Pasar a Utilidades Sistema Ayuda
Mensajes Lista clases mensaje
Clase mensajes ZPR
Clase desarrollo
Modificado por VMPGR8
Fecha modif. 17.05.2001 Hora modif. 11:52:20
Atributos
Idioma maestro ES Español
Responsable VMPGR8
Txt.breve Mensajes de prueba
El objeto será creado con el idioma maestro Español (ES).
VMD (1) (002) /MSMAD0 INS 11:53AM

9 TABLAS INTERNAS.

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

9.1 Como declarar tablas internas.

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

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

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

El parámetro OCCURS n define cuántas entradas en la tabla son creadas inicialmente. El valor de n no es significativo excepto cuando se usa la instrucción “APPEND SORTED BY f ”. Esta instrucción inserta la nueva entrada en la tabla y la ordena en orden descendente de acuerdo al campo f. Cuando el número de entradas de la tabla alcanza el valor n que se ha declarado con el OCCURS, la última entrada será borrada si el valor de f de la nueva entrada es mayor. Sólo se puede ordenar por un campo.

Ejemplo:

```
DATA: BEGIN OF COMPANIES OCCURS 3,
            NAME(10),
            SALES TYPE I,
END OF COMPANIES.

COMPANIES-NAME = ‘big’.
COMPANIES-SALES = 90.
APPEND COMPANIES.

COMPANIES-NAME = ‘small’.
COMPANIES-SALES = 10.
APPEND COMPANIES.

COMPANIES-NAME = ‘too small’.
COMPANIES-SALES = 5.
APPEND COMPANIES.

COMPANIES-NAME = ‘middle’.
COMPANIES-SALES = 50.
APPEND COMPANIES SORTED BY SALES.
CLEAR COMPANIES.
```

Ahora la tabla tiene tres entradas (=> OCCURS 3). La línea con la entrada ‘too small’ en el campo NAME será borrada ya que la entrada ‘middle’ tiene un valor más grande en el campo SALES. Esa entrada aparecerá en la tabla en segundo lugar (después de ‘big’ y antes de ‘small’).

NOTA: Hay que tener en cuenta que si se utiliza el APPEND con el parámetro SORT BY, el sistema cada vez recorrerá toda la tabla; así pues, a veces, será mejor llenar la tabla con APPEND y una vez se ha terminado de llenar, ordenarla en orden ascendente o descendente.

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, como se muestra en el ejemplo anterior.

  APPEND <intab>.

  NOTA: Después de realizar un APPEND debe ir un CLEAR <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. Si no coincide con ninguno, lo que hará será añadir la línea de cabecera a la tabla.

  COLLECT <intab>

  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 ordena 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., con las siguientes cláusulas se puede especificar:

SORT ... **ASCENDING.** o **DESCENDING.**

Ejemplo: Si tenemos la siguiente tabla *clientes* que esta ordenada por clave.

Clave	Nombre	Número de Cuenta	Sucursal	Saldo
0001000	Ezequiel Nava	80194776-456	123	2,589.50
0001029	Ivonne Rodriguez	82325690-455	122	15,368.00
0001053	Sara Rosas	80000236-456	123	1,500.25
0001082	Alfredo Martínez	79268978-457	124	30,549.61
0001097	Rosalinda Trejo	85689782-457	124	21,987.30
0001256	Javier Solano	81569782-460	135	13,569.20
0001299	Pedro Gamboa	80000468-456	123	5,698.21
0001356	Marco Antonio Balderas	90265874-460	135	29,350.00
0001468	Jorge Villalon	78698014-457	124	62,165.50

Si la desearamos por ordenar por sucursal, tendríamos que escribir la siguiente instrucción.

Sort clientes **by** sucursal.

Y obtendríamos la siguiente tabla.

Clave	Nombre	Número de Cuenta	Sucursal	Saldo
0001029	Ivonne Rodriguez	82325690-455	122	15,368.00
0001000	Ezequiel Nava	80194776-456	123	2,589.50
0001299	Pedro Gamboa	80000468-456	123	5,698.21
0001053	Sara Rosas	80000236-456	123	1,500.25
0001082	Alfredo Martínez	79268978-457	124	30,549.61
0001468	Jorge Villalon	78698014-457	124	62,165.50
0001097	Rosalinda Trejo	85689782-457	124	21,987.30
0001256	Javier Solano	81569782-460	135	13,569.20
0001356	Marco Antonio Balderas	90265874-460	135	29,350.00

9.4 Procesamiento de una tabla interna.

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

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

...

ENDLOOP.

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

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

Si no existe ningún registro de la tabla que cumpla la condición especificada en la cláusula WHERE, la variable del sistema **SY-SUBRC** será distinta de **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.	Realiza las instrucciones que hay a
... ⟶	continuación del AT FIRST para la
ENDAT.	primera entrada de la tabla.
AT NEW <campo>.	Realiza las instrucciones que hay a
... ⟶	continuación del AT NEW para cada
ENDAT.	inicio de nivel de ruptura.
AT LAST.	Realiza las instrucciones que hay a
... ⟶	continuación del AT LAST para la
ENDAT.	última entrada de la tabla.
AT END OF <campo>.	Realiza las instrucciones que hay a
... ⟶	continuación del AT END para cada
ENDAT.	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.

Ejemplo: Considerando la tabla *clientes*, y queremos obtener la suma del saldo por sucursal, y saldo total de esos clientes no importando la sucursal, tendremos el siguiente código.

Código:

```
Loop at clientes_tab.
        At new sucursal.
                Write:/  'Sucursal', clientes_tab-sucursal.
        Endat.

Write:/ clientes_tab-clave, 15 clientes_tab-nombre, 25 clientes_tab-cuentas, 35 clientes_tab-saldo.
        At end of sucursal.
                Sum.
                Write:/ 'Total de la sucursal:',  35 clientes_tab-saldo.
                Uline.
                Skip.
        Endat.

        At last.
                Sum.
                Write:/ 'Total de las sucursales:', 35 clientes_tab-saldo.
        Endat.
```

Endloop.

Salida:

Sucursal: 122			
0001029	Ivonne Rodriguez	82325690-455	15,368.00
Total de la Sucursal:			15,368.00
Sucursal: 123			
0001000	Ezequiel Nava	80194776-456	2,589.50
0001299	Pedro Gamboa	80000468-456	5,698.21
0001053	Sara Rosas	80000236-456	1,500.25
Total de la Sucursal:			9,789.96
Sucursal: 124			
0001082	Alfredo Martínez	79268978-457	30,549.61
0001468	Jorge Villalon	78698014-457	62,165.50
0001097	Rosalinda Trejo	85689782-457	21,987.30
Total de la Sucursal:			114,702.41
Sucursal: 135			
0001256	Javier Solano	81569782-460	13,569.20
0001356	Marco Antonio Baldera	90265874-460	29,350.00
Total de la Sucursal:			42,919.20
Total de las Sucursales:			167,411.57

9.6 Lectura de entradas de una tabla.

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

  READ TABLE <intab>.

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

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

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

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

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

- Podemos buscar por clave con:

  READ TABLE <intab> WITH KEY <clave>.

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

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

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

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

  READ TABLE <intab> INDEX <num>.

9.7 Modificando tablas internas.

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

- **MODIFY** :Podemos sobreescribir el contenido de la entrada <i> con el contenido de la línea de cabecera.

  MODIFY <intab> (INDEX <i>).

  NOTA: No modifica solo un campo sino modifica todo el registro (renglón).

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

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

  INSERT <intab> (INDEX <i>).

- **DELETE** : Para borrar una entrada de una tabla.

  DELETE <intab> (INDEX <i>).

Otras instrucciones de manejo de tablas:

- Inicializar el área de trabajo o línea de cabecera.

  CLEAR <intab>.

- Inicializar (borrar) contenido de una tabla.

  REFRESH <intab>.

  Esta instrucción borra toda la tabla excepto la linea de cabecera

- Liberar el espacio ocupado por una tabla en memoria.

  FREE <intab>.

- Para obtener información sobre una tabla interna.

  DESCRIBE TABLE <tab>

  LINES <contador_entradas>.

10 BASE DE DATOS. Como leer y procesar tablas.

10.1 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 : Tablas, campos, Elementosde datos y dominios. Los **elementos de datos** describen el significado de un campo independientemente de las tablas donde se utilicen. Es decir, tienen un carácter semántico. Los **dominios** describen el campo de valores posibles. Tendrán un carácter técnico.

Ejemplo :

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

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

Existen diversos tipos de tablas:

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

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

10.2 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.

10.3 Instrucciones SQL de ABAP/4.

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

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

Además de las variables del sistema:

SY-SUBRC : Código de retorno de una operación.

SY-DBCNT : Cantidad de registros afectados por la operación procesada.

10.3.1 SELECT.

La sentencia **SELECT** será la instrucción fundamental para leer información de la base de datos.

- **Lectura de un único registro :**

  SELECT SINGLE * FROM <tab>

  WHERE <cond>.

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

Si **SY-SUBRC** = 0 Registro encontrado. Resultado en área de trabajo.

Si **SY-SUBRC** = 4 No existe el registro buscado.

- **Lectura Iterativa** : Selección de un grupo de registros (su sintaxis es la siguiente).

  SELECT [DISTINCT] * FROM <tab> INTO <lugar>

  (WHERE <cond>) (GROUP BY <campos>) (ORDER BY <campos>) .

  ENDSELECT.

Donde:

Distinct Excluye duplicados de líneas.

Into Determina el área (blanco) hacia el cual los datos seleccionados sean colocado para ser leídos posteriormente.

Where Especifica cuales son las líneas que serán leidas especificando condiciones de selección.

Group by Produce una sola línea de resultado desde grupos de varias líneas. Un grupo de líneas con los mismos datos.

Order by Proporciona un orden en las líneas seleccionadas por los campos <campos>

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

Si **SY-SUBRC** = 0 Algún registro encontrado.

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

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

Otras posibilidades del WHERE:

SELECT * FROM <tab> WHERE <campo> ...

BETWEEN <var1> AND <var2>. → Si <campo> está entre los valores <var1> y <var2>.

LIKE <literal enmascarado>. → Si <campo> cumple la máscara. Se pueden utilizar :

'_' como carácter cualquiera.
'%' como una cadena de caracteres.

IN (<var1> , <var2> ...) → Si <campo> esta en el conjunto de valores <var1> ,<var2> ...

- **Otras lecturas :**

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

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

Llena la tabla interna <intab> machacando los registros que pudiera tener esta.

Si queremos que respete los registros que tenía la tabla interna antes de realizar el SELECT tendremos que utilizar :

SELECT * FROM <tab> APPENDING TABLE <intab>

(WHERE <cond>).

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

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

Si queremos seleccionar un registro para bloquearlo de posibles modificaciones.

SELECT SINGLE FOR UPDATE * FROM <tab>.

Ejemplo para un **select ***: Se hará una selección del tabla *clientes* en donde se obtendrá del select, el nombre del cliente, cuenta y saldo, dependiendo de la cuenta, es decir, se obtendrán las cuentas que terminen con **-456**.

Código:

```
Report zlb402.

Tables: clientes.
Data: begin of clientes-tab occurs 100,
        nombre        like    clientes-nombre,
        num_cuenta    like    clientes- num_cuenta,
        saldo         like    clientes-saldo,
        end of clientes-tab.

Select * from clientes where  cuenta like '%-456'
        Order by nombre.
        Move-corresponding clientes to clientes_tab.
        Append clientes_tab.
        Clear clientes_tab.
Endselect.
```

```
Write:/  'Sucursal:', clientes_tab-sucursal.
Loop at clientes_tab.
        Write:/ clientes_tab-nombre, clientes_tab-num_cuenta, clientes_tab-saldo.

        At last.
                Sum.
                Write:/  'Saldo total:', 30 clientes_tab-saldo.
                Uline.
        Endat.
Endloop.
```

Salida:

Sucursal: 123

Ezequiel Nava	80194776-456	2,589.50
Pedro Gamboa	80000468-456	5,698.21
Sara Rosas	80000236-456	1,500.25
Total de la Sucursal		9,789.96

10.3.2. 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 de la BDD.

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.

10.3.3. 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>

en este caso, lo que se hace es aumentar o agregar un valor al valor del campo actual o bien sustraerlo a dicho campo.

Ejemplo del usu de **update:** Considerando la tabla cleintes_tab del ejemplo de select *.

Código:

```
UPDATE clientes SET num_cuenta = '8000235-456'
                    saldo      ='2,000.00'
              Where clave      = '0001053'
```

Select * .....

Salida:

En esta ocasión tedremos en la salida:

Sucursal: 123

Ezequiel Nava	80194776-456	2,589.50
Pedro Gamboa	80000468-456	5,698.21
Sara Rosas	80000235-456	2,000.00
Total de la Sucursal		10,389.96

Es posible modificar registros desde una tabla interna:

UPDATE <tab> FROM TABLE <intab>.

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

Si **SY-SUBRC = 0** Todos los registros modificados.

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

En **SY-DBCNT** Tendremos la cantidad de registros modificados.

10.3.4. MODIFY.

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

MODIFY <tab>.

MODIFY <tab> FROM TABLE <intab>.

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

10.3.5 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>.

Ejemplo usando la tabla clientes_tab del ejemplo select *. Si deseamos borrar al cliente Ezequiel Nava el código será el siguiente:

Código

DELETE FROM clientes **WHERE** nombre = 'Ezequiel Nava'.
Select * from clientes

Salida

Sucursal: 123

Pedro Gamboa	80000468-456	5,698.21
Sara Rosas	80000235-456	2,000.00
Total de la Sucursal		7,698.21

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.

10.4 OTROS ASPECTOS DE LA PROGRAMACIÓN DE BD.

- 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 utilizaremos **FOR UPDATE.**

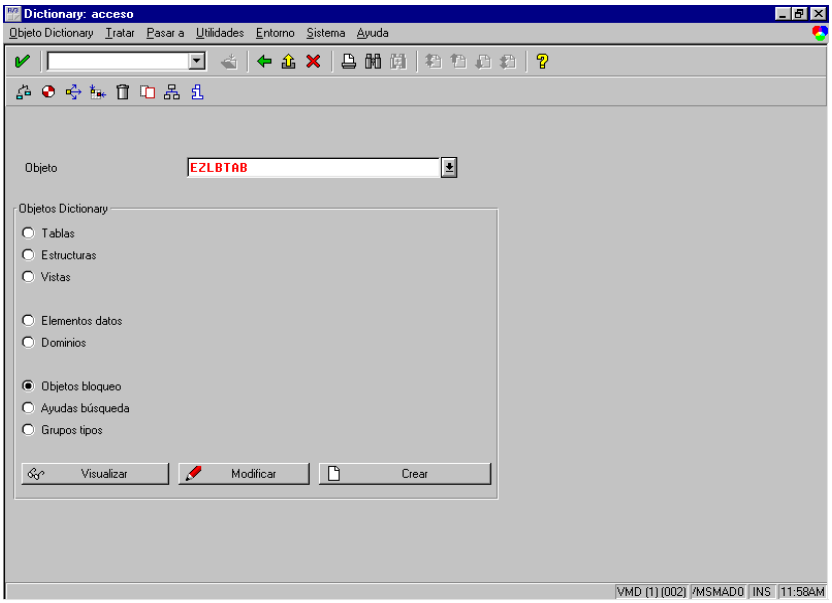
  SELECT SINGLE FOR UPDATE * FROM <tab>

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 operación hasta la última.

Para implementar los bloqueos es necesario utilizar objetos de bloqueo, que pueden ser compuestos de un registro de una cierta tabla o de una colección de registros de una o varias tablas, como se indica en los siguientes pasos:

1. Ir a ABAP4 Dictionary
2. En la pantalla se le debe de colocar en el campo Nombre_del_Objeto la letra E seguida del nombre de la tabla que se desea bloquear.Se selecciona el radio button de Objeto_de_bloqueo (Lock Objects) como se ve en la siguiente figura.

   Después de que se ha presionado el botón para Crear, aparecerá una siguiente pantalla.



1. En la cual en el primer campo que solicita información (**Descripción Breve**) se introducirá una pequeña descripción del objeto creado.
2. En el segundo campo **Tabla Primaria**, se introduce el nombre de la tabla que se desea bloquear.
3. Si se requiere bloquear otras tablas, se ponen en tablas secundarias y en ese caso va a **Pasar a** => **Tablas**.
4. Indicar cual es la relación entre ellas (join).
5. Si se va a **Pasar a** => **Argumentos bloqueo**, aquí se observaran en la pantalla los campos por los que se va a seleccionar la entrada a bloquear, por defecto toma todos los campos clave de las tablas. Solo se tendrá que ir a esta pantalla en caso de que hayan seleccionado varias tablas y coincidan nombre de campos claves entre ellas o si por otro motivo se quiere cambiar el nombre de los argumentos.
6. Verificar, generar y salvar. Es muy importante GENEAR, de otra forma nunca estará activado el objeto.

Al activarlo, inmediatamente se crean las siguientes funciones:

ENQUEUE_<*nombre del objeto de bloqueo*> para bloquearlo.

DEQUEUE_<*nombre del objeto de bloqueo*> para desbloquearlo.

Los parámetros de la función **ENQUEUE** son:

Exporting

- *arg* and *xarg* Estos dos parámetros existen para cada campo *arg* en el argumento de bloqueo. Se coloca en *arg* el valor del campo llave. Si no se especifica el valor para el campo, se omite el parámetro *arg*. Si se desea el valor inicial del campo como el actual valor, se indica colocando en *x_arg* una **'X'.**
- *_SCOPE* Toma los valores 1,2,3 e indica en que momento se va a desbloquear la entrada, si en su programa antes de llamar al programa que actualiza las tablas de la base de datos, si lo desbloquea dicho programa, o si hay que desbloquear en los dos sitios en el programa de dialogo (el suyo) y en el programa que actualiza las tablas.
- *COLLECT* Por defecto tiene un blanco y es loq ue debe de tener ya que sino no lo bloquea directamente sino que lo deja en espera hasta que llamas a la función FLUSH_ENQUEUE.
- *_WAIT* Indica si en caso de que esté bloqueada esa entrada; si debe esperar o no.

Exceptions

Después de llamar a la función ENQUEUE, se deberá checar que el objeto ha sido bloqueado en el programa. Las siguientes excepciones están definidas en el módulo de la función.

- *FOREING_LOCK* El objeto ha sido bloqueado por otro usuario. La variable del sistema SY_MSGV1 contiene el nombre de este usuario.
- *SYSYTEM_FAILURE* Error general del sistema.

Para ***desbloquear el objeto***, solo basta con escribir la siguiente sentencia.

CALL FUNCTION DEQUEUE_<*nombre del objeto de bloqueo*>.

NOTA: Si cuando se le da el nombre del objeto de bloqueo, el nombre de la tabla corresponde a una que haz creado tú, todo es inmediato, PERO si es una tabla de SAP, te pide una clave de acceso para crear el objeto de bloqueo. Esa clave la tienes que pedir por OSS a SAP.

- **Actualización de la base de datos o Recuperación** :

Para finalizar una unidad de procesamiento lógico (**LUW**) de base de datos se utiliza un **COMMIT WORK**, que realiza un UPDATE físico en la base de datos, haciendo irrevocable cualquier modificación en la base de datos.

Si deseamos deshacer todas las operaciones realizadas sobre la base de datos desde el último COMMIT WORK, realizaremos un **ROLLBACK WORK.**

- **Chequeo de autorizaciones**:

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

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

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

AUTHORITY-CHECK OBJECT <objeto_de_autorización>

ID <Campo1> FIELD <f1>

ID <Campo2> FIELD <f2>

ID <Campo3> DUMMY.

...

Donde <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.

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

AUTHORITY CHECK OBJECT 'F_LFA1_BUK'

ID 'ACTVT' FIELD '01'

ID 'BUKRS'FIELD '0001'.

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

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.

10.5 BASES DE DATOS LOGICAS

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 (BD 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 a traves de la 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 SE36, el sistema generará todo lo necesario para utilizar la base de datos lógica, incluyendo el programa de lectura.

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

En el programa que va a utilizar bases de datos lógicas será necesario especificar en los atributos del programa la LDB que va a ser utilizada. ***Y*** en el código simplemente utilizaremos el evento **GET.**
GET <tablaBDDl>.
<sentencias evento>
GET <tablaBDD2>.
<sentencias evento>

Mediante el GET dispondremos de un registro de la base de datos que especifiquemos, siempre y cuando esta tabla esté dentro de la estructura de la base de datos lógica. Para comunicar el programa de lectura con nuestro report se utiliza el **PUT,** que suministra el registro de la BDD que especifiquemos, previamente habrá realizado el SELECT.
PUT <tablaBDD>.

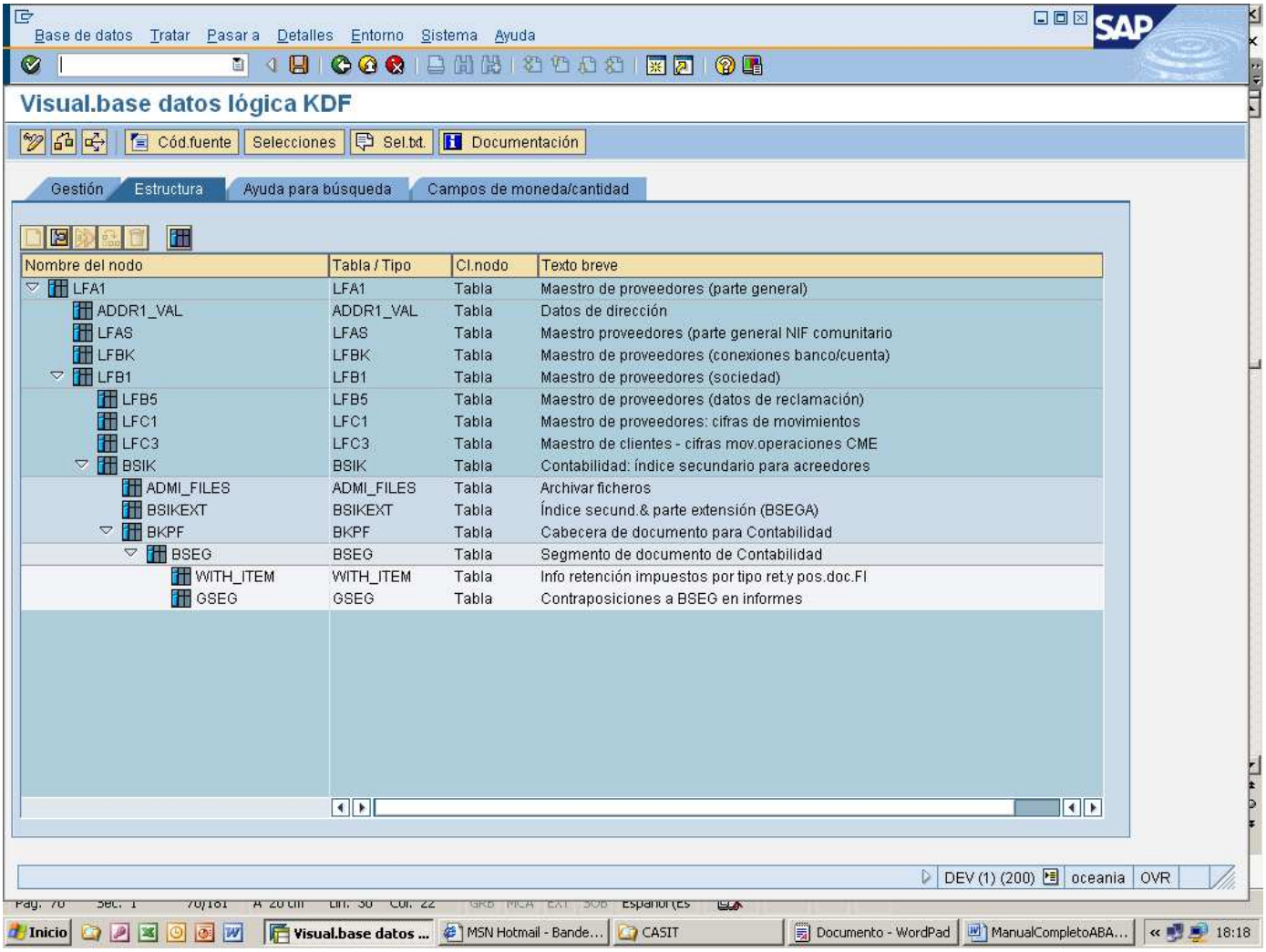
Una base de datos lógica tiene tres componentes fundamentales
- Una definición de la **estructura de las tablas** que utiliza.
- Una **pantalla de selección** de los datos a leer. (SELECT-OPTIONS)
- Un **programa de lectura** de datos de la BDD. (PUT).

También existe la posibilidad de utilizar el evento:
GET <tabBDD> LATE.

Este evento se produce cuando se han procesado todas las entradas de tablas subordinadas a un registro de datos de una tabla, y antes de que el sistema solicite la siguiente entrada de la misma tabla (mismo nivel jerárquico).
Existe una instrucción de salto o finalización de lectura de una tabla, REJECT. Esta instrucción sale del

proceso del registro en curso y continúa 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>.**

Estructura de la BDD KDF

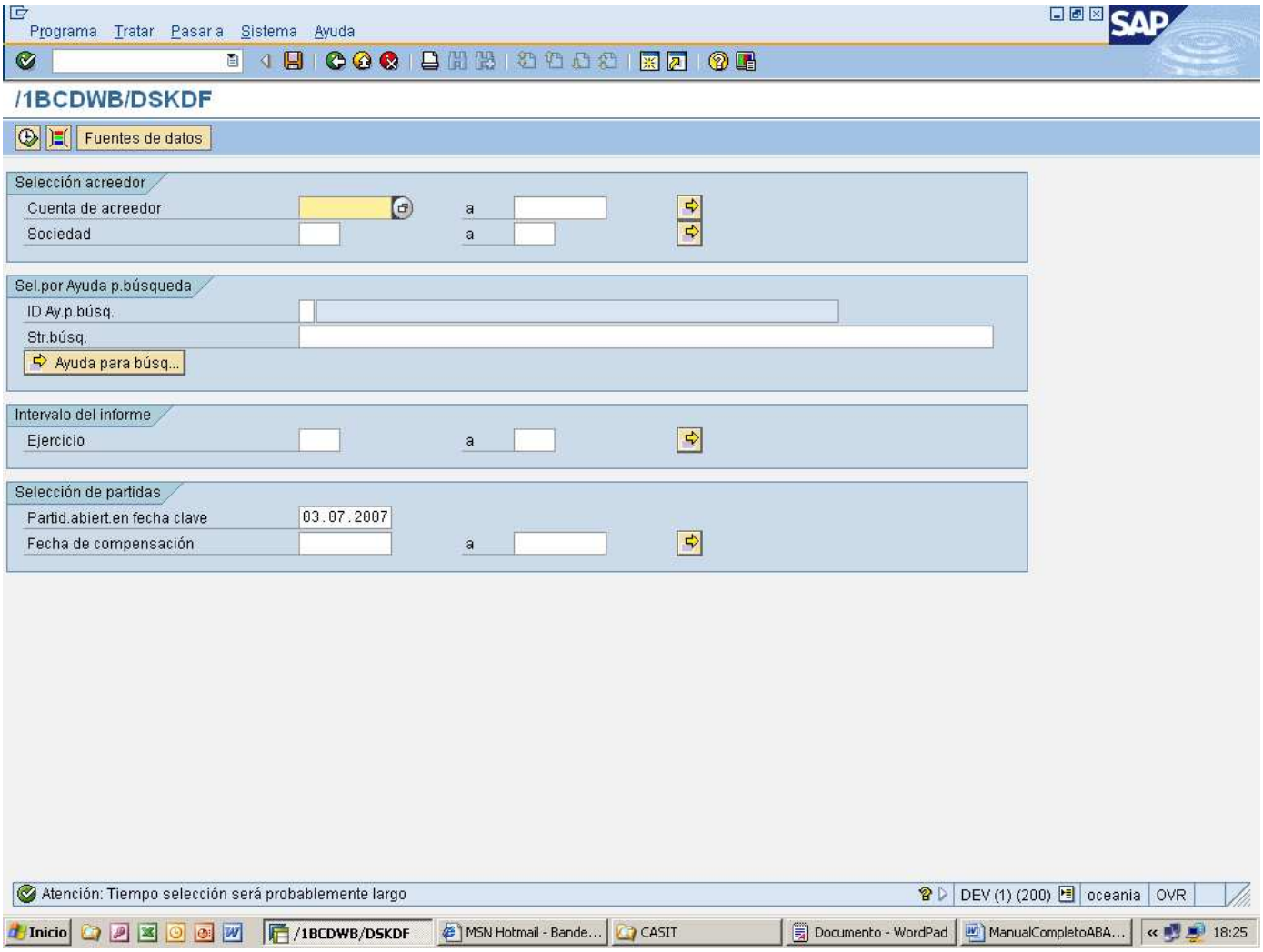


No siempre se necesitan datos de todas las tablas de la estructura de la base de datos lógica. Por ejemplo podemos estar interesados solamente en datos de la tabla LFAT, con lo cual:

TABLES: LFA1.
GET LFA1.

Pero si quisiéramos datos únicamente de la tabla BSIK, deberíamos declarar bajo TABLES todas las tablas que haya en el “path” hasta llegar a la que nos interesa. Así pues quedaría:
TABLES: LFAT, LFB1, BSIK.
GET BSIK.

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 desanollar muchos reports con una estmctura de lectura muy similar, y esta no está en ninguna base de datos lógica, utilizaremos la transacción SE36.



Pantalla deseleccion para Base de datos logica

11 FIELD-GROUPS.

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

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

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

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

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

FIELD-GROUP : HEADER, <f_g_1>, <f_g_2>…

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

Para determinar qué 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. (Utilizaremos la base de datos lógica KDF).

TABLES: LFA1, LFB1, BSIK.

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

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

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

EXTRACT DIRECCION

PROVEEDOR1	RIVERLAND DIAGONAL 618 BARCELONA

EXTRACT IMPORTES

PROVEEDOR1 DOC1	100.000

Así el dataset se irá llenando :

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

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

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

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

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

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

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

```
AT DIRECCION WITH IMPORTES.

      WRITE: LFA1-NAME1 …..
```

```
ENDAT.
```

También podemos contar o sumar por campos con las instrucciones:

CNT (<campo>).

SUM (<campo>).

Así podríamos completar nuestro listado de proveedores del ejemplo con:

```
END-OF-SELECTION.
  SORT.
  LOOP.
      AT DIRECCION WITH IMPORTES.
          WRITE: LFA1-NAME1, LFA1-STRAS,
                 LFA1-PSTLZ, LFA1-ORT01.
      ENDAT.
      AT IMPORTES.
          WRITE: BSIK-BELNR, BSIK-DMBTR.
      ENDAT.
      AT END OF LFA1-LIFNR.
          SKIP.
          WRITE: 'Suma proveedor', LFA1-LIFNR,
                 SUM(BSIK-DMBTR).
          SKIP.
      ENDAT.
  ENDLOOP.
```

12 SUBRUTINAS.

Una subrutina es una sección de código que puede ser usado más de una vez. Es como un mini-programa que puede ser llamado desde otro punto de tu programa. Dentro de él se puedn declarar variables, ejecutar sentencias, realizar cálculos y escribir salidas a la pantalla. Para definir una subrutina, se usa la sentencia **FORM** para indicar el comienzo de una subrutina, y se usa **ENDFORM** para indicar el final de la subrutina. El nombre de uan subrutina no puede exceder de 30 caracteres.

Para llamar a una subrutina, se usa la sentencia. **PERFORM.**

12.1 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 interfaz de llamada claramente definido.

12.2 Subrutinas internas.

PERFORM <modulo>. ⟶ Llamada a un procedimiento o subprograma.

FORM <modulo>

.... ⟶ Subprograma.

ENDFORM.

Se pueden pasar datos entre los programas y las subrtuinas utilizando parámetros.

- Los parámetros definidos durante la definición de una subrutina con la sentencia FORM se llaman **parámetros formales.**
- Los parámetros especificados durante la definición de una subrutina con la sentenciaq PERFORM se llaman **parámetros actuales.**

Se pueden distinguir varios tipos de parámetros:

- Parámetros de entrada, se usan para pasar datos a las subrutinas.
- Parámetros de salida, se usan para pasar datos de las subrutinas.
- Parámetros de entrada/salida, se usan para pasar datos de y hacia las subrutinas.

```
FORM <subr>  [ TABLES      <formal  table  list> ]
                  [ USING        <formal  input  list> ]
                [CHANGING <formal  output  list>] ...
PERFORM <subr> [(<program>)] [TABLES       <formal  table  list> ]
                                  [USING          <formal  input  list> ]
                                     [CHANGING    <formal  output  list>] ...
```

Las opciones TABLES, USING y CHANGING se deben escribir en la secuencia descrita

La lista de parámetros que va detrás de USING y de CHANGING pueden ser objetos de cualquier tipo y field symbols. Los parámetros de la lista de TABLES son tablas internas. Se pueden pasar tablas internas utilizando TABLES, USING o CHANGING.

Para cada parámetro formalq ue se va detrás del USING o del CHANGING en una sentencia FROM, se puede especificar diferentes métodos de pasar los datos:

- **Llamada por referencia**: Durante una llamada a subrutina, sólo la dirección del parámetro actual se transfiere a los parámetros formales. El parámetro formal no tiene memoria él mismo. Dentro de la subrutina se trabaja con el campo del programa que hace la llamada. Si cambiamos el parámetro formal, el contenido del campo en el programa que hace la llamada también cambia.
- **Llamada por valor:** Durante la llamada a subrutina, los parámetros formales son creados como copias de los parámetros actuales. Los parámetros formales tienen memoria para si mismos. Los cambios en los parámetros formales no tienen efecto en los parámetros actuales.
- **Llamada por valor y resultado:** Durante una llamada a subrutina, los parámetros formales son credos como copias de los parámetros actuales. Los parámetros formales tienen su propio espacio de memoria. Los cambios a los parámetros formales son copiados a los parámetros actuales al final de la subrutina.

Paso de parámetros por referencia:

Los parámetros que se cambian dentro de la subrutina se cambian también en el programa que hace la llamada.

Se utilizarán las opciones USING y CHANGING en las sentencias FORM y PERFORM.

FORM ...[USING <fi1> .... <fin>] [CHANGING <fo1> ... <fon>] ...

PERFORM [USING <ai1> ... <ain>] [CHANGING <ao1> ...<aon>]...

Ejemplo:

Código

```
data:  f1 value ‘A’,
       f2 value ‘B’.

write:/ f1,f2.
perform EJEMPLO1 using f1 changing f2.
write:/ f1,f2.
form EJEMPLO1 using p1 changing p2.
       p1 = p2 = ‘X’.
endform.
```

Salida

```
A  B
X  X
```

Paso de parámetros por valor:

Para asegurarnos de que un parámetro de entrada no se cambia en el programa que hace la llamada, incluso si se cambia en la subrutina, pasaremos los datos a la subrutina por valor. Para ello, usaremos la opción USING de las sentencias FORM y ENDFORM.

FORM . USING ...VALUE(<fii>)....

PERFORM....USING ..<aii> ....

Ejemplo:

Código

```
data: f1 value  'A'.
perform EJEMPLO2 using f1.
write: / f1.
form EJEMPLO2  using value (p1).
        p1 = 'X'
        write: / p1.
enform.
```

Salida

X

A

Paso de parámetros por valor y resultado:

Si queremos devolver un parámetro de salida cambiando desde una subrutina al programa que la llama únicamente después de que la rutina se haya ejecutado correctamente, usaremos CHANGING en la sentencia FORM y ENDFORM.

FORM . CHANGING ...VALUE(<fii>)....

PERFORM....CHANGING ..<aii> ....

Solamente cuando alcanzamos la sentencia ENDFORM el sistema pasa el valor actual de <fii> a <aii>.

Ejemplo:

Código

```
Data: f1 value 'A'.
perform: EJEMPLO3 changing f1,
              EJEMPLO4 changing f1.
end-of-selection.
    write:/ 'Stopped. f1 =', f1.
form EJEMPLO3 changing value (p1).
      p1 = 'B'.
endform.
form EJEMPLO4 changing value (p2).
      p1 = 'X'.
      stop.
endform.
```

Salida

Stopped. f1 = B

Paso de tablas internas a subrutinas:

Podemos pasar tablas internas como parámetros en la lista después de USING y CHANGING en las sentencias FORM y ENDFORM.

Las tablas internas, que son pasadas mediante TABLES, siempre son llamadas por referencia.

En cualquier caso, si queremos acceder a los componentes de una tabla interna, debemos especificar el tipo del correspondiente parámetro formal. De otro modo, sólo podremos ejecutar operaciones con las líneas en la línea.

FORM <subr> TABLES <intab> STRUCTURE <esctructura> USING <var1>.

PERFORM<subr> TABLES<intb> USING<var1>

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

Ejemplo:

Código

```
Types: begin of flight_sctruc,
              Flcarrid  like  sflight-carrid,
              Price     like  sflight-fldate,
       End of flight_struc.

Data:  my_flight  type  flight_struc occurs 0,
       ibook1     like  sbook         occurs 0,
       ibook2     like  ibook1        occurs 0,
       struc       like  sbook.

Perform display using my_flight ibook1 ibook2  struc.
Form display using     p_itab      like   my_flight [ ]
                       p_book1   like    book1 [ ]
                       p_book2   like    book2 [ ]
                       p_struc     like    struct.

       Data:  l_flight  like  line of p_itab,
              l_carrid like  l_flight-flcarrid.
      . . . . . .
       write:/  p_struc-carrid,  p_struc-carrid.
       . . . . .
       loop at p_itab into l_flight where flcarrid = l_carrid.
       . . . . . .
       endloop.
Endform.
```

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.

Dentro de cada subrutina es posible declarar datos con las sentencias 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 podrá utilizar variables con el mismo nombre que variables globales, aunque el valor que tenga 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>

12.3 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 una interface definida ya que su diseño facilita el paso de parámetros tanto de entrada como de salida.

CALL FUNCTION <funcion>

EXPORTING <par_E> = <valor>

IMPORTING <par_S> = <valor_ret>

TABLES <tab_Func> = <tab_Prog>

EXCEPTIONS <excep> = <valor>.

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

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

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

MESSAGE .... RAISING <excepcion>. o

RAISE <excepción>.

Para acceder a la biblioteca de módulos de función es posible utilizar el comando SHOW FUNCTION * desde el editor de programas o desde el tratamiento de módulos de función del menú **Herramientas -> CASE -> Desarrollo -> Actualizar programas -> Módulos de función (transacción SE37)**, desde donde podremos además crearlos y mantenerlos.

Ejemplo de una llamada a función que confirma un paso:

Código:

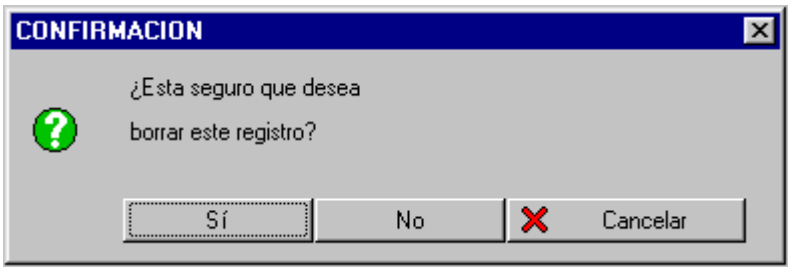
```
Call function 'POPUP_TO_CONFIRM_STEP'
    Exporting
      Titel        =  'CONFIRMACION'
      Textline1    =  '¿Esta seguro que desea'
      Textline2    =  'borrar este registro?'
    Importing
      Answer       =  respuesta.
```

Salida:

La variable **respuesta** que fue definida al principio del programa, puede regresar los valores

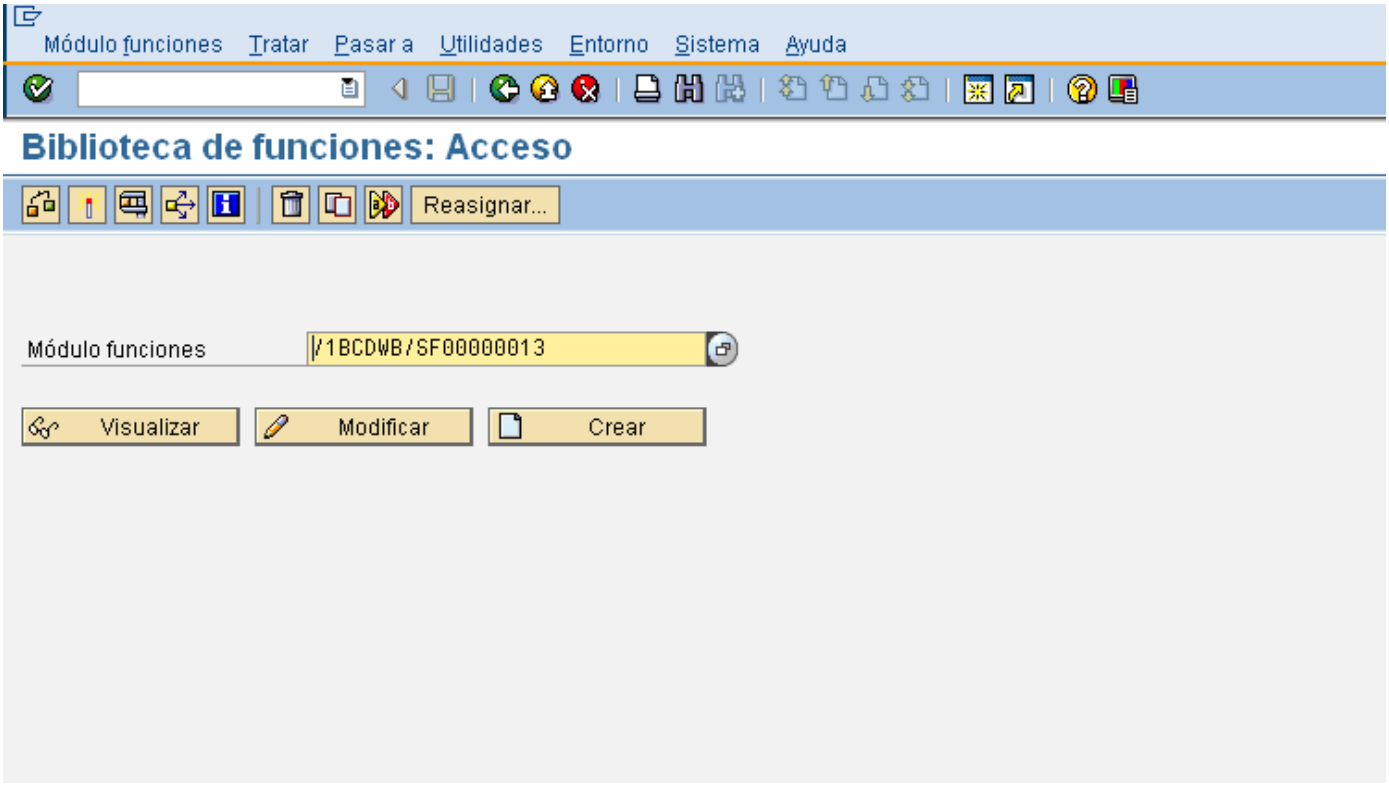
J	Afirma el paso.
N	No ejecuta el paso.
Z	Cancela la pregunta.

Finalmente la pantalla que aparece al invocar esta función es la siguiente:



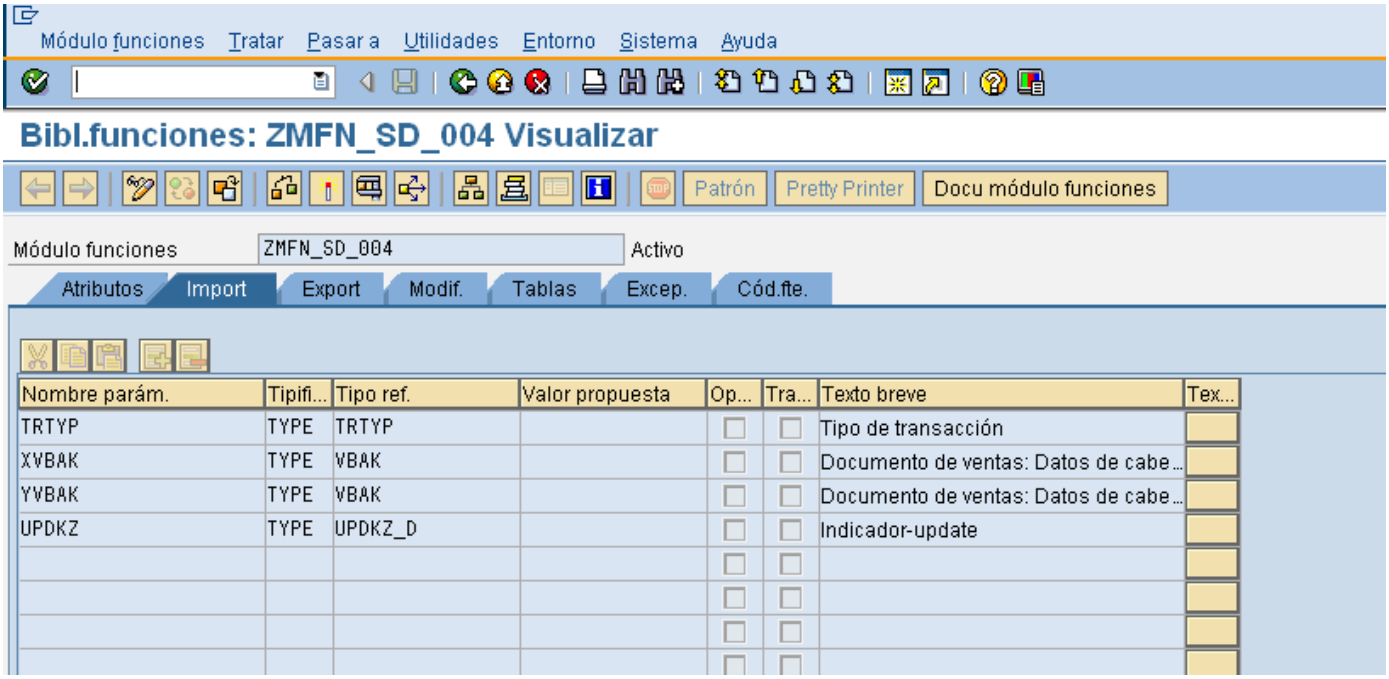
La forma de crear una función es la siguinete:

1. Se debe estar en la pantalla de **ABAP Workbench** seleccionar el boton **Biblioteca de Funciones**, o acceder directamente a la transacción SE37, en donde aparecerá la siguiente pantalla:



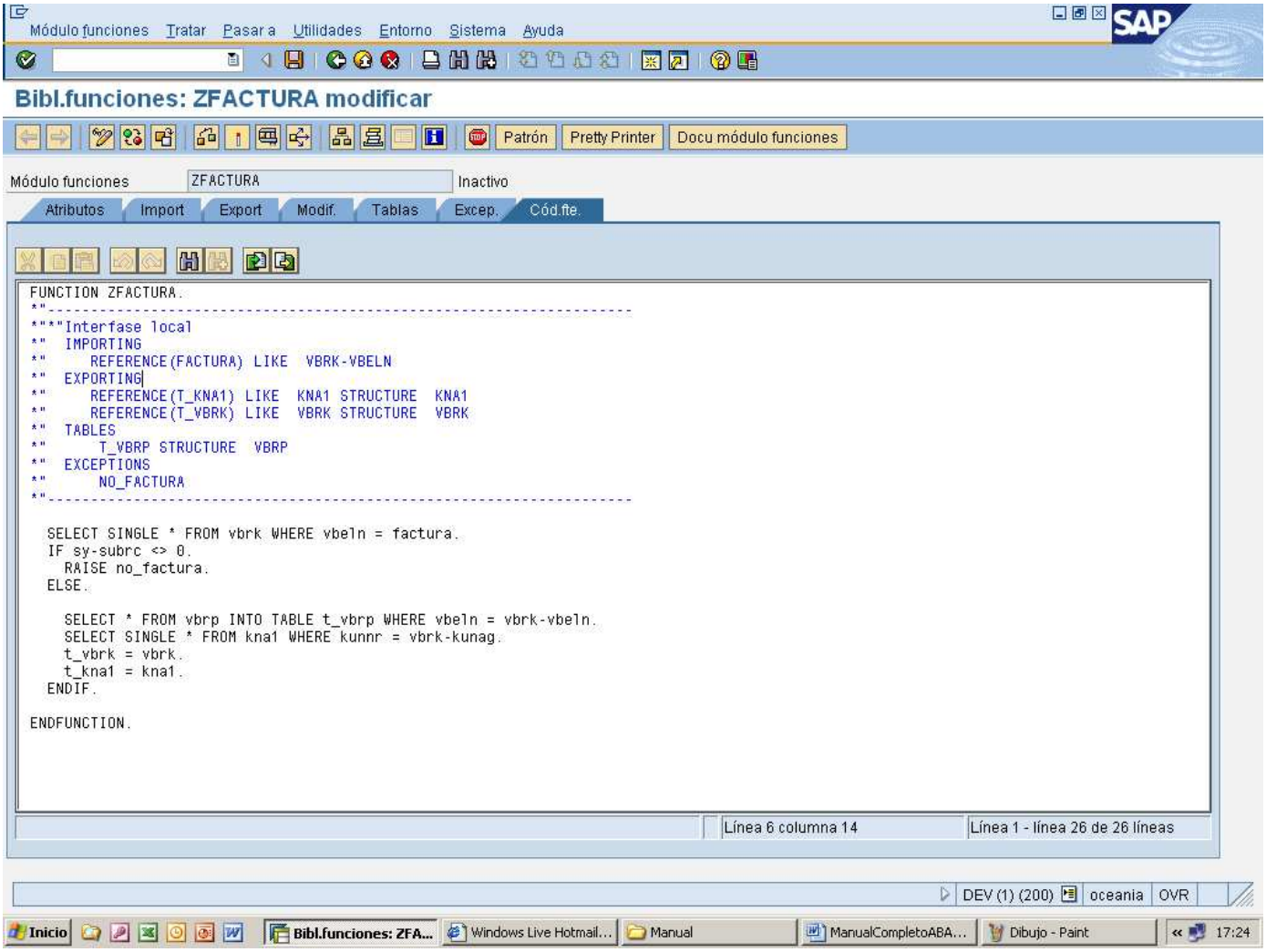
En el campo **Módulo Funciones** se escribirá el nombre de función que se desee crear.

2. Se presionará el botón **Crear**, en caso de que ya exista y lo que se deseé es probar o consultar sus parámetros se presionará el botón **Test ind.**
3. Aparecera una pantalla solicitando el grupo funcional
4. Una vez que se ha introducido el grupo funcional, y se escribirá un texto corto y se debe SALVAR, y GENERAR.
5. Para colocar los parámetros de salida (IMPORT), se selecciona la pestaña que dice IMPORT en la pantalla anterior y se llenará la pantalla con nombre de estos parámetros, estructura, tipo de dato, si es opcional o referencia, se debe SALVAR y GENERAR.
6. Para colocar los parámetros de entrada (EXPORT), se selecciona la pestaña con el título EXPORT y se realiza el mismo procedimiento que con los parámetros de IMPORT se debe SALVAR y GENERAR.



7. Para colocar las tablas que se van a emplear se selecciona la pestaña con el título TABLES, se SALVA y se GENERA.

8. Para colocar los parámetros de excepciones se selecciona la pestaña con el título EXCEPTIONS, se SALVA y se GENERA.
9. En el boton de texto explicativo se escribe el texto que se empleara para cada variable.
10. Y en la pestaña de codigo fuente escribiremos el codigo de la funcion


Módulo funciones Tratar Pasar a Utilidades Entorno Sistema Ayuda
Bibl.funciones: ZFACTURA modificar
Patrón
Pretty Printer
Docu módulo funciones
Módulo funciones
ZFACTURA
Inactivo
Atributos
Import
Export
Modif.
Tablas
Excep.
Cód.fte.
FUNCTION ZFACTURA.
*"----------
""Interfase local
*" IMPORTING
*" REFERENCE(FACTURA) LIKE VBRK-VBELN
*" EXPORTING
*" REFERENCE(T_KNA1) LIKE KNA1 STRUCTURE KNA1
*" REFERENCE(T_VBRK) LIKE VBRK STRUCTURE VBRK
*" TABLES
*" T_VBRP STRUCTURE VBRP
*" EXCEPTIONS
*" NO_FACTURA
*"----------
SELECT SINGLE * FROM vbrk WHERE vbeln = factura.
IF sy-subrc <> 0.
RAISE no_factura.
ELSE.
SELECT * FROM vbrp INTO TABLE t_vbrp WHERE vbeln = vbrk-vbeln.
SELECT SINGLE * FROM kna1 WHERE kunnr = vbrk-kunag.
t_vbrk = vbrk.
t_kna1 = kna1.
ENDIF.
ENDFUNCTION.
Línea 6 columna 14
Línea 1 - línea 26 de 26 líneas
DEV (1) (200)
oceania
OVR
Inicio
Bibl.funciones: ZFA...
Windows Live Hotmail...
Manual
ManualCompletoABA...
Dibujo - Paint
17:24

12.4 SUBMIT.

Submit nos sirve para ejecutar y enviar parámetros a un programa distinto en el que se está. Puede entenderse como una llamada a una función, los únicos requisitos son que se las variables que se deseen pasar del programa1(programa en donde se realiza la llama) al programa2 deben de ser declarados como parámetros en el programa2 (programa donde se ejecutan las acciones con estos parámetros). Con la sentencia RETURN se devuelve el mando al programa inical.

Ejemplo:

Programa1:

```
report prog1.
data:  numero1 type i   value  10,
       numero2 type i.
numero1 =  numero1 * 10.
numero2 = numero1.
Write:/ 'Programa 1'.
submit  prog2 with  num1  =  numero1
              with  num2  =  numero2
              and return.
write:/ 'Programa  terminado '.
```

Programa2:

```
report prog2.
parameters:  num1    type   i,
             num2    type   i .
data:  resultado1 type i.
resultado1 = num1 + num2.
Write:/ num1,'+', num2, '=', resultado1.
```

Salida

Programa 1.

100 + 100 = 200.

Programa terminado.

12.5 Listados ALV.

Se trata de un tipo de listado que genera un report visual muy vistoso que permite una serie de opciones sobre los datos. Permite al usuario reestructurar las columnas, elegir las columnas que quiere o no quiere visualizar, ordenar los registros, e incluso sumarizar los valores numericos agrupando por alguna columna.

El proceso es muy simple, tras haber seleccionado los datos que queremos mostrar por pantala, el programa los mostrará por pantalla en este nuevo sistema usando solamente la funcion 'REUSE_ALV_GRID_DISPLAY'. Pasandole como datos el layout (caracteristicas globales) y las columnas a mostrar, la funcion nos dibuja un listado vistoso con los datos que le pasamos en la estructura T_outtab. A continuacion podemos ver un ejemplo.

```
CALL FUNCTION 'REUSE_ALV_GRID_DISPLAY'
```

```
  EXPORTING
    i_callback_program       = sy-repid
    i_callback_pf_status_set = 'F_STATUS'
    i_callback_user_command  = 'F_USER_COMMAND'
    is_layout                = r_layout
    it_fieldcat              = tfieldcat
    i_save                   = 'A'
  TABLES
    t_outtab                 = t_facturas
  EXCEPTIONS
    program_error            = 1
    OTHERS                   = 2.
```

Donde t_facturas es la tabla con los datos, r_layout son las caracteristicas clave del listado, y tfieldcat contiene la definicion de todos los campos que saldran por pantalla. A continuacion podemos ver la parametrizacion de la tabla tfieldcat.

```
* Número de pedido
  lr_fieldcat-fieldname     = 'VBELN'.
  lr_fieldcat-tabname       = 'T_FACTURAS'.
  lr_fieldcat-key           = 'X'.
  lr_fieldcat-no_zero       = 'X'.
  lr_fieldcat-seltext_l     = 'Num. de Factura'.
  lr_fieldcat-ref_fieldname = 'VBELN'.
  lr_fieldcat-ref_tabname   = 'ZFACTURAS'.
  APPEND lr_fieldcat TO i_fieldcat.
  CLEAR lr_fieldcat.
* Posicion
  lr_fieldcat-fieldname     = 'POSNR'.
  lr_fieldcat-tabname       = 'T_FACTURAS'.
  lr_fieldcat-key           = 'X'.
  lr_fieldcat-seltext_l     = 'Posicion'.
  lr_fieldcat-ref_fieldname = 'POSNR'.
  lr_fieldcat-ref_tabname   = 'ZFACTURAS'.
  APPEND lr_fieldcat TO i_fieldcat.
  CLEAR lr_fieldcat.
```

El campo fieldname y tabname corresponden al nombre del campo y de la tabla interna donde se encuentran los datos. Ref_fieldname y ref_tabname se refieren a las tablas de las que se han extraido los datos, y de las cuales se consigue el tamaño y titulo. Key indica si es clave primaria, y seltext es el titulo que muestra en la cabecera del listado para esa columna.

13 INTERFACES.

13.1 Tipos.

Los interfaces sirven para conectar un sistema SAP con otro sistema cualquiera (también puede ser SAP). Un interface es una utilidad para transferir información de forma segura y automatizada. Hay varios tipos de interfaces: Batch input, Bapis, Idocs, RFCs y Guis.

Un Batch Input es simular mediante un proceso Batch la introducción de datos en el sistema vía transacción online.

Una Bapi (Bussines Application Programming Interface) es una función estándar de SAP que ya hace el interface, con lo que el programador tan sólo tiene que hacer una llamada a esa función.

Al entrar un Idoc al sistema, dispara una función que hace la entrada de datos automáticamente.

Una RFC (Remote Function Call) es una llamada en un sistema (servidor) desde otro sistema (cliente), es decir, desde SAP llamamos a un trozo de código en el sistema que queremos conectar con SAP que ejecuta, dentro de SAP, la entrada de datos.

13.2 BAPIS.

13.2.1 Introducción a las BAPIs

Las BAPIs, Business Application Programming Interfaces (BAPIs) son el núcleo de las herramientas de desarrollo de SAP Business Object Repositoy (BOR). Las BAPIs se introdujeron en ABAP a partir de la versión 3.x pero cobraron importancia en cuanto a número y funcionalidad a partir de la versión 4.x.

SAP implementa BAPIs a través de RFCs. Las BAPIs son un ejemplo perfecto de la potencia de las RFCs. Llegarán a ser mas y más importantes para los programadores en la medida en que la integración de SAP con otros sistemas sea más común.

13.2.2 Definición de BAPI

Una BAPI se define como un método de SAP Business Object. Las BAPIs posibilitan un método cómodo de acceso a los datos y funciones de SAP desde programas tanto externos como internos.

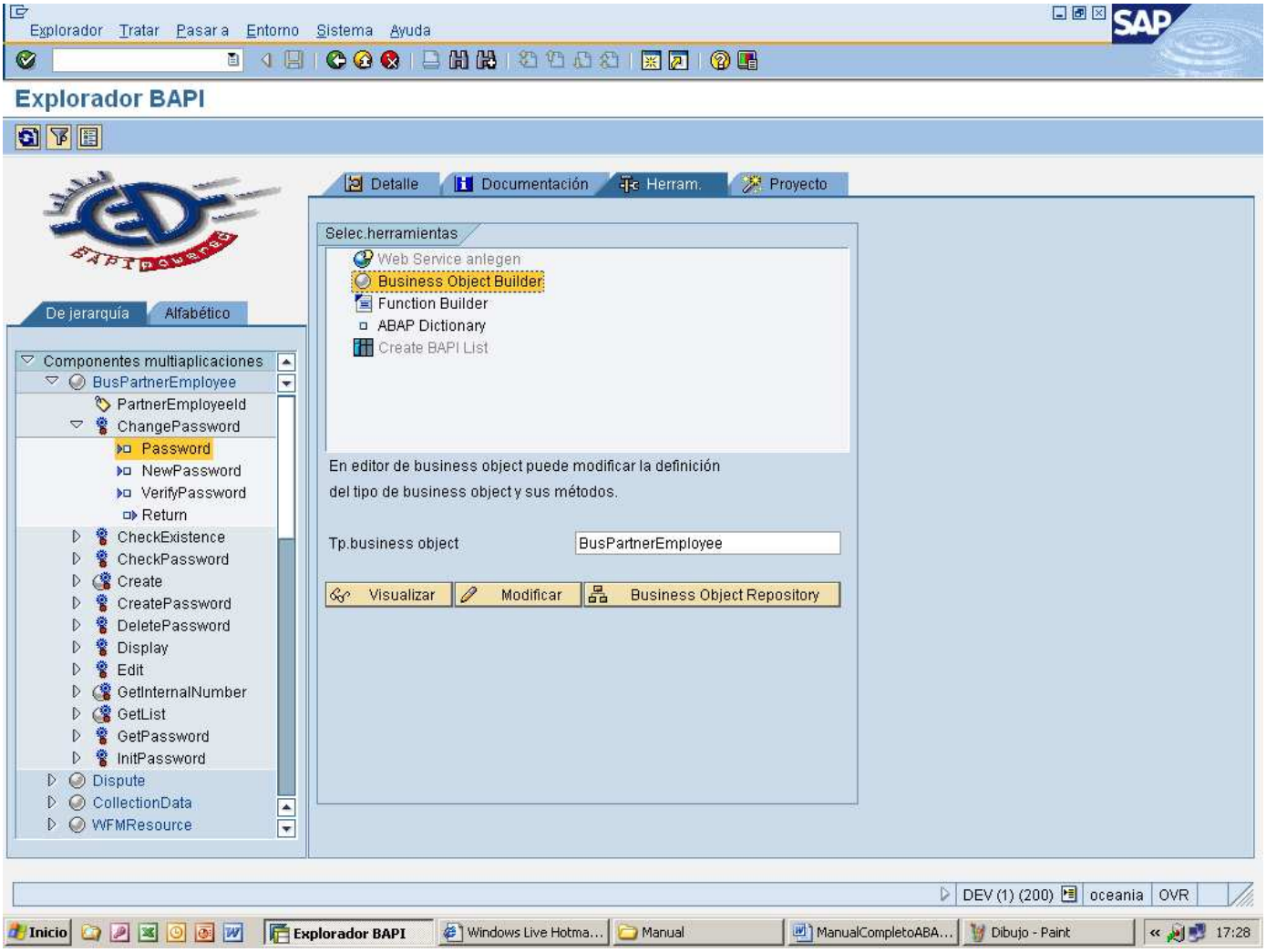
El programa que llama a la BAPI no requiere conocer cómo SAP almacena los datos o el funcionamiento interno de SAP; solo necesita conocer la interfaz de la BAPI. Esto implica conocer el significado y configuración de los parámetros y tablas que son exportadas e importadas desde la BAPI

SAP basa la tecnología actual de las BAPI en RFCs, existiendo algunas diferencias entre el módulo de una función normal RFC y un módulo de función BAPI RFC, tales como:

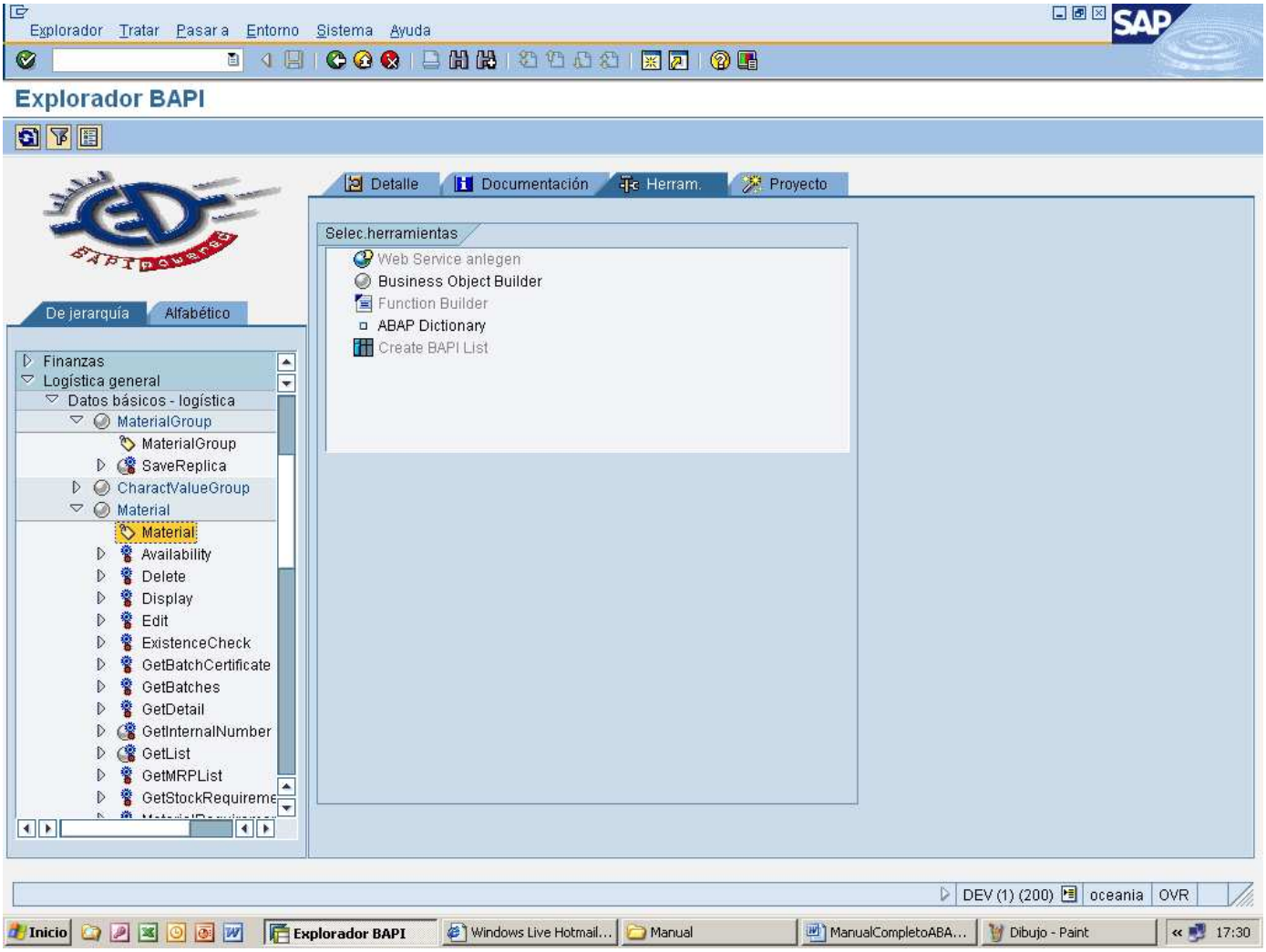
- Las BAPIs RFC se registran como métodos dentro del BOR.
- Las BAPIs RFC no pueden contener pantallas o módulos de diálogo.
- Las BAPIs RFC no contienen el comando ABAP **“COMMIT WORK”**
- Las BAPIs RFC deben seguir un estricto conjunto de reglas que los nombres de las propias BAPIs, los nombres de los parámetros, las estructuras y la documentación de éstas. Esto fuerza la consistencia entre todas las BAPIs.

13.2.3 Cómo se integran las BAPIs en SAP

Las BAPIs se integradas en SAP a través del BOR. Esta definición se puede encontrar en la transacción de código BAPI, encontrándonos con una pantalla similar a la siguiente:



Y expandiendo la estructura de arbol para los campos claves y los métodos podemos dirigir las asociaciones entre las BAPIs y sus parámetros asociados. Por ejemplo, si pinchamos, en la pantalla principal, en **Logística General → Datos básicos – logística → Maestro de materiales →Material** y expandiendo el árbol tanto para los campos clave como para los métodos podemos obtener la siguiente pantalla:



13.2.4 Ejemplo de RFC usando BAPIs

Los siguiente son ejemplos de cómo realizar llamadas a funciones remotas BAPI.Una de las ventajas de las BAPI RFCs es que ellas no necesitan ser usadas como los RFCs. Si queremos ejecutar una función que tiene construida una llamada BAPI, podemos usar esta directamente en el codigo ABAP.

Obteniendo datos del Maestro de Materiales.

Este ejemplo recupera datos para un número de material específico.

```
DATA: WS_MATNR    LIKE BAPIMATDET-MATNR,
      STR_RETVAL  LIKE BAPIMATDOA,
      BAPI_RC     LIKE BAPIRETRUN,
      WS_MESS(255).

WS_MATNR = 'TSTMAT'.
CALL FUNCTION 'BAPI_MATERIAL_GET_DEATIL'
DESTINATION 'PLUTO'
```

```
  EXPORTING
     MATERIAL             = WS_MATNR
  IMPORTING
     MATERIAL_GENERAL_DATA   = STR_RETVAL
     RETURN               = BAPI_RC
  EXCEPTIONS
  SYSTEM_FAILURE = 1
     MESSAGE WS_MESS
  COMMUNICATIONS_FAILURE = 2
     MESSAGE WS_MESS.

IF SY-SUBRC EQ 1.
    WRITE: /001 'RFC System Error',
         /001 WS_MESS.
    EXIT.
ENDIF.
IF SY-SUBRC EQ 2.
    WRITE: /001 'RFC Communications Error',
         /001 WS_MESS.
    EXIT.
ENDIF.
WRITE: /001 STR_RETVAL-MATNR,
      020 STR_RETVAL-SPART,
      025 STR_RETVAL-MAKTX.
```

Obteniendo una lista de materiales

```
DATA:INT_MATLST LIKE BAPIMATLST OCCURS 50 WITH HEADER LINE,
   INT_MATSEL LIKE BAPIMATRAM OCCURS 10 WITH HEADER LINE,
   INT_BAPI_RET LIKE BAPIRET2 OCCURS 10 WITH HEADER LINE,
   WS_MESS(255).

INT_MATSEL-LOW  = 'B*'.
INT_MATSEL-SIGN = 'I'.
INT_MATSEL-OPTION = 'CP'.
APPEND INT_MATSEL.

CALL FUNCTION 'BAPI_MATERIAL_GETLIST'
DESTINATION 'PLUTO'
```

```
  EXPORTING
      MAXROWS = '20'
  TABLES
      MATNRSELECTION = INT_MATSEL
      MATNRLIST      = INT_MATLST
      RETURN         = INT_BAPI_RET
  EXCEPTIONS
      SYSTEM_FAILURE = 1
         MESSAGE  WS_MESS
  CUMMINICATIONS_FAILURE = 2
         MESSAGE  WS_MESS.
IF SY-SUBRC EQ 1.
   WRITE: /001 'RFC System Error',
        /001 WS_MESS.
   EXIT.
ENDIF.
IF SY-SUBRC EQ 2.
   WRITE: /001 'RFC Communications Error',
        /001 WS_MESS.
   EXIT.
ENDIF.
LOOP AT INT_MATLST.
  WRITE: /001 INT_MATLST-MATNR,
       /020 INT_MATLST-MAKTX.
ENDLOOP.
LOOP AT INT_BAPI_RET.
  WRITE: /001 INT_BAPI_RET-MESSAGE.
ENDLOOP.
```

13.3 BATCH INPUTS.

13.3.1 Introducción.

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

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

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

Generalmente todos estos datos maestros (clientes, materiales, proveedores,...) ya están en el antiguo sistema informático. Por lo tanto lo ideal sería disponer de 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 SAP, como si fueran introducidos manualmente y uno por uno, por el usuario. Es decir realmente la técnica del Batch Input consiste en simular repetidamente un proceso online (transacción), durante un proceso Batch.

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

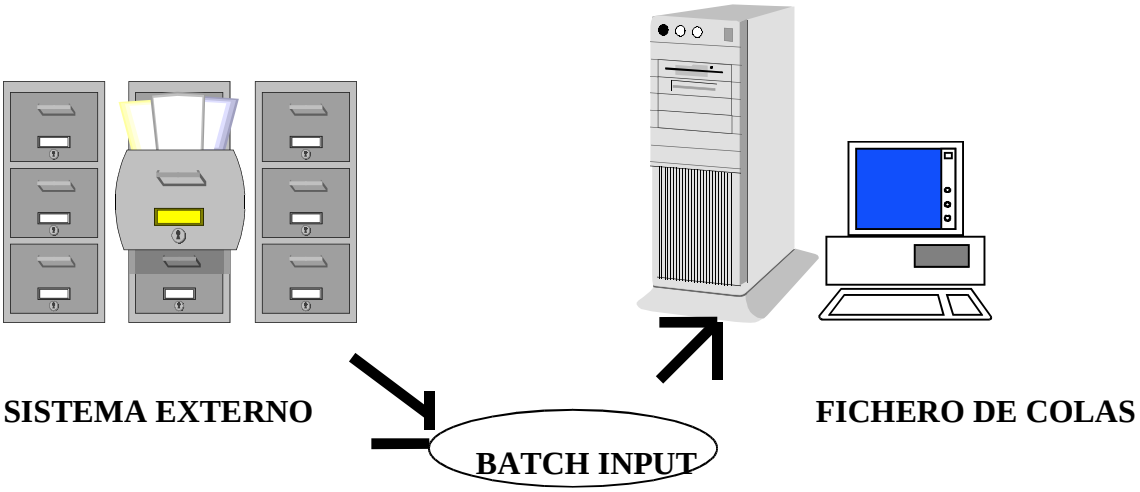
Fase de Generación : A partir de una fuente de información, como puede ser un fichero de entrada, donde estarán todos los datos que queremos cargar en SAP, se transformarán estos datos en un formato determinado, para almacenarlo en una estructura de SAP que llamaremos fichero de colas.

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

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

13.3.2 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:

13.3.2.1 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).

13.3.2.2 El programa Batch Input.

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

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

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

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

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

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

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

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.

13.3.2.3 El fichero de colas.

Todos los programas Batch Input graban entradas (sesiones) en el **fichero de colas**. Para posteriormente poder identificar cuál 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 ejecutarse por falta de recursos del sistema.

13.3.3 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 :
- N° Transacciones leídas.

  N° Transacciones procesadas con éxito.

  N° Transacciones erróneas.

Siempre que existan transacciones con errores se podrán reprocesar.

- **Procesamiento Visible :** Con este método se procesa cada una de las transacciones visualmente, es decir, el usuario va visualizando todas y cada una de las pantallas que hemos programado. El usuario únicamente debe ir pulsando <intro> para saltar de una pantalla a otra. Asimismo, si se cree conveniente, se permite modificar los valores de algún campo de la pantalla.

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

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

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

13.3.4 Consejos prácticos en la utilización de Batch Inputs.

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

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

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

Pasar a -> Análisis -> Juego de datos.

- Si se está ejecutando una transacción en modo Invisible, podemos ir viendo el 'Log' de proceso de las transacciones que se van ejecutando. Una utilidad práctica es, en el caso de un elevado número de transacciones, mirar el tiempo de proceso de una transacción y extrapolar este dato para todo el proceso, para tener una idea de la hora en la que finalizará el proceso.
- Antes de realizar un programa de Batch Input es aconsejable asegurarse de que SAP no disponga ya del mismo. Por ejemplo SAP nos ofrece bastantes Batch Inputs para carga de datos. Por ejemplo :

Carga de clientes.

Carga de proveedores.

Carga de documentos contables.

Carga de pedidos pendientes.

Carga de condiciones.

Carga de stocks...

- Nótese que entre la fase de generación y la fase de procesado, existe un tiempo indeterminado. Si este tiempo es muy grande, es posible que durante la fase de procesado se produzcan numerosos errores, ya que puede que haya cambiado el estado en el que se llevo a cabo la fase de generación.

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

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

13.3.5 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> OCCURS <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 **F1-F9** (Ayuda-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 : FSS1
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 **F1** sin soltar el botón del ratón, sobre el menú deseado.

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

Ejemplo :

```
CLEAR 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 de Batch Input, seguiremos los siguientes pasos en la codificación :

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

2.- Para cada transacción de la sesión :

2.a.- Llenaremos la tabla **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 cómo 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 dé ejecución a 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 la 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 immediata, 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 qué 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.

Cómo se utilizará la técnica del CALL TRANSACTION :

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

Usar la instrucción CALL TRANSACTION para llamar a la transacción.

```
CALL TRANSACTION <transaccion>
                 USING      <tabint>
                 MODE       <modo_ejec>
                 UPDATE     <tipo_actual>.
```

Donde :

<tabint> Tabla interna (con estructura BDCDATA).

<modo_ejec> Modo de ejecución.

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

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

'E' Ejecución visualizando sólo errores.

<tipo_actual> Tipo de actualización en la base de datos.

Puede ser : **'S'** Actualización Síncrona. (inmediata).

'A' Actualización Asíncrona. (la transacción a la que llamamos no espera a que se completen todos los updates).

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 qué ha ocurrido podemos ejecutar la instrucción:

```
MESSAGE ID SY-MSGID
    TYPE SY-MSGTY NUMBER SY-MSGNO
    WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
```

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

  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>.

14 FIELD SYMBOLS.

Cuando tenemos que procesar una variable, pero únicamente conocemos de qué 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>.
    WHILE <PUNTERO> EQ SPACE.
```

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

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

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

ASSIGN (<campo>) TO <«Field Symbol»>.

Ejemplo:

```
PARAMETERS: Dia7(10) value 'DOMINGO'
DATA CAMPO(10).
FIELD-SYMBOLS: <F>.
CONCATENATE 'DIA' '7' INTO CAMPO.
ASSIGN (CAMPO) TO <F>.
WRITE <F>.
```

Salida: 'DOMINGO'

También podríamos usar los Field Symbols con la instrucción SORT:

SORT …. <F>.

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

15 TRATAMIENTO DE FICHEROS.

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 APPENDING / 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.

Los datos deben de ser de **tipo char**, no importando que sean unicamente números.

Ejemplo:

```
DATA: BEGIN OF REC,
        LIFNR(16),
        BAHNS(16),
      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 pueden leer de/hasta 4 ficheros simultáneamente

- 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.

- Para subir un fichero desde Windows NT, utilizaremos la funcion GUI_UPLOAD.

```
CALL FUNCTION 'GUI_UPLOAD'
  EXPORTING
    filename             = l_path
    filetype             = 'ASC'
    has_field_separator  = 'X'
  TABLES
    data_tab             = i_file_table.
```

.

Donde l_path es el lugar donde se encuentra el fichero, y i_file_table es la tabla interna donde quedaran guardadas las lineas del fichero. En este caso, se trata de un fichero de texto ASCII.

- Para guardar un fichero en Windows NT, utilizaremos la funcion GUI_DOWNLOAD.

```
CALL FUNCTION 'GUI_DOWNLOAD'
  EXPORTING
    filename                = l_path
    write_field_separator   = 'X'
  TABLES
    data_tab                = i_file_table.
```

Donde l_path es el lugar donde se grabará el fichero, y i_file_table es la tabla interna donde estan los datos que queremos guardar en el fichero local. En este caso, se trata de un fichero de texto ASCII.

16 REPORTING INTERACTIVO.

16.1 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.

16.2 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 implementa 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 o ocultándola totalmente. Se podrán tener varias listas, cada una de ellas en ventanas distintas.

Para controlar el flujo del report interactivo, 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 interactivo 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**.

16.3 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 aparecerán los datos de esta nueva lista, solapándose parcialmente con la lista anterior.

En el evento AT LINE-SELECTION, utilizamos 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 necesitemos 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 3 : 'Variables del sistema para reporting interactivo'** para más información.

Ejemplo:

```
        TABLES LFA1.
GET LFA1.
        WRITE LFA1-LIFNR.
        WRITE LFA1-NAME1.
        HIDE LFA1-LIFNR.
        AT LINE-SELECTION.
                IF SY-LSIND = 1.
                        SELECT * FROM LFBK
                                WHERE LIFNR = LFA1-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	:	Cancel
P…	:	Funciones de Scroll.

Ejemplo :

```
        TABLES LFA1.
    GET LFA1.
        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 LFA1.
        WRITE LFA1-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 tecla de función se utilice para diferentes funciones.

16.4 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 es 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 (1ª. 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.

■ Manipulación 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 de ramificación.

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 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 <report> (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 quién tiene 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 una 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>.

17 PROGRAMACIÓN DE DIÁLOGO.

17.1 Introducción.

Hasta el momento, tanto en el capítulo anterior como en el 8, 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**.

Las transacciones son programas de diálogo con el usuario. Por ello, deben ofrecer :

- un interface de usuario agradable,
- chequeos de consistencia y formato para los datos de entrada que proceden del usuario,
- fácil corrección de los errores,
- acceso a la base de datos.

Básicamente una transacción esta formada por :

- **Dynpros** : conjuntos de pantallas y sus procesos lógicos asociados,
- **Module pool** : programa formado por módulos que controla el flujo de la transacción y realiza las acciones necesarias para cumplir la funcionalidad de ésta.

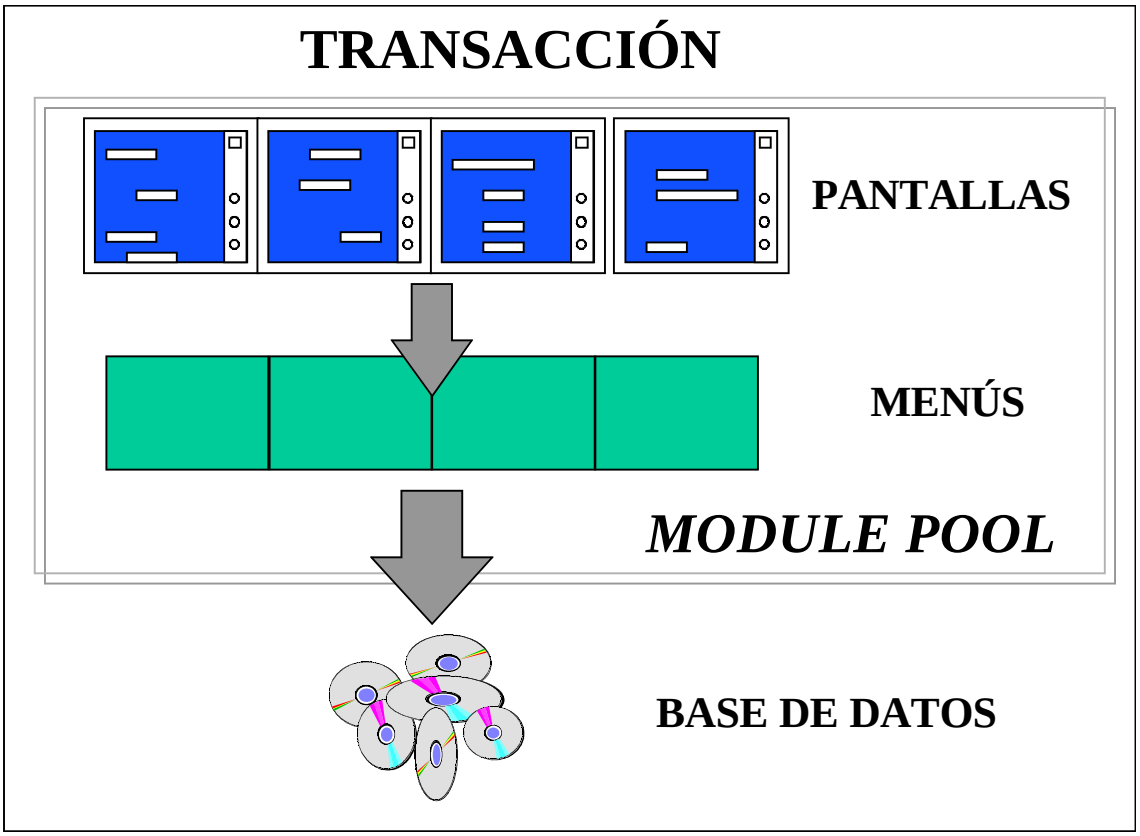
Los módulos son llamados desde los procesos lógicos (**Flow logic**) de las pantallas de la transacción, estableciendo una relación continua

MODULE POOL - FLOW LOGIC

En base a lo anterior, 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**).

17.2 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** (transacción SE93**)**.

Herramientas -> Case -> Desarrollo -> Transacciones.

Herramientas -> Workbench ABAP/4 -> Desarrollo -> Más herramientas -> Transacciones. (SAP R/3 3.xx)

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**), indicando que el programa es de tipo M.

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.

18 DISEÑO DE MENÚS (MENU PAINTER)

18.1 Introducción.

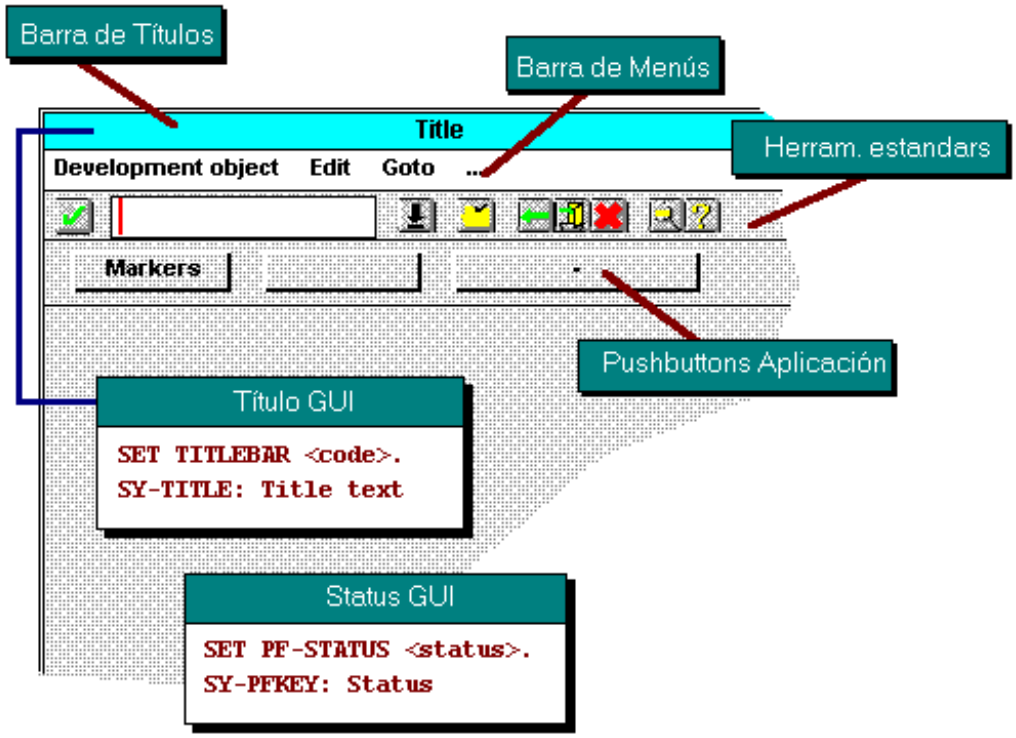
Con el **Menu Painter** diseñaremos las superficies **GUI**, (Grafical User Interface) , sobre las que correrán las transacciones SAP.

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 una transacción .

Para iniciar el Menú Printer, seleccionar : **Herramientas -> 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…).

18.2 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, el sistema añadirá automáticamente **System** y **Help.**

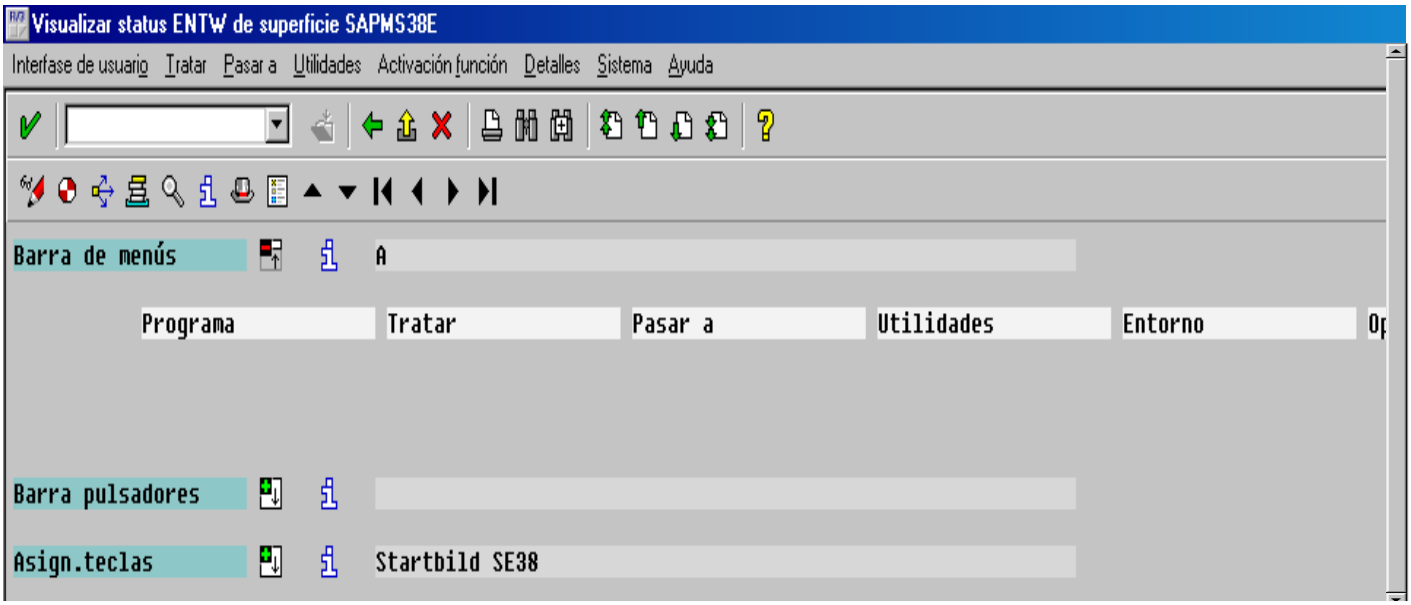
Cada menú puede tener hasta 15 entradas. Cada una de las cuales puede ser otro menú en cascada o una función.

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-Click podemos desarrollar las opciones del submenú.



18.3 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.

18.4 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 **19.3**).

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.

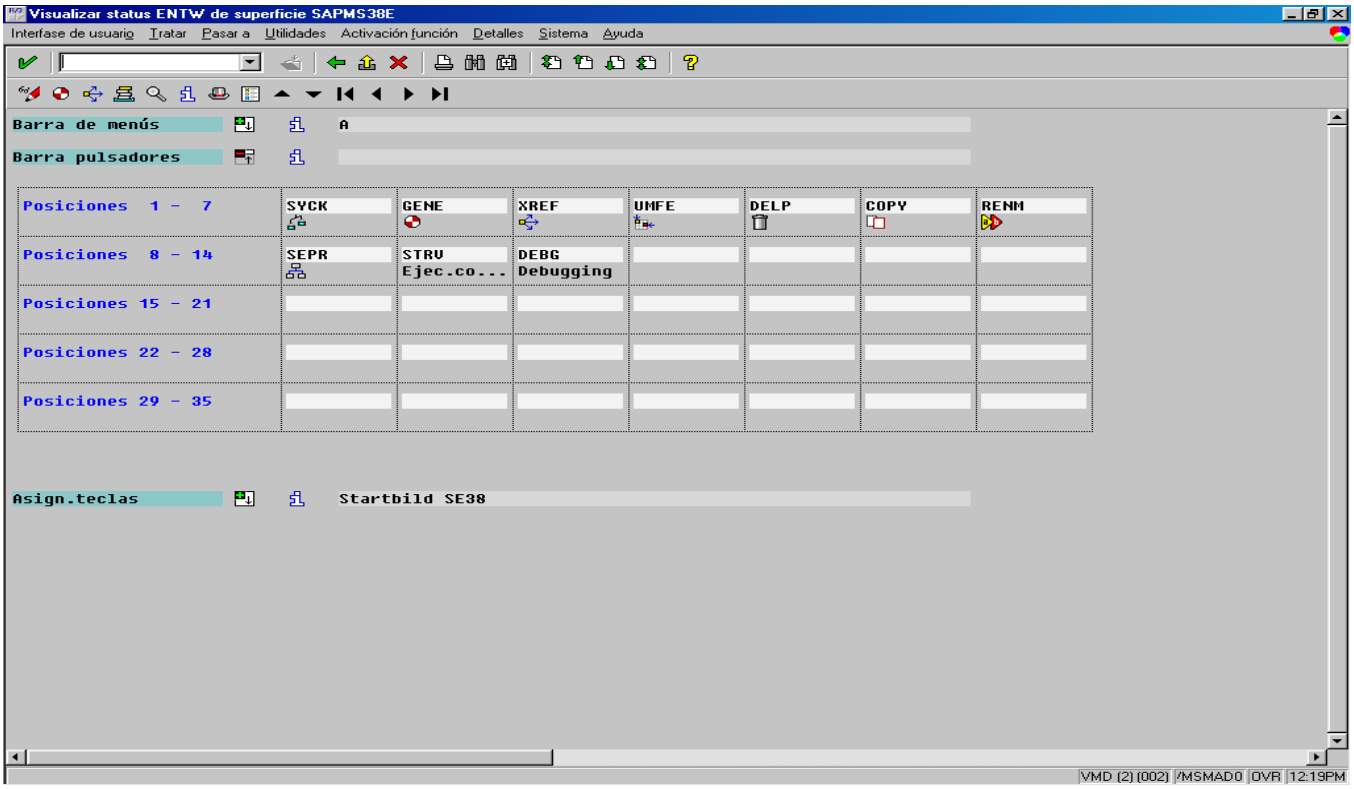
No será necesario definir las funciones de la barra de herramientas estándar, '**Standard Toolbar**'.

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.



18.5 Otras utilidades del Menú painter.

18.5.1 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 '**Activación Función**'**.**

18.5.2 'FastPaths'.

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

Podemos mantener 'Fastpaths' seleccionando :

Pasar a -> Otros destinos -> Acceso directo.

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

18.5.3 Títulos de Menú.

Es posible mantener distintos títulos para un menú.

Pasar a -> Lista títulos.

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 <var1> <var2> ….

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

18.5.4 Prueba, chequeo y generación de Status.

Podemos probar el Status simulando la ejecución de la interface con :

Interfase de usuario -> EfectuarTest Status , y 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: **Interfase -> Verif. Syntax.**. Posteriormente realizaremos un proceso de generación de la interface que incluye el chequeo y la grabación de la misma.

18.6 Menús 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.

Tools…Development -> Other Tools -> Area Menus.

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**'.

19 DISEÑO DE PANTALLAS (SCREEN PAINTER)

19.1 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** (`Dynamic 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 X11/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

19.2 Diseño de pantallas.

19.2.1 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 numero 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.

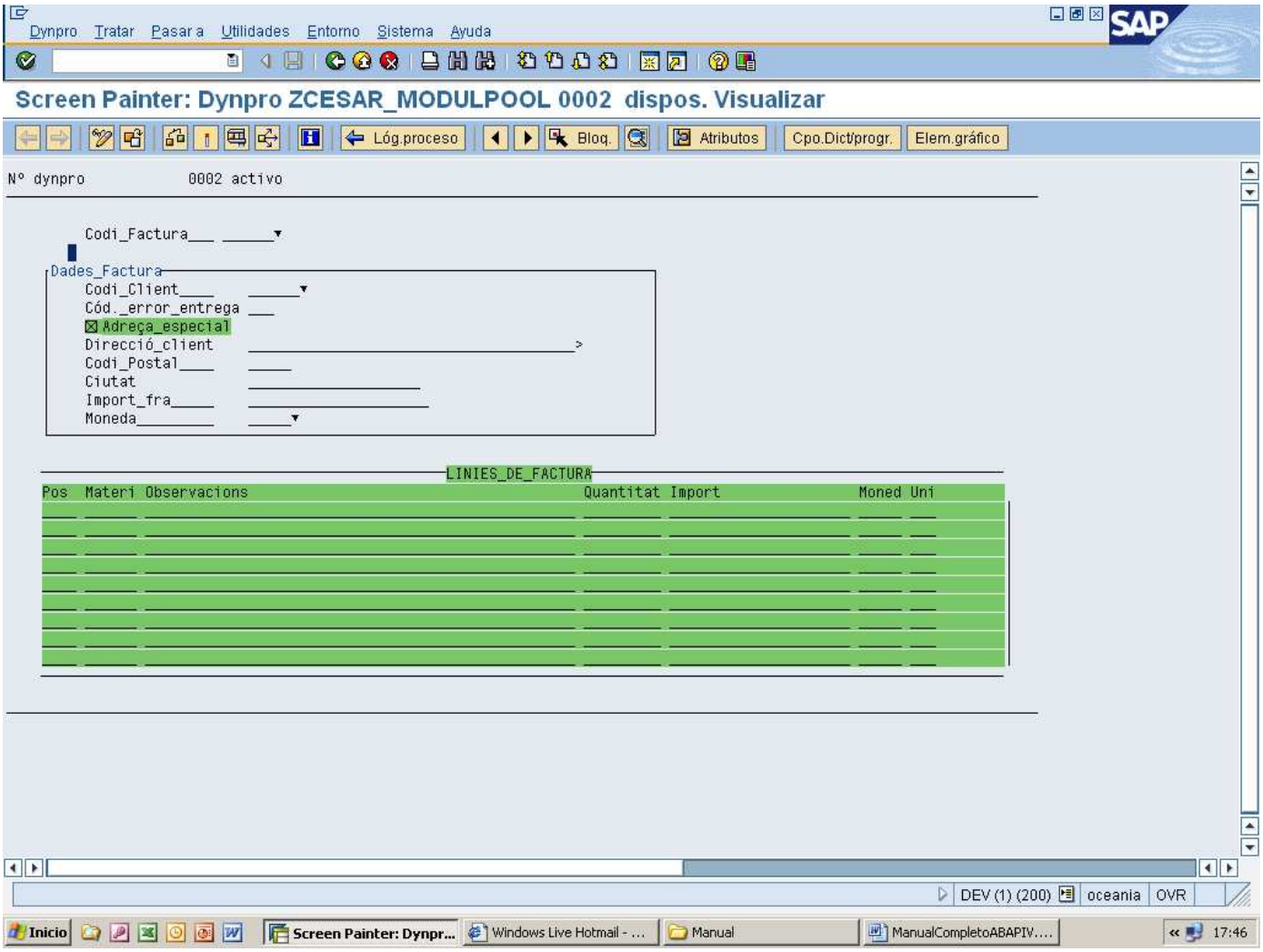
Tipo de pantalla (normal, de selección, modal, Subscreen).

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 que 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 estamos diseñando.

19.2.2 Creando objetos en la 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.
- **Subscreens :** 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,** que 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.

19.2.3 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 datos ('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.

19.2.4 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 pasamos por encima de un icono con el ratón.
- **Line** : Especifica la línea donde el elemento aparecerá. El sistema completa este valor automáticamente.
- **Cl** : 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 pushbuttons.
- **FctType** : Especifica el tipo de evento en el cual el campo será tratado.
- **Ltype** : Tipo de steep 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.
- **Lcnt** : Número de líneas de un step loop.
- **Groups** : Identifica grupos de modificación para poder modificar varios campos simultáneamente. Podemos asignar un campo a varios (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 SET/GET. (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** : Visualiza un campo de salida en un tamaño fijo (no proporcional). Sólo se puede utilizar en campos Output only.
- **Bright** : Visualiza un campo en color intenso.
- **Invisible** : Oculta un campo.
- **2-dimens** : Visualiza un campo en dos dimensiones en vez de en tres.

19.3 Lógica de proceso de una pantalla.

19.3.1 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 logic.**

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:

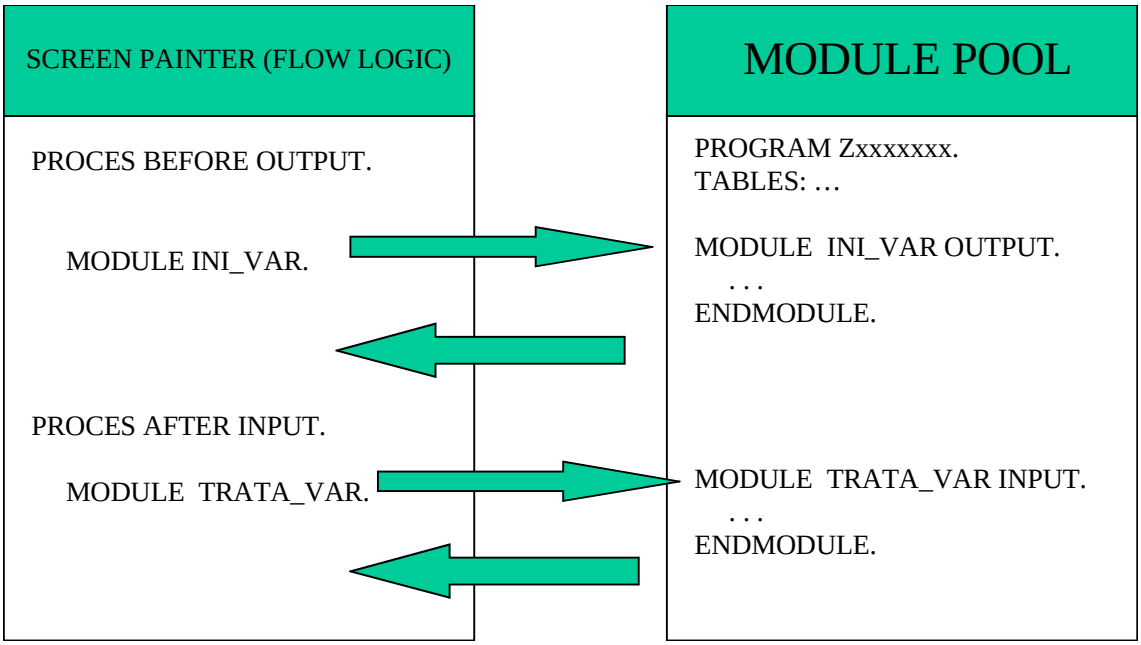
PROCESS BEFORE OUTPUT (PBO).

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 llaman a los módulos del module pool que se encargan de esta tarea.



19.3.2 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_modulo> 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.

Una vez que se conoce la estructura de esta tabla, en el programa POOL en el modulo STATUS_XXXX_OUTPUT, se escribe lo siguiente:

```
loop at screen.
 if group1 = 'xxx'          Este valor se asignó cuando se creó la pantalla
      screen-active  = 1.   Indica que el campo esta activo
      screen-output = 1.    Indica que el campo es solo de salida
      screen-input   = 0.   Indica que el campo no es de salida.
        ..
      modify screen.
 endif.
endloop.
```

- 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 que le hemos dado al área de subscreen en el Screen Painter.

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

<screen> : 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 que campo está situado el cursor. Para hacer esto utilizamos :

GET CURSOR FIELD <var_campo>.

19.3.3 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.

En este punto de la ejecución, es cuando se produce la transferencia de datos desde la pantalla al Module Pool. Se transfieren todos los datos a la vez excepto en algunos casos que veremos posteriormente.

En el PAI la llamada a los módulos se realiza del mismo modo que en el PBO, es decir ,

MODULE <nombre_modulo>.

La diferencia con el PBO aparece en el Module Pool donde los módulos que se llaman desde el PAI se definen como :

MODULE <nombre_modulo> INPUT.

En el PAI la llamada a una subscreen se realiza mediante la orden :

CALL SUBSCREEN <area>.

19.3.3.1 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 Menu 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 utilizamos:

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>’)

(‘<valor1>’, ‘<valor2>’,…NOT ‘<valorn>’)

(BETWEEN ‘<valor1>’ AND ‘<valor2>’)

(NOT BETWEEN ‘<valor1>’ 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 a 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 a 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.
```

En el caso del CHAIN, tambien existen opciones de condicionamiento en las llamadas a los módulos de validación :

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

***Observación* :** Como excepción a la transferencia de datos entre la pantalla y el programa, el contenido de los campos mencionados en algún comando FIELD no se transfiere hasta que no se llega a dicho comando FIELD en la ejecución del proceso lógico de la pantalla. Además, si un campo aparece en más de un comando FIELD sólo se transfiere en el primero de ellos, considerando el orden de ejecución.

Este tiempo de transferencia es importante tenerlo en cuenta, para no utilizar un campo cuyo contenido todavía no ha sido transferido.

19.3.3.2 Ejecución del PAI después de un error de validación

Cuando se detecta un error en una validación, el sistema vuelve a mostrar los campos involucrados en dicho error para aceptar nuevas entradas. Después de permitir nuevos datos, el sistema vuelve a ejecutar el proceso de PAI. Pero, en realidad, no empieza a ejecutarlo desde el principio en todos los casos, sino que el sistema determina dónde se debe empezar a ejecutar, siguiendo los siguientes pasos :

1. Determina qué campos en el proceso lógico :

- están especificados en el comando CHAIN o FIELD que ha dado el error, y
- fueron modificados por el usuario en la última visualización de la pantalla.

2. Busca en el evento PAI el primero de los comandos FIELD que menciona alguno de los campos del paso 1.
3. Comienza el proceso del PAI desde este comando FIELD o , en el caso de ser un CHAIN, al comienzo del comando CHAIN.

19.3.3.3 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 que función a 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 modulo 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. Dicho módulo aparece siempre por defecto en el Flow Logic de manera comentada y aparece bajo el nombre de :

MODULE USER_COMMAND_<no.pantalla>

Ejemplo :

LOGICA DE PROCESO	**MODULE POOL**
PROCESS BEFORE OUTPUT.	MODULE OK_CODE.

```
…                                          CASE OK-CODE.
                                            WHEN 'BACK'.
                                                 SET SCREEN 0.
PROCESS AFTER INPUT.                             LEAVE SCREEN.
                                            WHEN 'CHNG'.
  FIELD …                                        PERFORM CHANGE_DATA.
  MODULE …                                  WHEN 'DISP'.
  MODULE …                                       PERFORM DISPLAY_DOC.
                                            WHEN 'COPY'.
                                                 CALL SCREEN 0200.
  MODULE OK_CODE.                           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.

19.3.3.4 Procesando Step loops.

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

Por ejemplo :

Client	Name	City	Std.currency	Changed on
000	SAP AG	Walldorf	DEM	
001	Auslieferungsmandant R11	Kundstadt	ECU	
002	RIVERLAND TEST CLIENT	BARCELONA	ESP	11.04.1996
066	EarlyWatch	Walldorf	DEM	

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 steep 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 ENDLOOP, 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 <L1> 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.

El proceso en el PAI es :

1. Se posiciona una fila de la tabla interna en la cabecera.
2. Todos los campos de la pantalla son transferidos a los campos del programa ABAP/4 que tienen el mismo nombre.
3. Se ejecutan todas las acciones del loop para la fila de la tabla interna.

Observación : Si se quiere modificar la tabla interna con los contenidos de la cabecera se debe utilizar la instrucción MODIFY porque el sistema no lo hace automáticamente

Ejemplo :

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

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

***Nota* :** La utilización de tablas es similar al proceso de step loops pero con unas pequeñas variaciones :

- Se debe declarar una estructura de control para la tabla que se ha declarado en la pantalla. Para ello, se utiliza la sentencia CONTROLS.

  Ejemplo :

  CONTROLS: Tabla TYPE TABLEVIEW USING SCREEN <_>

Esta estructura declarada nos servirá para modificar atributos de la tabla de la pantalla. Los atributos que podemos modificar son los especificados en la estructura TABLEVIEW.

- En la llamada a la sentencia del LOOP se debe incluir el parámetro adicional WITH CONTROL , es decir :

LOOP WITH CONTROL <tabla_control>.

.....

ENDLOOP .

o,

LOOP AT <tabla_interna> WITH CONTROL <tabla_control>.

.....

ENDLOOP.

19.3.4 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> <línea_inicio>

  ENDING AT <columna_fin> <línea_fin>.

Con esta ultima 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 un 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 terminar el proceso de una pantalla e ir directamente a ejecutar otra, es usando la instrucción **LEAVE**.

LEAVE TO SCREEN < no._pantalla>.

o

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. o **SET SCREEN 0.**

LEAVE SCREEN.

De este modo, el proceso continuaría en la pantalla que hizo la llamada CALL SCREEN y se empezaría a ejecutar la instrucción siguiente a ésta.

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.

LEAVE TO LIST-PROCESSING:

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

LEAVE TO LIST-PROCESSING.

Cuando se utiliza esta orden el sistema lleva a cabo dos operaciones :

- cambiar a modo listado, con lo cual detrás de esta orden podremos utilizar todas las sentencias y eventos disponibles para reports,
- visualizar el listado como siguiente pantalla a la actual. Dependiendo de lo que necesitemos en cada caso, podremos visualizar ambas pantallas o inhibir la pantalla actual para visualizar sólo el listado

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** o **LEAVE LIST-PROCESSING**.. El sistema automáticamente volverá al PBO de la pantalla que estaba procesando y que hizo la llamada **LEAVE TO LIST-PROCESSING**.

Si queremos volver a otra pantalla distinta a la que se estaba procesando, se utiliza la sentencia :

LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN <pantalla>

Utilizando esta opción si el sistema alcanza un LEAVE SCREEN, el sistema comenzará a ejecutar el PBO de la pantalla indicada.

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

La sentencia LEAVE TO LIST-PROCESSING puede utilizarse tanto en el PAI como en el PBO. Si se codifica al final del PAI de la pantalla actual entonces se visualizan la pantalla actual y el listado. Si se codifica en el PBO y despues codificamos un LEAVE SCREN el sistema sólo visualizará la pantalla del listado.

SUBMIT:

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 module 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.

Nota : en la variable **answer** se devuelven los valores 'J' si la respuesta es afirmativa y 'N' si la respuesta es negativa.

■ **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 -GET respectivamente :

SET PARAMETER ID 'param' FIELD <campo>.

GET PARAMETER ID 'param' FIELD <campo>.

19.3.5 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. Utilizaremos el ROLLBACK 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.

Observación : No se puede ejecutar más de una vez la misma función con los mismos parámetros.

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 la función comienza a ejecutarse cuando termina la ejecución del FORM, es decir, no se necesita otra sentencia COMMIT WORK para lanzar la función, nos vale con la que lanzó la ejecución del FORM. Los valores de actualización son los que existan previamente a la actualización física.

Ejemplo :

```
A = 1.
PERFORM  ZFORM ON COMMIT.
A = 2.
COMMIT WORK.
```

```
FORM ZFORM.
      CALL FUNCTION 'UPD' IN UPDATE TASK
            EXPORTING PAR =A .
ENDFORM.
```

En este ejemplo la función se llama con el valor de A = 2.

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.

19.3.6 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. Para definir un objeto de bloqueo, especificamos las tablas que están incluidas en él y las claves primarias de dichas tablas. Las claves primarias forman el argumento de bloqueo para el objeto.

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.

Los parámetros de las funciones son :

- Exporting :
  - **<arg>** y **<x _arg>** : (ENQUEUE, DEQUEUE)

    existe un parámetro de estos tipos para cada campo del argumento. En la llamada de la función se les asigna los valores correspondientes. Si no se les quiere dar valor se omiten y si se les quiere dar el valor inicial se realiza la llamada con x_arg = 'X'.
  - **_SCOPE** : (ENQUEUE) sirve para definir el alcance del bloqueo.
  - **_WAIT** : (ENQUEUE) indica si el programa debe o no esperar en el caso de que el objeto se encuentre bloqueado por otro ususario.
- Excepciones : (ENQUEUE)
  - **FOREIGN_LOCK**: Si el objeto está bloqueado por otro usuario.
  - **SYSTEM_FAILURE**: Error de sistema.

Utilizando las excepciones podremos comprobar si el bloqueo se ha llevado a cabo correctamente o no.

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

19.3.7 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 F1 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>.

Dentro de este evento es normal el utilizar las funciones

DYNP_VALUES_READ

DYNP_VALUES_UPDATE

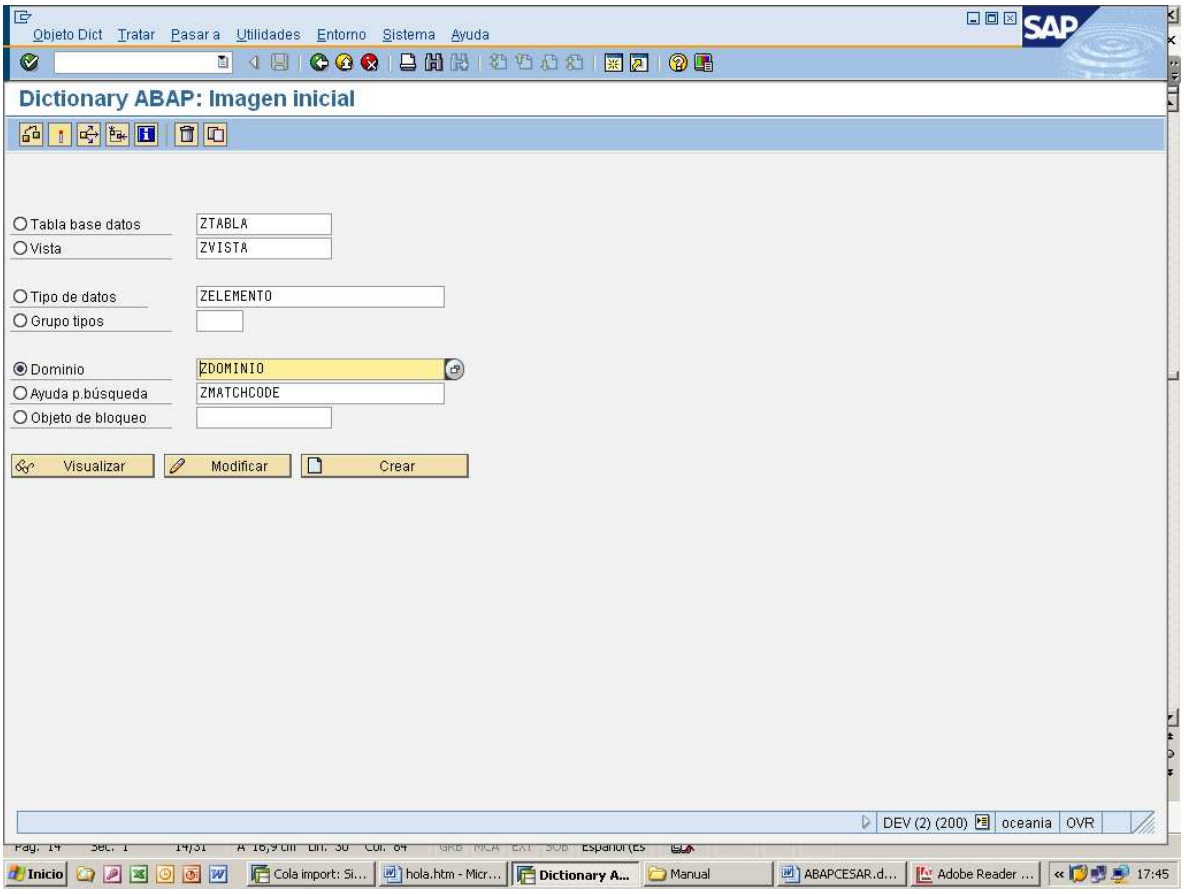
que sirven para leer y modificar los datos de un campo antes de que se hayan transferido los datos con el evento PAI.

Nota : SAP ofrece una serie de funciones estandares que nos sirven para visualizar ayuda y posibles valores. Dichas funciones se encuentran en el grupo de funciones SHL3 y se utilizarían conjuntamente con las especificadas anteriormente.

20 DICCIONARIO DE DATOS. Creación de nuevos objetos.

20.1. El proceso de creación de una tabla.

Para crear una tabla primero hemos de crear un dominio, seguidamente un elemento de datos para ese dominio u otro que exista y por último la tabla que tendrán una serie de elementos ya existente o que los hayamos creado antes. Además de explicar como se realiza un dominio, un elemento de datos y una tabla, también explicaremos como las podemos visualizar, modificar y otras operaciones diversas.



Transacción SE11, donde podemos crear tablas, vistas, elementos y dominios

DOMINIO

Introducir el nombre del Dominio en el campo Dominio que vamos a crear, visualizar, modificar o buscar. Pulsar el botón que deseamos.

En esta pantalla introduciremos los valores que le daremos al dominio. Hay que poner una descripción breve de lo que hace o para qué sirve ese dominio.

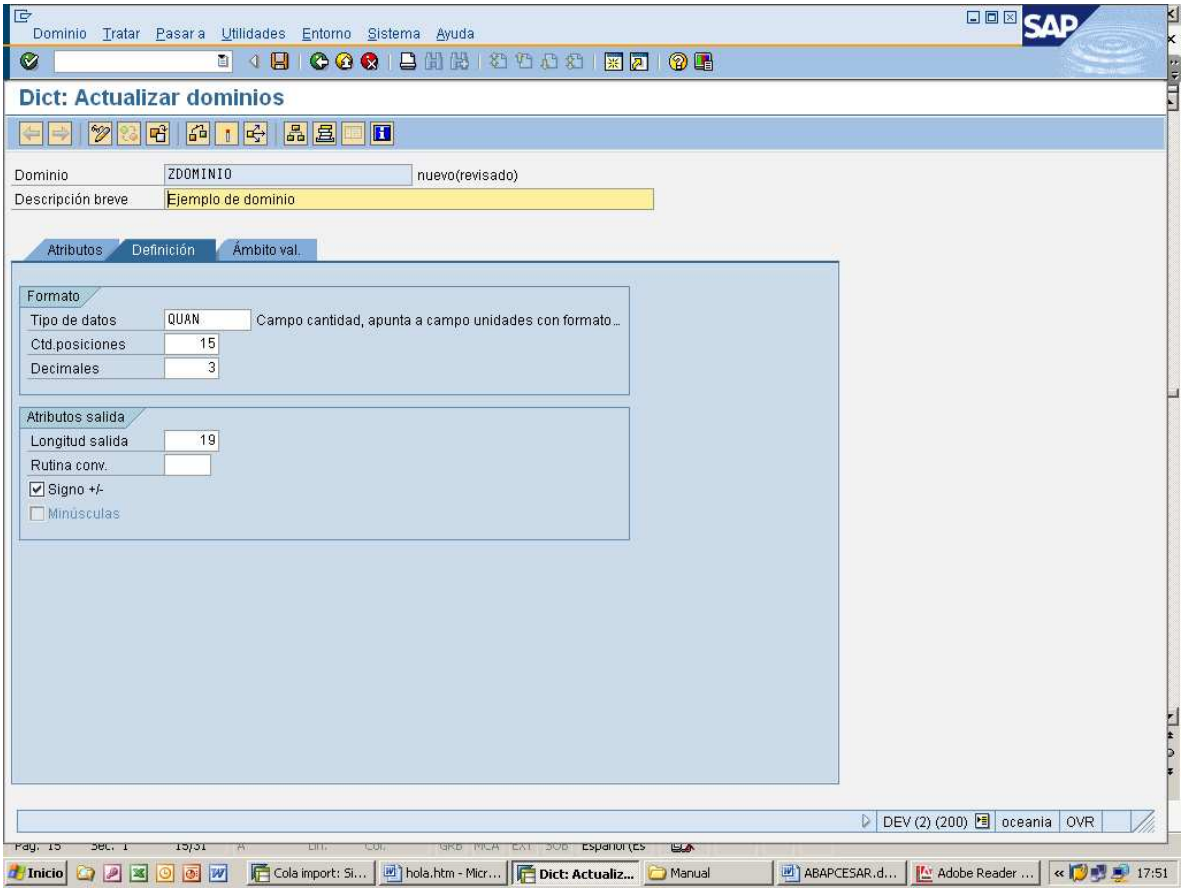
En “Formato”: en “tipo de datos”, haciendo doble clic nos saldrán los tipos de datos disponibles. Y “longitud” será la longitud que le demos.

"Valores permitidos": será la tabla de verificación. La tabla de verificación es una tabla en la que hay unos valores. Cada vez que entramos un dato en un campo que tenga una tabla de verificación irá a esa tabla para comprobar su validez. También podemos introducir límites para un campo.

Para introducir valores fijos vamos al recuadro de "valores permitidos" y pulsamos al botón de "valores fijos" o hacemos doble clic en el campo. Otra forma de llegar es en el menú "Pasar a (o F8)", "Valores fijos".

Como vemos, podemos introducir valores fijos (tantos como queramos) o límites inferior o superior (tantos como queramos). De vuelta a la pantalla principal de dominios:

En "Propiedades de salida", "Longitud de campo": qué longitud saldrá cuando visualicemos el contenido del campo. En un dominio también podemos poner valores fijos. Cuando introduzcamos algún valor en un campo y no concuerde con los valores preestablecidos el sistema avisará que el dato introducido no es correcto.



Cuando grabemos por primera vez nos preguntará por la Clase de desarrollo. Esta clase nos la da la empresa donde estemos y sirve para el transporte a producción. Para practicar se deja en blanco.

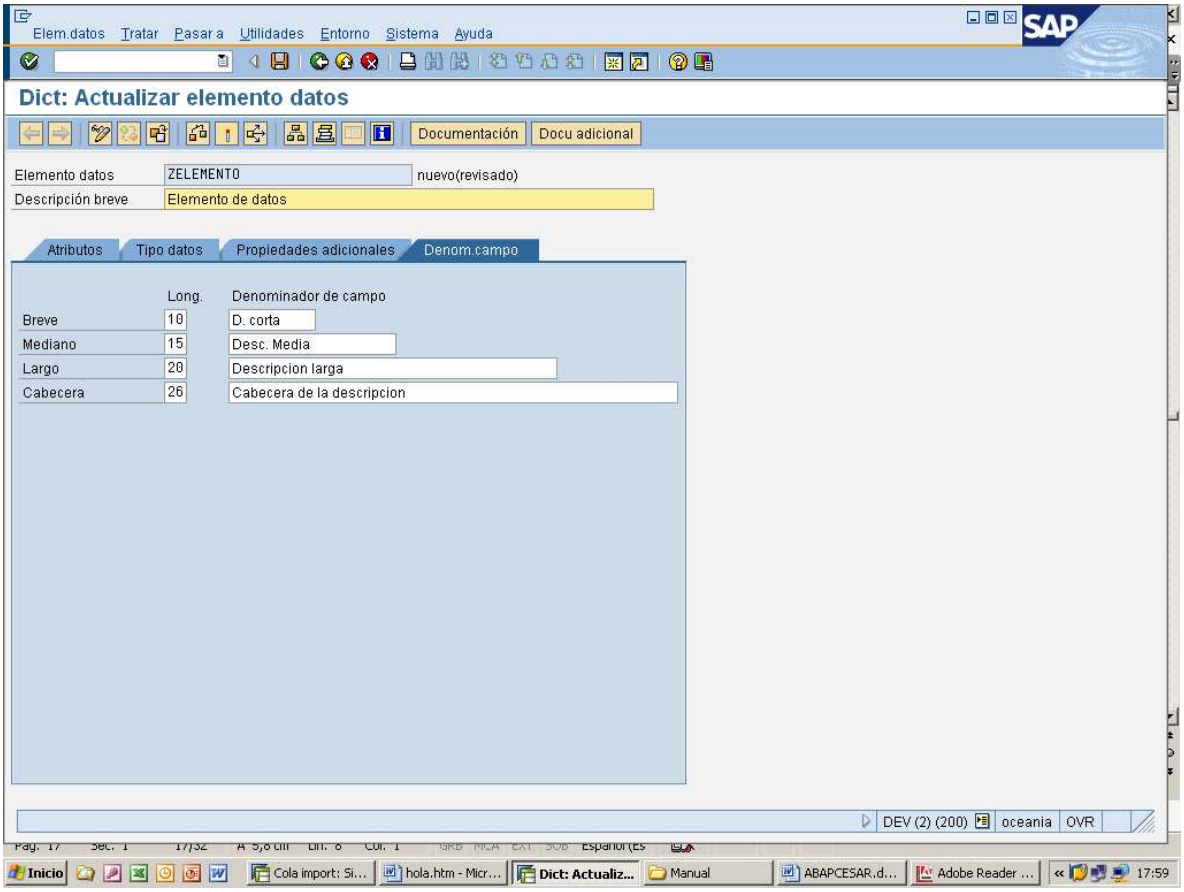
Para practicar la grabaremos como Objeto local (Botón que está abajo) nos pondrá como clase de desarrollo '$TMP'. Al igual que cualquier otro objeto, hay que verificar y activar el dominio para que pueda ser utilizado en un elemento de datos o tabla.

ELEMENTO DE DATOS

Introducir el nombre del Elemento de datos que vamos a crear, visualizar o modificar en el campo Tipo de datos. Pulsar el botón que deseamos y elegir el tipo de datos 'Elemento de datos'.

Un elemento de datos es información semántica o técnica de un campo. También un elemento puede estar en más de un campo (distinto o igual).

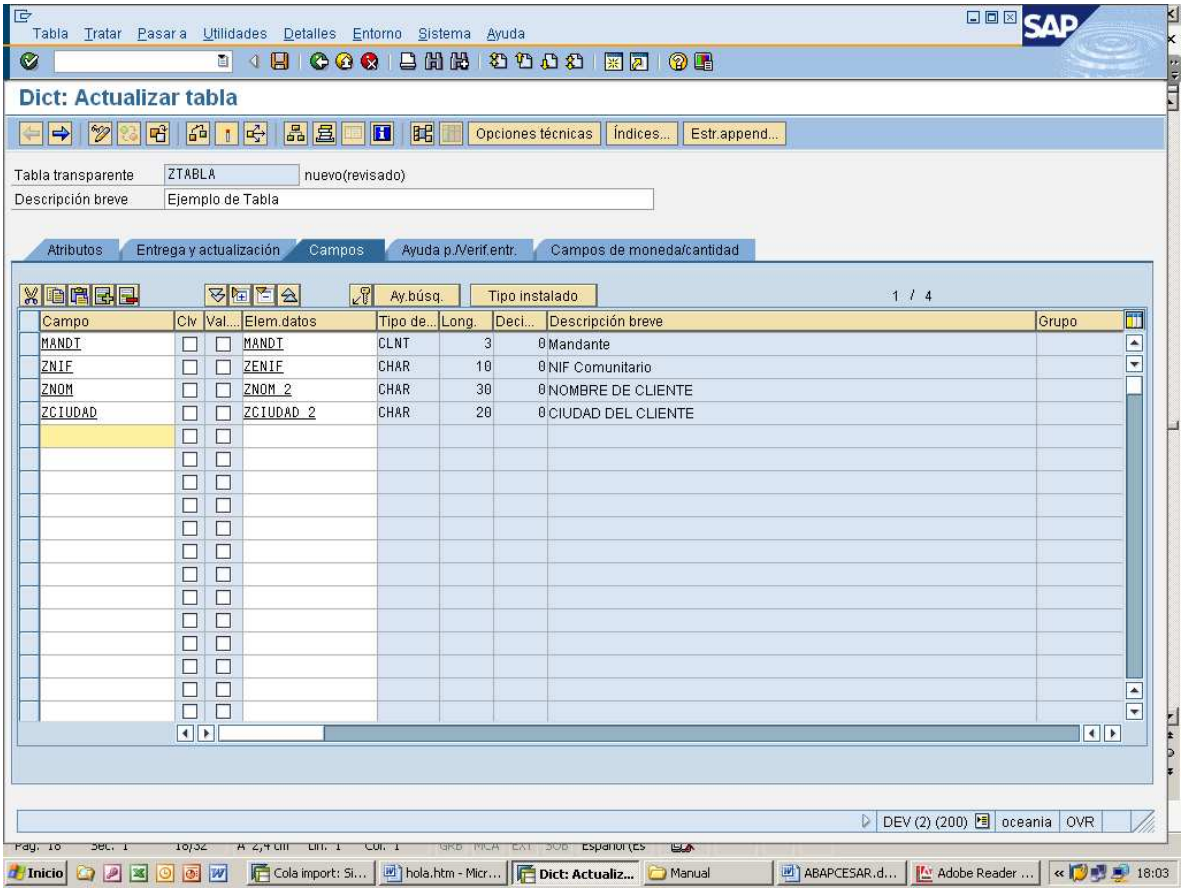
Hay que introducir una descripción para ese Elemento de datos. En Dominio introduciremos el dominio que tendrá (es un campo obligatorio), si no lo sabemos, ponemos el cursor en el campo y pulsando F4 podremos buscar el dominio que queramos utilizar.



Después pondremos la descripción breve, media o larga del campo, lo grabaremos y activaremos igual que el dominio.

CREAR, MODIFICAR O VISUALIZAR UNA TABLA

Desde la pantalla de DICTIONARY, introducir el nombre de la tabla que vamos a crear, visualizar o modificar. Pulsar el botón de "Objeto Dictionary" llamado "Tabla". Pulsar el botón de lo que deseamos hacer, y nos saldrá la pantalla siguiente:



Los campos que tengan fondo blanco y el carácter '?' son de obligada introducción. En la clase de entrega haremos clic y le pulsamos F4 para seleccionamos el tipo de clase de entrega.

También activaremos el Check Box que pone "Perm.actual.tablas" para que se puedan introducir datos y modificar estos. Este checkbox nos permitirá luego modificar los datos de la tabla directamente en ella. Si no queremos que los usuarios tengan este privilegio, no marcaremos dicho campo.

En la pestaña "campos", introduciremos todos los campos de la tabla. Hay que introducir uno obligatoriamente. Y uno ha de ser el campo clave. Se activa pulsando en el check-button que está debajo de la palabra "Clv".

Después aparecen los Elementos de datos que a su vez tienen Dominios (Estos dominios se asignan automáticamente al crear el elemento de datos). En "Elementos de datos" podemos introducir un campo que ya sepamos o posicionandonos sobre el campo y pulsando F4 podemos realizar una búsqueda de todos los elementos que haya o una condición de búsqueda. Cuando lo hayamos puesto pulsaremos ENTER para que SAP coga ese elemento de datos.

El boton 'Tipo Instalado' nos permite introducir el tipo de dato sin indicarle el elemento de datos.

MANDANTE

En el primer campo se suele poner, por no decir siempre, el campo "Mandt". Este campo sirve restringir el acceso de un usuario o usuarios a una determinada tabla o tablas. En una empresa puede haber más de una mandante. Para poder declararlo donde pone "nombre de campo" pondremos "Mandt", en

elementos de datos también escribiremos “Mandt”, pulsaremos ENTER para que lo coga y también le diremos que es un campo clave.

TABLA DE VERIFICACION

La tabla de verificación son tablas en las que hay unos valores. Cada vez que entramos un dato en un campo que tenga una tabla de verificación irá a esa tabla para comprobar su validez. Cuando en el campo que pone “Tab Verif.” sale un ‘*’ indica que puede haber un campo de verificación. Para poder asociar un campo a una tabla de verificación hay que realizar una clave foránea. Más adelante veremos un ejemplo de cómo se asocia a un campo una tabla de verificación y cómo se hace una clave foránea.

INDICES

Los índices sirven para crear campos claves en nuestra tabla. Lo que se consigue es mejorar la velocidad cuando buscamos por el campo clave de la tabla. Más adelante veremos un ejemplo completo de cómo se hace un pequeño índice.

CLAVES FORANEAS

Las claves foráneas nos permiten relacionar una o varias tablas a través de un campo clave. Una tabla de verificación para un campo ejerce un control sobre los datos que introducimos por pantalla, impidiendo meter datos en ese campo que no existan en la tabla de verificación.

POR ULTIMO

Después de poner todos los datos que queramos, tenemos que grabar, verificar y activar.

20.2. 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 **Pasar a -> Claves externas**.
- 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.

20.3. Otras posibilidades en la creación de tablas.

- Si hemos marcado la posibilidad de poder mantener el contenido de la tabla desde la transacción SM30, será necesario que ejecutemos un proceso de generación de dicho mantenimiento.

  Para hacerlo deberemos seleccionar : **Entorno -> Gen. act. tablas .**

  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 : **Pasar a -> Indices**.

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

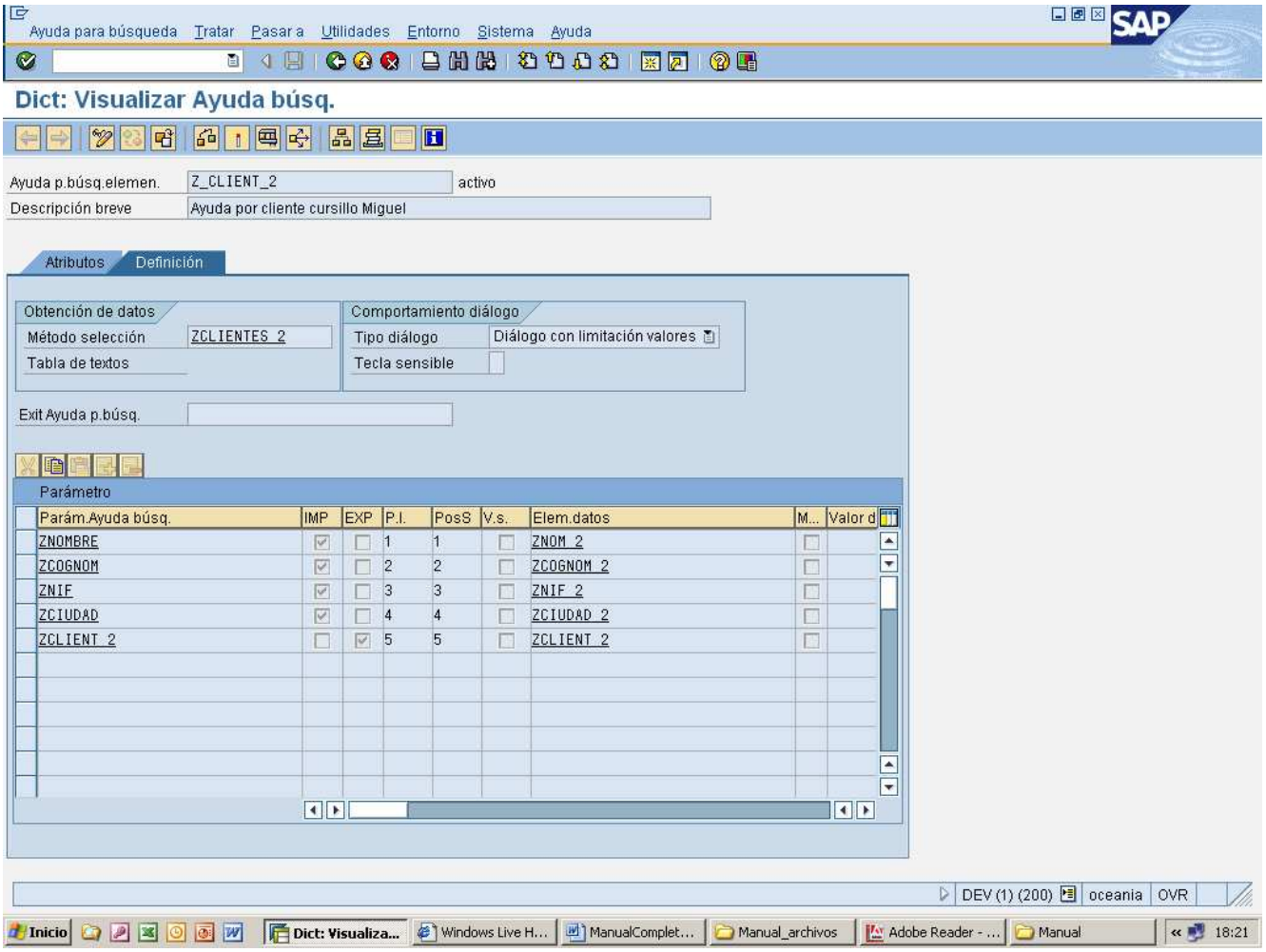
20.4 OBJETOS MATCHCODE

Un objeto matchcode es la tipica ayuda de busqueda que sale en aquellos campos donde podemos seleccionar un valor o varios valores.

Para crearlo tenemos que ir al Dictionary y ahí seleccionar “Ayuda p.busqueda”, después escribiremos el nombre que le daremos (máximo de 4 letras), y pulsaremos el botón “crear”.

Antes de todo hemos de poner la descripción breve y la tabla primaria en esta tabla donde se van a coger los campos a utilizar en el matchcode. Cuando hayamos introducido estos campos obligatorios es la hora de escoger que campos vamos a utilizar.

En la tabla iremos indicando los campos que queremos mostrar en pantalla para la ayuda de busqueda, asi como los campos que queremos elegir.



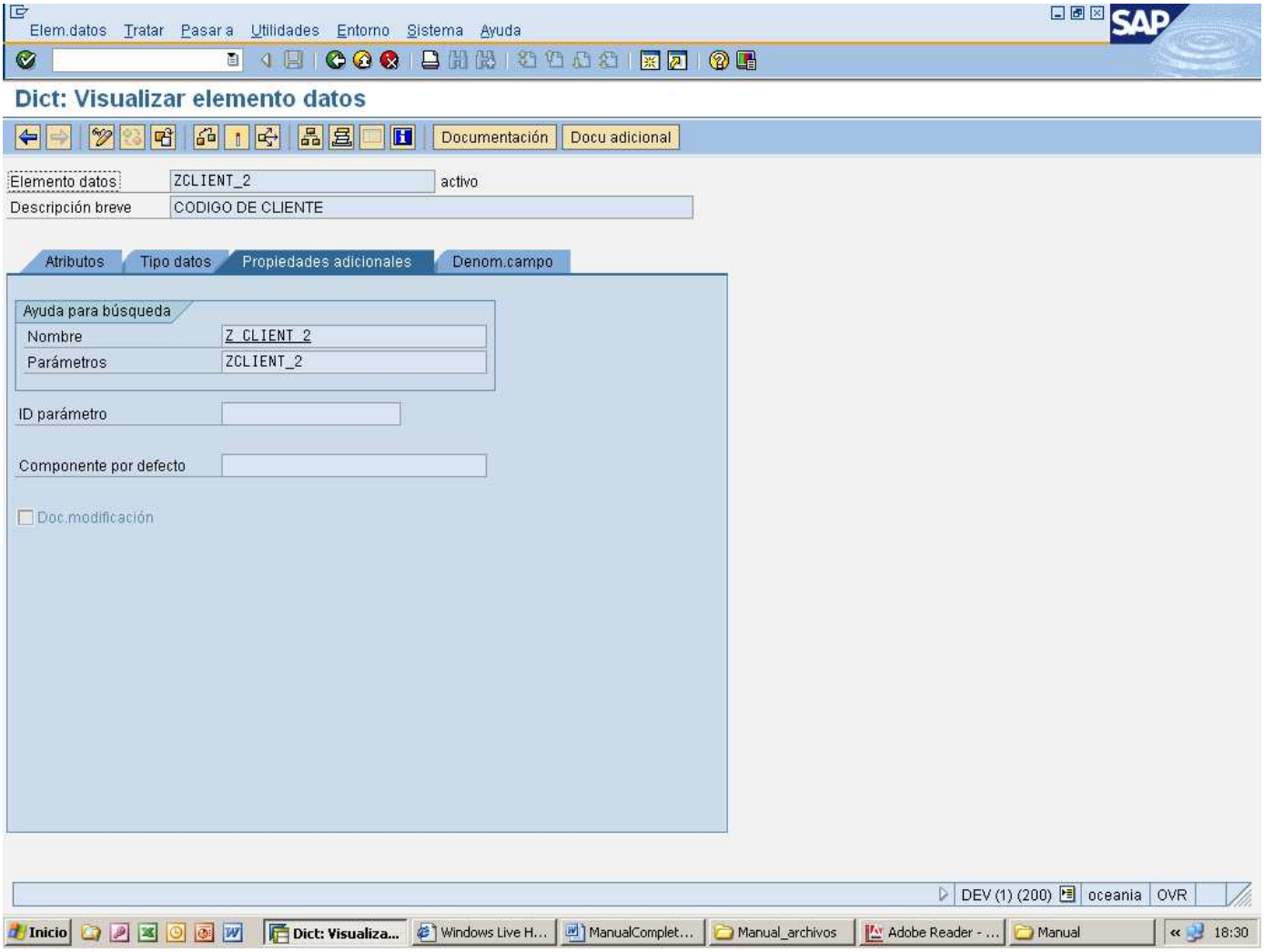
En dicha tabla tenemos varios campos que necesitan ser explicados:

- Param.Ayuda búsq: Se trata de los campos que se seleccionan de la vista o tabla
- IMP: Indica que este campo puede ser importado desde la pantalla de selección.
- EXP: Indica el campo del cual exportaremos el valor elegido.
- P.I.: Posición en la lista de aciertos

- PosS. Posición en ventana dialogo para selección de datos.

Una vez realizado todo este proceso, creado el matchcode y activado, es necesario asignar dicho machcode a un campo de la tabla. Para eso volveremos a la SE11, a la tabla y al campo que queremos 'ayudar', y le asignaremos el machcode en la columna de 'Ayuda de Busqueda'.

Tambien podemos asignar un machcode a un elemento de datos:



21. ABAP ORIENTADO A OBJETOS

21.1 ¿QUÉ ES LA ORIENTACIÓN A OBJETOS?

La programación orientada a objetos es un método de desarrollo de software basado en el comportamiento real de los objetos en el mundo real. Se pretende desarrollar componentes de software que se comporten como los objetos reales a los que representan. La orientación a objetos es una técnica usada en muchos lenguajes de programación los cuales comparten una terminología usada universalmente.

Un **objeto** es únicamente una porción de código fuente que contiene datos y proporciona servicios. Los datos constituyen los atributos del objeto. Los servicios que proporciona el objeto se conocen como métodos y se asemejan en su funcionamiento a las funciones.

Una **clase** es una entidad teórica que describe el comportamiento de un objeto. Desde un punto de vista meramente técnico, un objeto es una instancia en tiempo de ejecución de una clase. En principio se pueden crear cualquier número de objetos basados en una única clase. Cada instancia de una clase (objeto) tiene su propia identidad y su propio conjunto de valores para sus atributos. Dentro de un programa un objeto es identificado por su referencia, la cual le proporciona un nombre que define inequivocamente al objeto y permite acceder a sus métodos y atributos. Las clases globales
se definen en el generador de clases (transacción SE24) en el ABAP Workbench.

En la programación orientada a objetos, los objetos tienen normalmente las siguientes propiedades:
- Encapsulación: Los objetos restringen la visibilidad de sus atributos y métodos al resto de usuarios.
- Polimorfismo: Los métodos llamados igual pueden comportarse de manera distinta en clases diferentes.
- Herencia: Se pueden utilizar clases existentes para originar nuevas clases. Las nuevas clases originadas heredan los datos y los métodos de la superclase.

Las principales **ventajas** de la programación orientada a objetos son:

- Sistemas de software muy complejos se vuelven mucho más simples de comprender debido a que la orientación a objetos proporciona una representación mucho más cercana a la realidad que otras técnicas de programación.
- En un sistema correctamente diseñado con orientación a objetos es posible realizar cambios al nivel de la clases, sin tener que realizar cambios en ningún otro punto del sistema. Esto reduce significativamente el costo total del mantenimiento necesario.
- A través del polimorfismo y la herencia es posible la reutilización de componentes individuales.
- La cantidad de trabajo en revisión y mantenimiento del sistema se reduce debido a que muchos problemas pueden ser detectados y corregidos en la fase de diseño.

CLASES

Las clases en ABAP Objects se pueden declarar bien globalmente o bien localmente. Las clases locales son el conjunto de sentencias que están entre las sentencias CLASS… y ENDCLASS.

Una definición completa de una clase constará de una parte declarativa en la que se definen los componentes, y si es necesario una parte de implementación en la que se implementan estos componentes.

La parte declarativa de una clase está comprendida entre las sentencias:

```
CLASS <class> DEFINITION.
…
ENDCLASS.
```

La parte declarativa contiene la declaración de todos los componentes de la clase (atributos, métodos y eventos). Cuando se definen clases locales, la parte declarativa pertenece a los datos globales del programa, por tanto se habrá de situar al principio del programa.

Si se declaran métodos en la parte declarativa de una clase, se deberá escribir también su parte de

implementación. Ésta es la que va incluída entre las siguientes sentencias:

```
CLASS <class> IMPLEMENTATION.
…
ENDCLASS.
```

Estructura de una clase

Una clase contiene componentes. Cada componente se asigna a una sección de visibilidad (público, protegido o privado). Los componentes definen los atributos de los objetos en una clase.
En ABAP Obejcts, las clases pueden definir los siguientes componentes:

- **Atributos**: son los campos de datos internos de una clase..

- Atributos dependientes de instancia. – El contenido de estos atributos es específico de cada objeto. Se declaran usando la sentencia DATA.
- Atributos estáticos – El contenido de los atributos estáticos define el estado de la clase y es válido para todas las instancias la clase. Los atributos estáticos existen sólo una vez para la clase. Se declaran usando la sentencia CLASS-DATA.

- **Métodos**

Los métodos son procedimientos internos de una clase que definen el comportamiento de un objeto. Pueden acceder a todos los atributos de una clase y les permite cambiar el contenido de los atributos de un objeto. Poseen también una interface con parámetros que les permite recibir valores cuando son invocados y devolver valores después de la llamada.

Los atributos privados de una clase sólo pueden ser cambiados por métodos de la misma clase. Un método se define en la parte declarativa de la clase y se implementa en la parte de implementación usando las sentencias:

```
METHOD <meth>.
…
ENDMETHOD.
```

Los métodos se pueden llamar mediante la sentencia CALL METHOD.
- Métodos dependientes de instancia – Estos métodos se declaran usando la sentencia METHODS. Pueden acceder a todos los atributos de una clase, y pueden desencadenar todos los eventos de una clase.
- Métodos estáticos o independientes de instancia – Estos métodos se declaran usando la sentencia CLASS-METHODS. Sólo pueden acceder a los atributos estáticos y desencadenar eventos estáticos.
- Métodos especiales – Además de los métodos normales que se pueden llamar con la sentencia CALL METHOD, hay dos métodos especiales llamados CONSTRUCTOR y CLASS_CONSTRUCTOR que son automáticamente llamados cuando se crea un objeto (CONSTRUCTOR) o cuando se accede por primera vez a los componentes de la clase (CLASS_CONSTRUCTOR).

- **Eventos:** Cuando un evento es disparado, el correspondiente método manejador de eventos es ejecutado en todas las clases registradas para ese manejador.

- Eventos dependientes de instancia – Se declaran con la sentencia EVENTS. Sólo pueden ser desencadenados en un método dependiente de instancia.
- Eventos estáticos o independientes de instancia – Se declaran con la sentencia CLASS-EVENTS.

- **Tipos:** Tipos de datos definidos en una clase con la sentencia TYPE .

No son específicos de cada instancia y existen una sóla vez para todos los objetos de la clase.

- **Constantes:** Atributos estáticos con valor fijo.

Se declaran usando la sentencia CONSTANTS. Las constantes existen sólo una vez para todos los objetos de la clase.

- **Visibilidad**

La parte declarativa de una clase se divide en tres áreas de distinta visibilidad:

```
CLASS <class> DEFINITION.
PUBLIC SECTION.
…
PROTECTED SECTION.
…
PRIVATE SECTION.
…
ENDCLASS.
```

Estas tres áreas definen la visibilidad externa de los componentes de la clase, esto es, la interface entre la clase y el usuario. Cada componente de una clase ha de ser asignado a una de estas tres secciones:
- Public section – Todos los componentes declarados en la sección pública son accesibles para todos los usuarios de la clase y para todos los métodos de la clase y de cualquier clase que herede de ella. Los componentes públicos conforman la interface entre la clase y el usuario.
- Protected section – Todos los componentes declarados en la sección protegida son accesibles para todos los métodos de la clase y de las clases que heredan de ella. Los componentes protegidos conforman la interface entre una clase y todas sus subclases.
- Private section – Los componentes declarados en la sección privada son sólo visibles en los métodos de la misma clase. Los componentes privados no forman parte de la interface externa de la clase.

Esta sería la estructura de la parte de declaración y la parte de implementación de una clase local c1:

```
CLASS c1 DEFINITION.
        PUBLIC SECTION.
                DATA: a1 …
                METHODS: m1 …
                EVENTS: e1 …
        PROTECTED SECTION.
                DATA: a2 …
                METHODS: m2 …
                EVENTS: e2 …
        PRIVATE SECTION.
                DATA: a3 …
                METHODS: m3 …
                EVENTS: e3 …
ENDCLASS.
CLASS c1 IMPLEMENTATION.
        METHOD m1 … ENDMETHOD.
        METHOD m2 … ENDMETHOD.
        METHOD m3 … ENDMETHOD.
ENDCLASS.
```

Este seria el ejemplo de una clase local Contador:

```
CLASS CONTADOR DEFINITION.
        PUBLIC SECTION.
                METHODS: FIJAR_CONTADOR IMPORTING
                          VALUE(FIJAR_VALOR) TYPE I,
                          INCREMENTAR_CONTADOR,
                         OBTENER_CONTADOR EXPORTING
                         VALUE(OBTENER_VALOR) TYPE I.
        PRIVATE SECTION.
                DATA CONT TYPE I.
ENDCLASS.
CLASS CONTADOR IMPLEMENTATION.
```

```
        METHOD FIJAR_CONTADOR.
            CONT = FIJAR_VALOR.
        ENDMETHOD.
        METHOD INCREMENTAR_CONTADOR.
            ADD 1 TO CONT.
        ENDMETHOD.
        METHOD OBTENER_CONTADOR.
            OBTENER_VALOR = CONT.
        ENDMETHOD.
ENDCLASS.
```

La clase contador contiene tres métodos públicos, fijar_contador, incrementar_contador y obtener_contador. Cada uno de ellos trabaja con el atributo privado cont. Dos de estos métodos tienen parámetros de entrada y de salida. Estos son los que forman la interface de la clase. El atributo cont no es visible directamente.

21.2 UTILIZACIÓN DE OBJETOS

Antes de crear un objeto de una clase es necesario declarar una variable referenciada con la clase. Las referencias a clases se definen usando la siguiente adición:

```
… TYPE REF TO <class>.
```

Esta adición se usa en las sentencias TYPES o DATA. Estas variables permiten al usuario crear una instancia, o sea un objeto, de la clase y acceder a un componente visible de la siguiente manera:

```
cref->comp
```

Una vez que se ha declarado la referencia <obj> a la clase <class>, se puede crear el objeto usando la sentencia CREATE OBJECT <cref>. Esta sentencia crea una instancia de la clase <class>, y la variable referenciada <cref> contiene la referencia al objeto.

Los programas sólo pueden acceder a los componentes de las instancias usando las referencias de las variables referenciadas. La sintaxis es la siguiente, siendo ref la variable referenciada:
- Para acceder al atributo attr: ref->attr.
- Para llamar al método meth: CALL METHOD ref->meth.

Se pueden asignar referencias a distintas variables referenciadas usando la sentencia MOVE. De esta manera se puede tener varias variables referenciadas apuntando al mismo objeto. Esto tambien funciona con la sentencia cref1 = cref2.

EJEMPLO: Como crear y usar una clase.

```
DATA cref1 TYPE REF TO c_contador.
DATA cref2 LIKE cref1.
DATA numero1 TYPE i VALUE 5.
DATA numero2 TYPE i VALUE 0.

CREATE OBJECT cref1, cref2.

CALL METHOD: cref1->fijar_contador
                EXPORTING fijar_valor = numero1,
             cref2->fijar_contador
                EXPORTING fijar_valor = numero2.
CALL METHOD cref2->incrementar_contador.
CALL METHOD cref1->incrementar_contador.
```

```
CALL METHOD: cref1->obtener_contador
                    IMPORTING obtener_valor = numero1,
             cref2->obtener_contador
                    IMPORTING obtener_valor = numero2.
```

Declaracion y llamada de metodos

Para declarar métodos dependientes de instancia se usa la siguiente sentencia:

```
METHODS <meth>
IMPORTING.. [VALUE(]<ii>[)] TYPE type [OPTIONAL]..
EXPORTING.. [VALUE(]<ei>[)] TYPE type [OPTIONAL]..
CHANGING.. [VALUE(]<ci>[)] TYPE type [OPTIONAL]..
RETURNING VALUE(<r>)
EXCEPTIONS.. <ei>..
```

donde IMPORTING, EXPORTING, CHANGING, y RETURNING definen los parámetros de entrada, de salida, de entrada/salida y el código que devuelve el método. Para declarar métodos estáticos se usa se usa la sentencia CLASS-METHODS <meth>.. con la misma sintaxis que la anterior.

Al igual que en los módulos de funciones, se puede usar las sentencias RAISE <exception> y MESSAGE RAISING para controlar los errores. Para llamar a un método se usa la siguiente sentencia:

```
CALL METHOD <meth>
EXPORTING... <ii> =.<f i>...
IMPORTING... <ei> =.<g i>...
CHANGING ... <ci> =.<f i>...
RECEIVINGr=h
EXCEPTIONS... <ei>=rc i...
```

Desde fuera de la clase, podremos llamar sólo a los métodos cuya visibilidad nos permite hacerlo. Los métodos visibles dependientes de instancia pueden ser llamados desde fuera de la clase usando la sentencia CALL METHOD <ref>-><meth>... donde <ref> es una variable referenciada cuyo valor apunta a una instancia de la clase.

Un método se define como manejador de eventos mediante la adición:

```
... FOR EVENT <evt> OF <cif>...
```

Las interfaces de los métodos manejadores de eventos tienen unas reglas a cumplir:
- La interface sólo puede tener parametros de entrada (IMPORTING).
- Cada parámetro de entrada al método (IMPORTING) debe tener su correspondiente parámetro de salida (EXPORTING) en el evento.
- Los atributos de los parámetros están definidos en la declaración del evento (sentencia EVENTS) y son tomados por el método manejador de eventos.

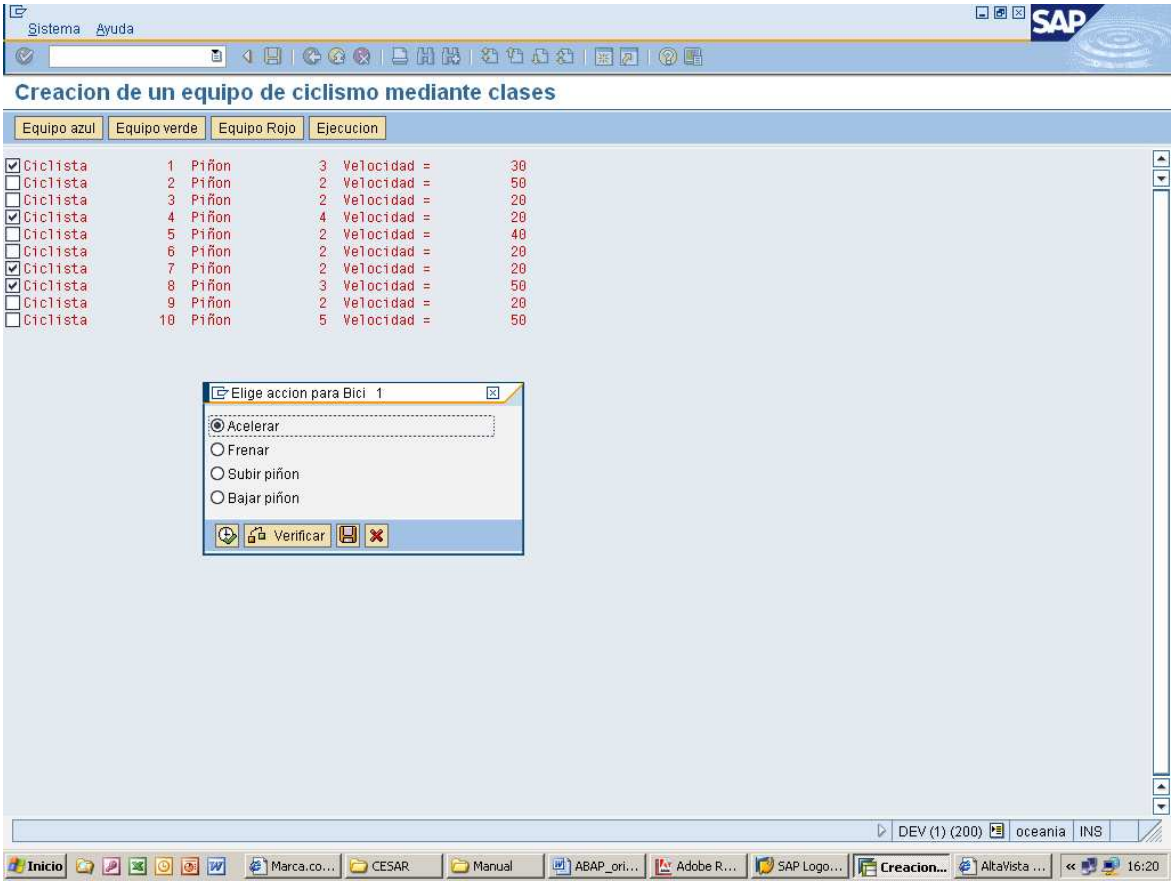
Los constructores son un tipo especial de métodos que no pueden ser llamados con la sentencia CALL METHOD. Estos métodos son llamados automáticamente por el sistema para fijar el estado inicial de un nuevo objeto o clase. Se declara en la sección pública de la siguiente manera:

```
METHODS CONSTRUCTOR
IMPORTING.. [VALUE(]<ii>[)] TYPE type [OPTIONAL]..
EXCEPTIONS.. <ei>.
```

El sistema llama al constructor justo después de que el objeto haya sido creado mediante la sentencia CREATE OBJECT. Se le pueden pasar parámetros de entrada y controlar sus errores usando las adiciones

EXPORTING y EXCEPTIONS en la sentencia CREATE OBJECT.

EJEMPLO: Este ejemplo usa tres clases llamadas c_team, c_biker y c_bicycle. El usuario puede crear objetos de la clase c_team. Ver el programa ZCLASS_TEAM.



21.3 HERENCIA

La herencia permite crear una nueva clase a partir de una nueva existente heredando la nueva clase sus propiedades. Esto se realiza añadiendo la adición INHERITING FROM a la sentencia de definición de la clase:

CLASS <subclass> DEFINITION INHERITING FROM <superclass>.

La nueva clase <subclass> hereda todos los componentes de la clase ya existente <superclase>. Si la superclase no tiene una sección privada, la subclase es una réplica exacta de la superclase.
Se puede usar la pseudoreferencia SUPER-> para acceder a la definición de un método en una superclase que ha sido oscurecida por la redefinición en la subclase.

Ejemplo de herencia

El siguiente ejemplo muestra el concepto de herencia en ABAP Objects. Una nueva clase counter_ten hereda de la clase ya existente counter.

```
CLASS counter DEFINITION.
        PUBLIC SECTION.
                METHODS: set IMPORTING value(set_value) TYPE i,
                         increment,
                         get EXPORTING value(get_value) TYPE i.
```

```
        PROTECTED SECTION.
                DATA count TYPE i.
ENDCLASS.
CLASS counter IMPLEMENTATION.
        METHOD set.
                count = set_value.
        ENDMETHOD.
        METHOD increment.
                ADD 1 TO count.
        ENDMETHOD.
        METHOD get.
                get_value = count.
        ENDMETHOD.
ENDCLASS.
*********************************************************************
CLASS counter_ten DEFINITION INHERITING FROM counter.
        PUBLIC SECTION.
                METHODS increment REDEFINITION.
                DATA count_ten.
ENDCLASS.
CLASS counter_ten IMPLEMENTATION.
        METHOD increment.
                DATA modulo TYPE I.
                CALL METHOD super->increment.
                WRITE / count.
                modulo = count mod 10.
                IF modulo = 0.
                        count_ten = count_ten + 1.
                        write count_ten.
                ENDIF.
        ENDMETHOD.
ENDCLASS.
*********************************************************************

Report Example
DATA: count TYPE REF TO counter,
        number TYPE i VALUE 5.
START-OF-SELECTION.
        CREATE OBJECT count TYPE counter_ten.
        CALL METHOD count->set EXPORTING set_value = number.
        DO 20 TIMES.
                CALL METHOD count->increment.
        ENDDO.
```

La clase counter_ten se deriva de la clase counter. Se redefine el método increment. Para hacer esto se tiene que cambiar la visibilidad del atributo count de privado a protegido. El método redefinido llama al método 'oscurecido' de la superclase usando la pseudoreferencia SUPER->. El método redefinido es una especialización del método heredado. El ejemplo instancia la subclase. La variable referenciada que apunta a la subclase tiene el tipo de la superclase. Cuando el método increment es llamado usando la referencia de la superclase, el sistema ejecuta el método redefinido de la subclase.

Interfaces

Las interfaces son estructuras independientes que se pueden implementar en una clase para extender el ámbito de esa clase. Son como includes que pueden ser añadidas en una clase. Las interfaces extienden el ámbito de una clase añadiendo sus propios componentes a la sección pública. Esto permite a los usuarios acceder a diferentes clases por medio de un punto de contacto común.

Las interfaces junto con la herencia proporcionan uno de los pilares básicos del polimorfismo, ya que permiten que un sólo método con una interface se comporte distinto en diferentes clases.

La definición de una interface local es el código existente entre las sentencias:

```
INTERFACE <intf>.
…
ENDINTERFACE.
```

La definición contiene la declaración de todos los componentes (atributos, métodos, eventos) de la interface. Se pueden definir los mismos componentes en una interface que en una clase. Los componentes de las interfaces pertenecen a la sección pública de la clase en la que la interface es implementada.

Las interfaces no tienen una parte de implementación ya que sus métodos son implementados en la clase que implementa la interface.

Cuando se implementa una interface en una clase, los componentes de la interface se añaden al resto de componentes de la sección pública. Un componente <icomp> de una interface <intf> puede ser direccionado como si fuese un miembro de la clase bajo el nombre <intf~icomp>:

```
METHOD <intf~imeth>.
…
ENDMETHOD.
```

Las interfaces pueden ser implementadas por diferentes clases. Cada una de las clases es ampliada con el mismo conjunto de componentes, aunque los métodos de la interface pueden ser implementados de manera distinta en cada clase.

El siguiente **ejemplo** muestra cómo se puede usar una interface para implementar dos contadores diferentes pero que se pueden llamar de la misma manera.

```
**********************************************************************
INTERFACE I_COUNTER.
        METHODS: SET_COUNTER IMPORTING VALUE(SET_VALUE) TYPE I,
                        INCREMENT_COUNTER,
                        GET_COUNTER EXPORTING VALUE(GET_VALUE) TYPE I.
ENDINTERFACE.
**********************************************************************
CLASS C_COUNTER1 DEFINITION.
        PUBLIC SECTION.
                INTERFACES I_COUNTER.
        PRIVATE SECTION.
                DATA COUNT TYPE I.
ENDCLASS.
CLASS C_COUNTER1 IMPLEMENTATION.
        METHOD I_COUNTER~SET_COUNTER.
                COUNT = SET_VALUE.
        ENDMETHOD.
        METHOD I_COUNTER~INCREMENT_COUNTER.
                ADD 1 TO COUNT.
        ENDMETHOD.
        METHOD I_COUNTER~GET_COUNTER.
                GET_VALUE = COUNT.
        ENDMETHOD.
ENDCLASS.
**********************************************************************
CLASS C_COUNTER2 DEFINITION.
        PUBLIC SECTION.
```

```
                INTERFACES I_COUNTER.
        PRIVATE SECTION.
                DATA COUNT TYPE I.
ENDCLASS.
CLASS C_COUNTER2 IMPLEMENTATION.
        METHOD I_COUNTER~SET_COUNTER.
                COUNT = ( SET_VALUE / 10) * 10.
        ENDMETHOD.
        METHOD I_COUNTER~INCREMENT_COUNTER.
                IF COUNT GE 100.
                        MESSAGE I042(00).
                        COUNT = 0.
                ELSE.
                        ADD 10 TO COUNT.
                ENDIF.
        ENDMETHOD.
        METHOD I_COUNTER~GET_COUNTER.
                GET_VALUE = COUNT.
        ENDMETHOD.
ENDCLASS.
*********************************************************************
```

La interface i_counter tiene tres métodos, set_counter, increment_counter y get_counter. Las clases c_counter1 y c_counter2 implementan la interface en sus secciones públicas, pero de distinta manera. La interface ejerce casi como un include.

21. 4 DISPARAR Y MANEJAR EVENTOS

En ABAP Objects hay ciertos métodos que se conocen como disparadores (triggers) y otros que se conocen como manejadores (handlers). Los triggers son los métodos que disparan un evento, mientras que los handlers son los métodos que se ejecutan cuando ocurre un evento.

Para disparar un evento una clase tiene que:
- declarar el evento en la parte declarativa
- disparar el evento en uno de sus métodos.

Los eventos se declaran en la parte declarativa de una clase o en una interface. Para declarar eventos dependientes de instancia se usa la sentencia:

```
EVENTS <evt> EXPORTING... VALUE(<ei>) TYPE type [OPTIONAL]..
```

Para declarar eventos estáticos se usa la sentencia:

```
CLASS-EVENTS <evt>...
```

Cuando se declara un evento se puede usar la adición EXPORTING para especificar parámetros que se pasan al manejador del evento. Los parámetros se pasan siempre por valor. Los eventos dependientes de instancia siempre contienen el parámetro implícito SENDER, el cual tiene el tipo de una referencia al tipo o a la interface en la cual el evento es declarado.

Un evento dependiente de instancia en una clase puede ser disparado por cualquier método en la clase. Los eventos estáticos son disparados por métodos estáticos. Para disparar un evento en un método se usa la siguiente sentencia:

```
RAISE EVENT <evt> EXPORTING... <ei>=<f i>...
```

Por cada parámetro formal <ei> que no esté definido como opcional se tiene que pasar el correspondiente parámetro real <f i> en la adición EXPORTING. La referencia a sí mismo ME es pasada automáticamente al parámetro implícito SENDER.

Los eventos se usan para ejecutar una serie de métodos. Estos métodos tienen que:
- estar definidos como eventos manejadores (handler) de ese evento
- estar registrados en tiempo de ejecución para el evento.

Una clase puede contener métodos manejadores de eventos para eventos tanto de su propia clase como de otras clases. Para declarar un método manejador de eventos dependiente de instancia se usa la siguiente sentencia:

```
METHODS <meth> FOR EVENT <evt> OF <cif> IMPORTING.. <ei>..
```

Para métodos estáticos se usa la misma sentencia con CLASS-METHODS en vez de METHODS. <evt> es un evento declarado en la clase o en la interface <cif>.

Para permitir a un método manejador de eventos reaccionar a un evento, se tiene que determinar en tiempo de ejecución el disparador al cual va a reaccionar. Esto se hace con la siguiente sentencia:

```
SET HANDLER... <hi>... [FOR]...
```

Esta sentencia relaciona los métodos manejadores de eventos con sus correspondientes métodos. Hay cuatro tipos diferentes de eventos:
- Eventos dependientes de instancia declarados en una clase.
- Eventos dependientes de instancia declarados en una interface.
- Eventos estáticos declarados en una clase.
- Eventos estáticos declarados en una interface.

La sintaxis y el efecto de la sentencia SET HANDLER depende de cual de los cuatro casos de arriba tenga lugar. Para un evento dependiente de instancia se tiene que usar la adición FOR para especificar la instancia para la cual se quiere registrar el manejador. Se puede especificar una sóla instancia como disparador usando
una variable referenciada <ref>:

```
SET HANDLER... <hi>...FOR <ref>.
```

o se puede registrar el manejador para todas las instancias que puedan disparar el evento:

```
SET HANDLER... <hi>...FOR ALL INSTANCES.
```

En este caso el registro se aplica incluso a las instancias que aún no han sido creadas cuando se registra el manejador. No se puede usar la adición FOR para los eventos estáticos:

```
SET HANDLER... <hi>...
```

Después de la sentencia RAISE EVENT, todos los métodos manejadores registrados son ejecutados antes de que la siguiente sentencia sea procesada (manejo de eventos sincrónico).

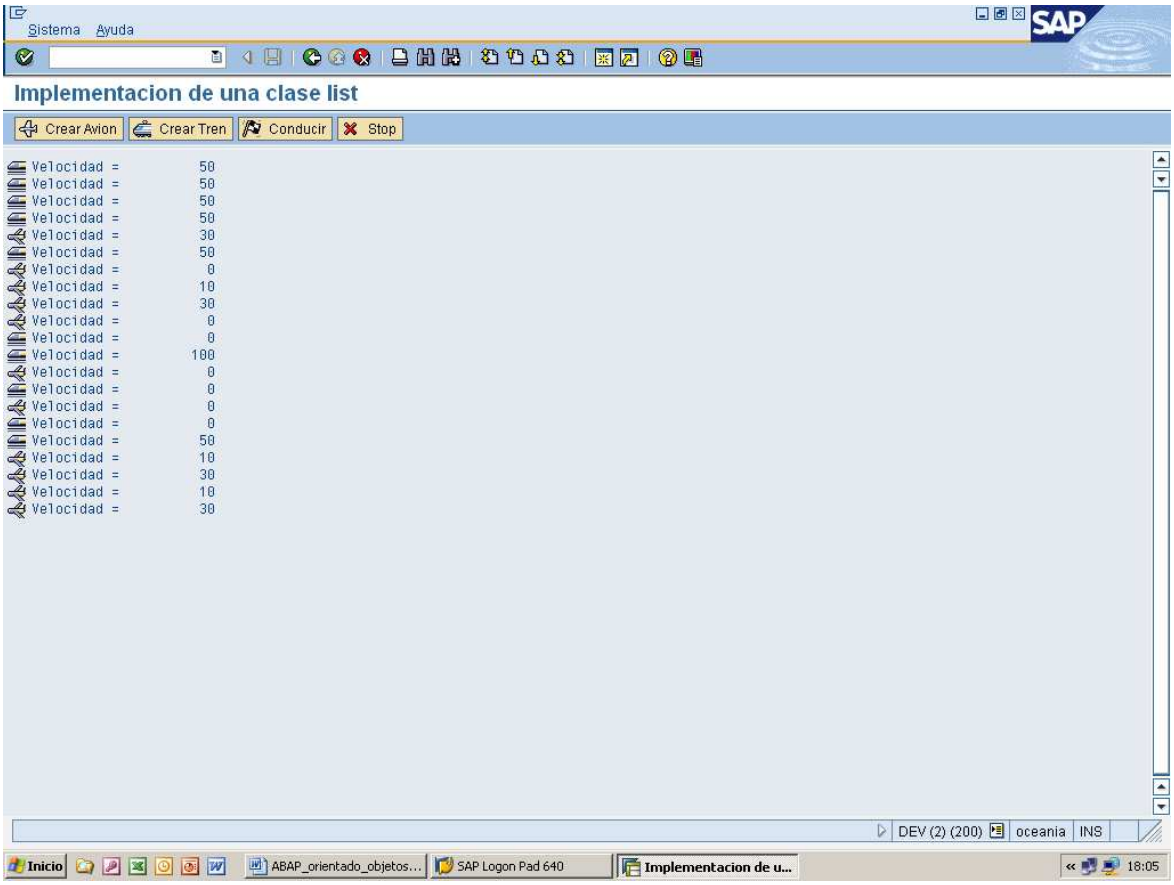
El siguiente **ejemplo** muestra cómo se trabaja con eventos en ABAP Objects:

```
REPORT demo_class_counter_event.
*********************************************************************
CLASS counter DEFINITION.
        PUBLIC SECTION.
                METHODS increment_counter.
                EVENTS critical_value EXPORTING value(excess) TYPE i.
        PRIVATE SECTION.
                DATA: count TYPE i,
                        threshold TYPE i VALUE 10.
```

```
ENDCLASS.
*****************************************************************
CLASS counter IMPLEMENTATION.
        METHOD increment_counter.
                DATA diff TYPE i.
                ADD 1 TO count.
                IF count > threshold.
                        diff = count -threshold.
                        RAISE EVENT critical_value EXPORTING excess = diff.
                ENDIF.
        ENDMETHOD.
ENDCLASS.
*****************************************************************
CLASS handler DEFINITION.
        PUBLIC SECTION.
                METHODS handle_excess
                FOR EVENT critical_value OF counter
        IMPORTING excess.
ENDCLASS.
*****************************************************************
CLASS handler IMPLEMENTATION.
        METHOD handle_excess.
                WRITE: / 'Excess is', excess.
        ENDMETHOD.
ENDCLASS.
*****************************************************************
DATA: r1 TYPE REF TO counter,
        h1 TYPE REF TO handler.
START-OF-SELECTION.
        CREATE OBJECT: r1, h1.
        SET HANDLER h1->handle_excess FOR ALL INSTANCES.
        DO 20 TIMES.
                CALL METHOD r1->increment_counter.
        ENDDO.
```

La clase counter implementa un contador. Se desencadena el evento critical_value cuando el valor umbral (threshold) es excedido, y se visualiza la diferencia

El ejemplo ZCLASS_LIST implementa dos clases, camion y barco, que comparten una interface e implementan una serie de eventos y metodos compartidos. Ver programa ZCLASS_LIST.

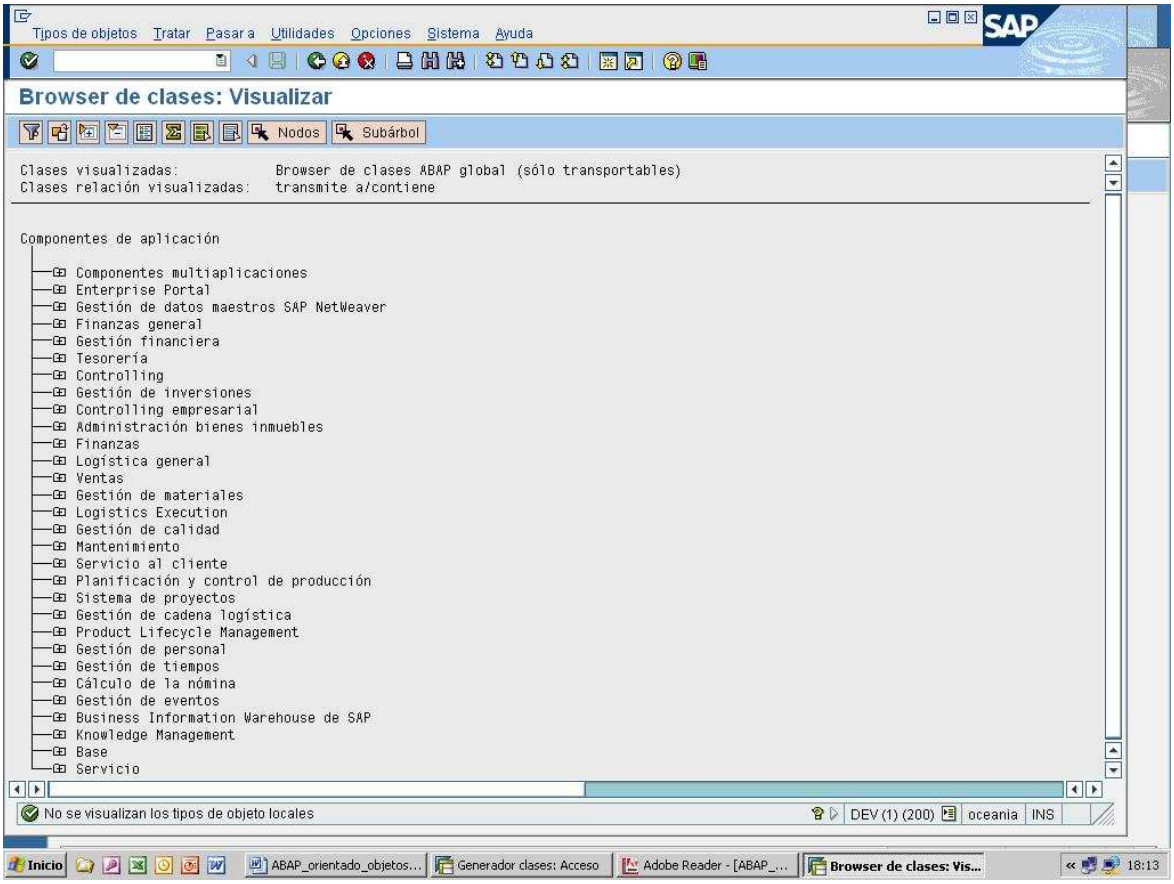


21.5 INTRODUCCIÓN AL GENERADOR DE CLASES

El constructor de clases (SE24) permite crear, visualizar y mantener clases e interfaces globales. Ambos tipos de objetos, al igual que los tipos de datos globales son definidos en el R/3 Repository.

Para visualizar los tipos de objetos existentes en la librería de clases se usa el class browser.
El class browser está integrado en el constructor de clases, se puede iniciar desde el constructor o bien desde la transacción CLABAP.
Existe un rango preconfigurado de vistas que se pueden usar para visualizar tipos de objetos. Se puede seleccionar por una serie de filtros:
- Todas las clases. Se visualizan todas las clases e interfaces de la librería de clases.
- Objetos de negocio (business objects). Se visualizan los tipos de objetos de negocio de la librería de clases
- Otras selecciones. Se pueden usar otros tres filtros:
- Tipos de objetos
- Relaciones entre objetos
- Otros

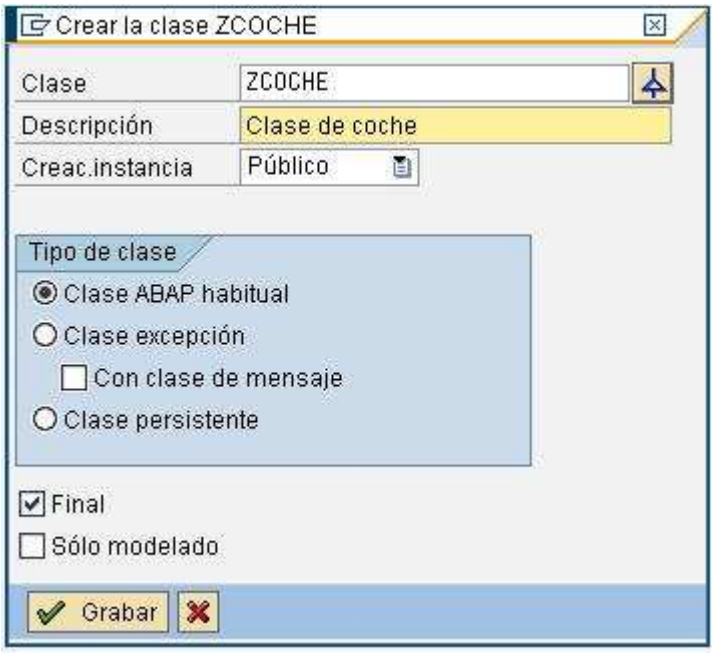
No se pueden crear nuevos tipos de objetos desde el class browser.



Crear nuevas clases e intefaces

Para crear una nueva clase desde la pantalla inicial del ABAP Workbench:

1) En tipo de objeto se escribe el nombre de la nueva clase.
2) Se selecciona el tipo de objeto clase (clase o interface)
3) Se definen los datos básicos, que son la siguiente información:
- Clase – nombre de la nueva clase.
- Descripción – descripción de la nueva clase.
- Crear instancia – el valor por defecto es público. Esto quiere decir que cualquier usuario puede instanciar la clase usando la sentencia CREATE OBJECT. Si se selecciona protegido, sólo la propia clase y sus subclases pueden instanciar la clase. Si se elige privado sólo la propia clase puede instanciarse (desde uno de sus métodos). Si se selecciona abstracto se crea una clase que no puede ser instanciada. Las clases abstractas sirven como plantillas para las subclases y sólo se puede acceder a ellas usando sus atributos estáticos o los de sus subclases.



- Final – si se selecciona final, se define una clase final. Una clase final es la última del árbol de herencia y no puede tener subclases. Si se crea una clase abstracta final sólo se puede acceder a sus componentes estáticos.
- Modelado – si se selecciona el sistema no definirá la subclase en el pool de clases
- Icono de herencia – Si se selecciona esta función (botón al lado de la clase) aparece un cuadro de diálogo. Desde aquí se puede introducir el nombre de una superclase. La superclase puede ser cualquier clase de la librería de clases que no esté definida como final.

4) Se elige grabar y aparece el cuadro de diálogo en el que se indica la clase de desarrollo.
5) Se graba y aparece el editor de clases. Desde aquí se pueden definir los componentes de la clase e incluir interfaces.

Ya se ha definido una nueva clase. Cuando se introducen los datos básicos el sistema crea un pool de clases para la nueva clase (siempre que no se haya definido como sólo modelado). Un pool de clases contiene la definición de una sóla clase global. Son similares a los grupos de funciones en el sentido de que se pueden definir clases locales auxiliares y tipos locales.

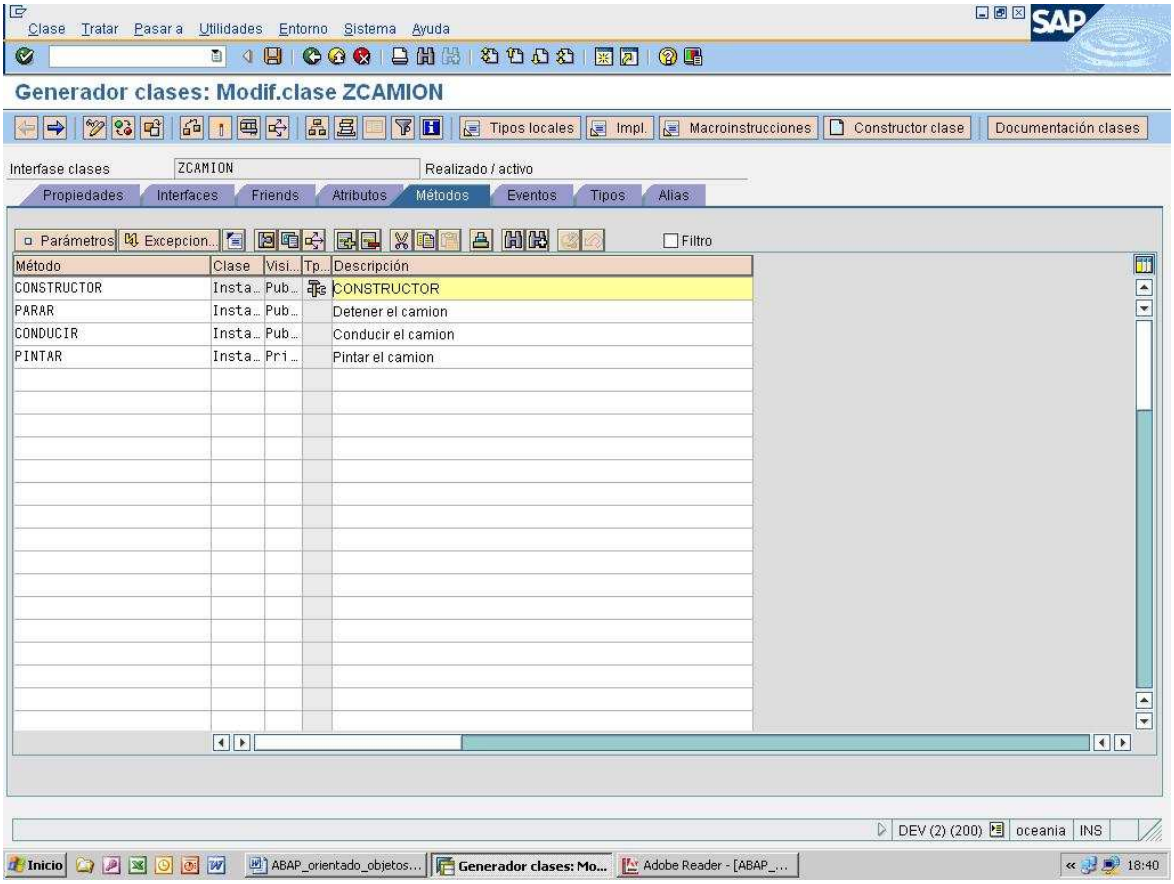
Una vez creada una clase o una interface como se acaba de describir accedemos a la pantalla desde la cual podemos definir los componentes de nuestra clase o interface. Desde esta pantalla podemos:
- Definir las clases o las interfaces asignándoles sus componentes.
- Implementar los métodos de las clases.
- Añadir interfaces a las clases e implementar sus métodos en las clases.
- Cambiar las definiciones ya existentes y la implementación de las clases.
- Definir tipos de datos locales dentro de las clases.

Asignamos componentes definiendo:
- Atributos.
- Métodos.
- Eventos.
- Tipos locales en las clases.
- Interfaces.

En la pestaña métodos podemos:
- Definir los parámetros de los métodos.
- Definir las excepciones de los métodos.
- Implementar los métodos.

En la pestaña interfaces podemos:
- Asignar interfaces a las clases.
- Implementar los métodos de las interfaces en las clases.

El **editor de clases** es una parte del constructor de clases en la que se definen los atributos, métodos, eventos, y tipos de datos que conforman los componentes de uan clase. Las clases o interfaces definidas en el editor de clases son guardadas en la librería de clases. Se puede acceder al editor ABAP al escribir la implementación de un método.

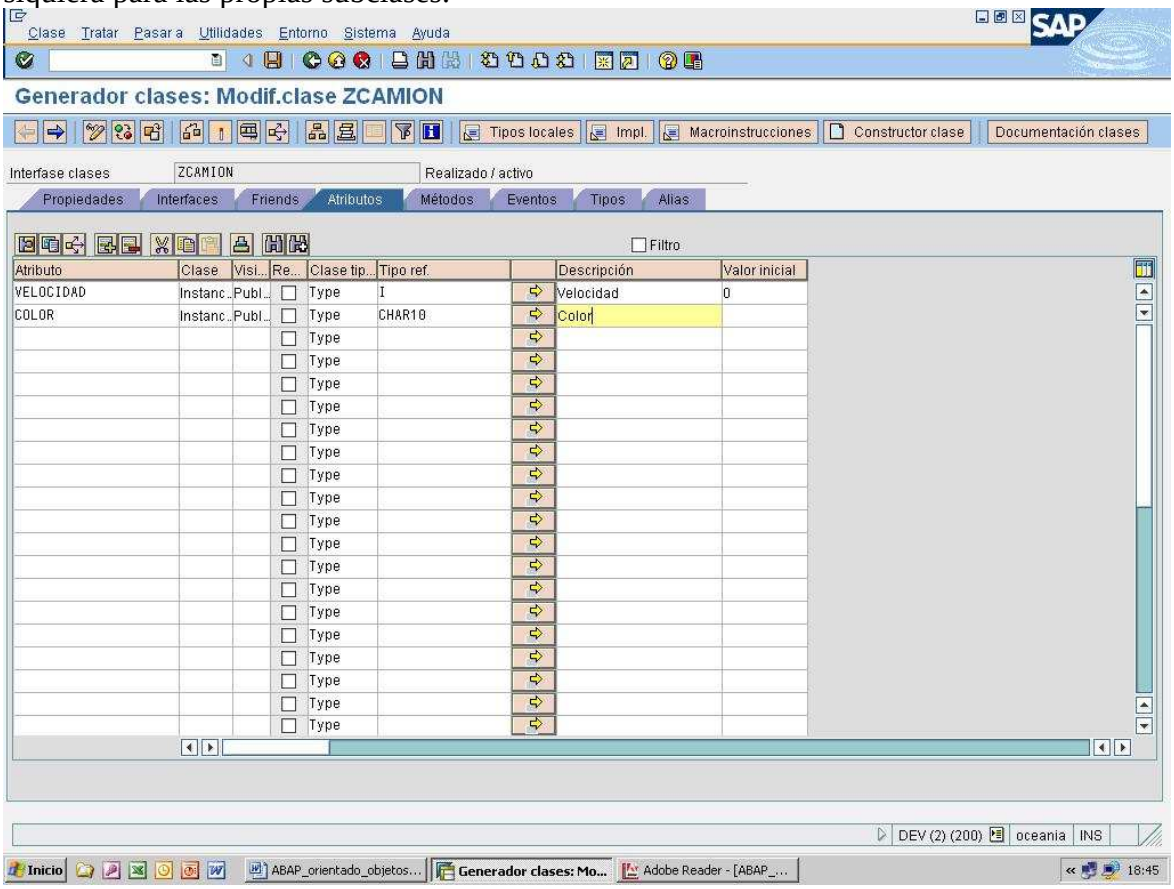


Las funciones básicas son las siguientes:
- Creación de atributos.
- Creación de métodos.
- Creación de eventos.
- Implementación de métodos.
- Creación de interfaces en clases.
- Creación de tipos internos en una clase.

Otras funciones que se realizan desde aquí son:
- La función Clases Locales (definición de tipos locales, ctrl + F5) permite crear clases locales auxiliares en el pool de clases de la clase global.
- La función Alias permite definir alias para los componentes.

- La función Documentación permite documentar las clases o las interfaces y sus componentes.
- La función Pasar a permite acceder al código (parte pública, protegida o privada)

Los **atributos** contienen datos. Son los que definen el estado de un objeto. Se va al editor de clases en modo modificar, se escoge la pestaña de atributos y se que rellenan los siguientes campos:
- Atributo – nombre que identifica al atributo.
- Clase – tipo de atributo, ya sea constante, dependiente de instancia o estático.
- Visibilidad – Define la visibilidad de los atributos. Si es público, se asigna a la sección pública de la clase y puede acceder a él cualquier usuario de la clase. Los atributos protegidos son visibles unicamente desde las subclases de la clase. Los atributos privados son sólo visibles desde la propia clase, y no son visibles ni siquiera para las propias subclases.



- Sólo modelado – No guarda el componente en el pool de clases y no puede ser llamado en tiempo de ejecución.
- Sólo lectura –esta opción podemos impide que el usuario cambie el valor del atributo.
- Clase tipif. – Referencia al tipo ABAP. Puede ser TYPE, LIKE o TYPE REF TO.
- Tipo ref. – puede ser un tipo ABAP elemental o un tipo de objeto.
- Descripción – Descripción breve del componente.
- Valor inicial – Si el atributo es una constante es necesario especificar este campo.

El sistema genera el correspondiente código ABAP en la definición de la clase o de la interface en el pool de clases para todos los atributos excepto para aquellos marcados como Sólo modelado.

Los **métodos** describen el comportamiento de los objetos. Se implementan usando funciones definidas dentro de las clases. Los métodos son operaciones que cambian los atributos de una clase o una interface.

Hay dos tipos de métodos, los dependientes de instancia, que son aquellos que se refieren a cada instancia particular y los métodos estáticos, que son aquellos compartidos por todas las instancias de una clase. Los métodos estáticos sólo pueden usar atributos estáticos. Para crear los métodos es conveniente haber creado antes los atributos de la clase para así poder implementarlos directamente.

Se va al editor de clases en modo modificar y se rellenan los siguientes campos:
- Métodos – nombre que identifica al método.
- Clase – tipo de método, dependiente de instancia o estático.
- Visibilidad – tipo de método, visible, protegido o privado. Un método privado sólo puede llamarse desde dentro de la clase, o sea desde otro método de la clase, mientras que uno protegido sólo puede llamarse desde otro método de la misma clase o bien de una subclase.
- Sólo modelado – El sistema no guarda el método en el pool de clases.
- Descripción – Descripción breve del método.

Si se crea un método constructor, el sistema asigna el nombre CONSTRUCTOR o CLASS_CONSTRUCTOR automáticamente. El constructor de clases también define otros atributos en este caso. Antes de implementar el método es conveniente crear sus parámetros y sus excepciones.

Los métodos pueden tener parámetros de entrada (importing o changing) y parámetros de salida (exporting, changing y returning). Para crear los parámetros y las excepciones se tienen que haber creado antes los métodos, los atributos y los eventos de la clase o la interface. Cuando se redefinen métodos heredados, no se puede cambiar la interface de parámetros o añadir nuevos parámetros.

Se posiciona en el nombre del método o del evento y se escoge **parámetros**. Se necesita completar la siguiente información:
- Parámetro – Nombre que identifica al parámetro.
- Clase – Puede tener el tipo importing, exporting, changing o returning.
- Traspasar valores
- Opcional – Para que el parámetro no sea obligatorio.
- Clase tipif. - Puede ser TYPE, LIKE o TYPE REF TO.
- Tipo ref. - puede ser un tipo ABAP elemental o un tipo de objeto (clase o interface).
- Valor propuesta – valor por defecto del parámetro.
- Descripción – descripción breve del parámetro.

Para crear **excepciones** se posiciona sobre el nombre del método y se selecciona excepciones. Se necesitan rellenar los siguientes campos:
- Excepción – nombre único que identifica a la excepción.
- Descripción – descripción breve de la excepción.

Implementacion de Metodos

Una vez definido el metodo se hace doble click sobre dicho metono o se elige código fuente con lo que se accede al editor ABAP. Ahí se escribe el código del método.

Creación de Eventos

Los objetos pueden indicar que su estado ha cambiado disparando un evento. Los eventos se pueden definir tanto en clases como en interfaces. Los eventos se desencadenan desde un método mediante la sentencia

RAISE EVENT. Cada clase o interface que va a manejar el correspondiente evento debe implementar el método manejador de eventos apropiado y registrarlo usando la sentencia SET HANDLER. Cuando un evento tiene lugar, el sistema llama al método manejador registrado para ese evento. Al igual que las definiciones de los métodos, los eventos tienen una interface de parámetros. La única diferencia es que los eventos sólo pueden tener parámetros EXPORTING.

En el editor de clases se escoge la pestaña Eventos. Se tiene que introducir la siguiente información:
- Evento – Nombre que identifica al evento.
- Clase – Especifica el tipo de evento, estático o dependiente de instancia.
- Visibilidad – Especifica el tipo de evento, público, protegido o privado.
- Sólo modelado – Si se selecciona esta opoción, el sistema no crea el evento en el pool de clases. No se podría entonces acceder al componente en tiempo de ejecución.
- Descripción – descripción breve del evento.

No se deben crear **tipos de datos** públicos dentro de las clases globales. Desde el editor de clases se elige la pestaña Tips internos y se rellena la siguiente información:
- Tipo – Nombre que identifica al tipo.
- Visibilidad - tipo de método, visible, protegido o privado.
- Sólo modelado - el sistema no crea el tipo en el pool de clases.
- Clase tipif. - Puede ser TYPE, LIKE o TYPE REF TO .
- Tipo ref. - puede ser un tipo ABAP elemental o un tipo de objeto (clase o interface).
- Descripción – descripción breve del tipo.

Se pueden definir las siguientes **relaciones entre dos tipos de objetos**:
- Herencia entre dos clases.
- Extensión de la funcionalidad de una clase mediante la implementación de interfaces. Esta es una relación entre clases e interfaces.
- Interfaces compuestas. Esta es una relación entre dos interfaces.

La herencia es una relación entre clases. Permite derivar una nueva clase a partir de la definición de una clase ya existente. La nueva clase se conoce como subclase mientras que la clase ya existente es la superclase.

Implementacion de interfaces

Las **interfaces** son extensiones a las definiciones de las clases y proporcionan un punto de contacto uniforme entre distintas clases. Al contrario que las clases, las interfaces no pueden ser inicializadas. En su lugar, las clases implementan las interfaces implementando todos sus métodos. En la pestaña Interfaces del editor de clases se añade la interface a la clase especificando:
- Interface – Nombre que identifica a la interface. Al pulsar enter, el sistema chequea que la interface exista en la librería de clases.
- Sólo modelado – el sistema no crea la interface en el pool de clases.

Las interfaces están en la parte declarativa de la clase en el pool de clases bajo la sentencia INTERFACES. Ahora se tienen que implementar los métodos de la interface de la manera que ya hemos visto. Los componentes de la interface, atributos, métodos y eventos aparecen en la clase en la forma <nombre de la interface>~<nombre del componente>. Esto asegura que no existan conflictos con los nombres de los componentes de la clase.

Creación de subclases

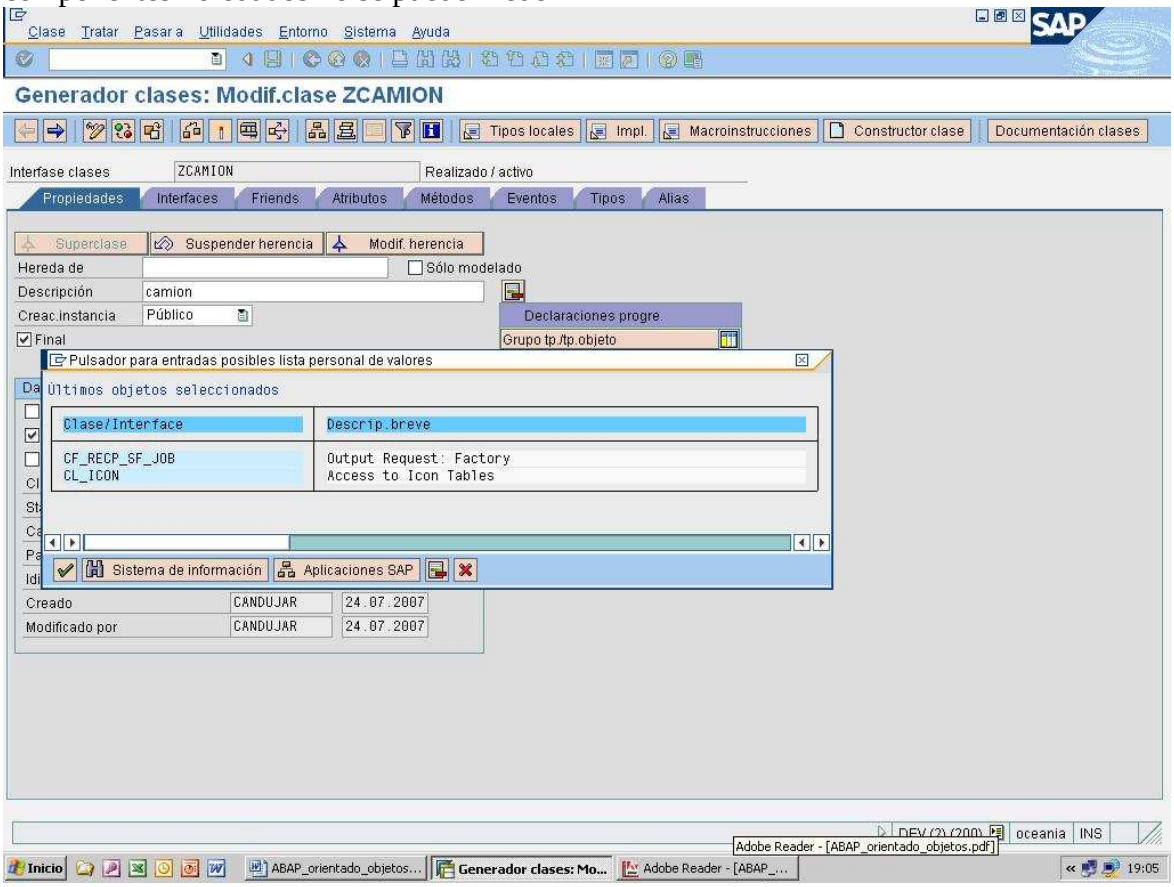
La herencia permite derivar clases de clases ya existentes. La nueva clase contiene un mayor rango de funciones específicas que su superclase. Esto se puede hacer añadiendo nuevos componentes a la subclase y redefiniendo métodos heredados de la superclase.

Para crear una subclase directa de una clase desde el editor de clases se escoge la pestaña Atributos (datos básicos) y dentro de ella subclase (siempre que la clase no sea final). Aparece el siguiente cuadro de diálogo.

La clase se crea ahora como una clase normal. La subclase hereda todos los componentes de la superclase excepto los métodos constructores. Las interfaces implementadas en la superclase también lo son en la subclase.

Los cambios en las subclases son aditivos, esto es, no se puede borrar un componente de una clase si ha sido heredado de una superclase. Una clase se puede ampliar de varias maneras:
- Añadiendo nuevos componentes.
- Redefiniendo métodos heredados.

Sólo se pueden redefinir métodos dependientes de instancia. Los atributos, los métodos estáticos y otros componentes heredados no se pueden redefinir



Se pueden definir nuevos componentes en las tres zonas de visibilidad (público, protegido o privado) de una subclase teniendo en cuenta que los nombres han de ser distintos entre los componentes heredados y creados en la subclase ya que si no existirán conflictos entre ellos.

9. Formularios

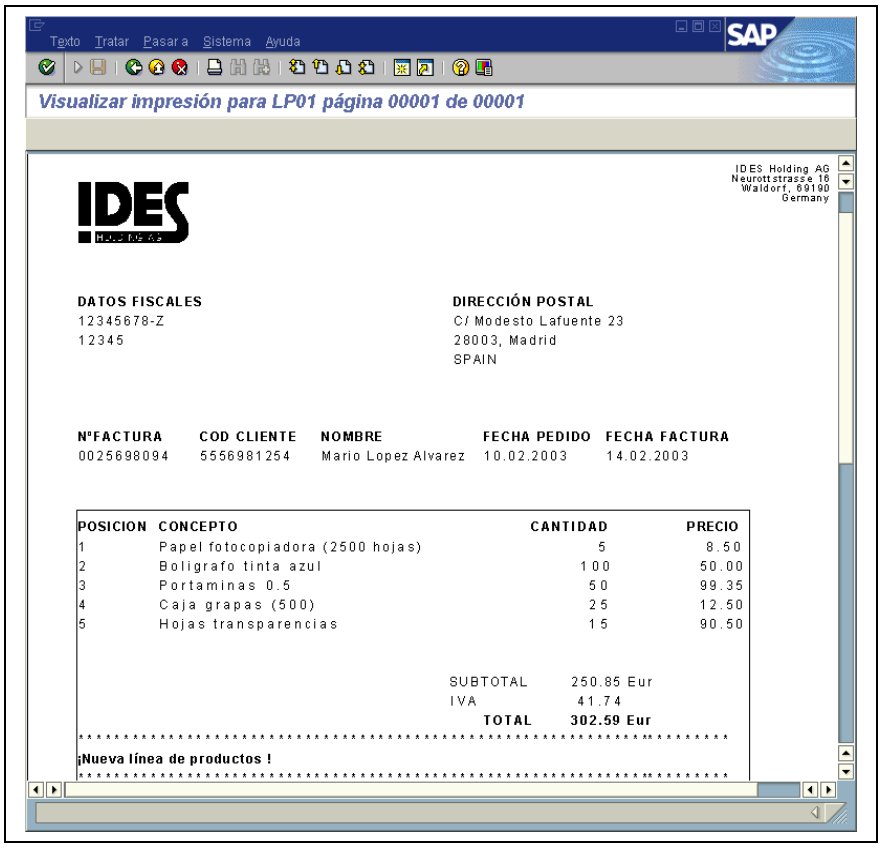
9.1 Introducción

Las empresas necesitan imprimir habitualmente informes, facturas, pedidos, etc. con un formato predefinido. Estos documentos pueden contener textos que se mantienen constantes entre documentos, pero también puede haber datos variables. En muchas ocasiones la creación de estos documentos se produce en masa creando múltiples copias de un mismo tipo de documento pero con propiedades distintas. De esta forma, la creación de los documentos como facturas, pedidos etc. se puede realizar de forma automática. SAPscript resuelve todos estos problemas con una plataforma común.

La creación de los documentos implica a varios elementos. Desde el punto de vista del programador SAPscript se puede dividir en formularios y programas de control de formularios. En realidad están implicados también el diccionario de SAP y el composer (un elemento que crea el formulario a partir de los elementos y de los requisitos del programa de control) Los formularios especifican la apariencia del texto en el documento (forman la plantilla del documento) y el programa de control especifica que valores contendrán los campos del formulario. De esta forma, para cambiar la apariencia de un documento sólo es necesario cambiar el formulario. Por el contrario, si se desea cambiar el contenido del documento es necesario cambiar los elementos de texto así como el programa de control del formulario.

El programa de control del formulario controla la salida del documento hacia la impresora, fax, pantalla, número de copias... Este programa selecciona los datos que se van a mostrar en el formulario del diccionario de ABAP o de las entradas del usuario, así como el formulario controlando los textos que se deben imprimir, su secuencia y frecuencia.

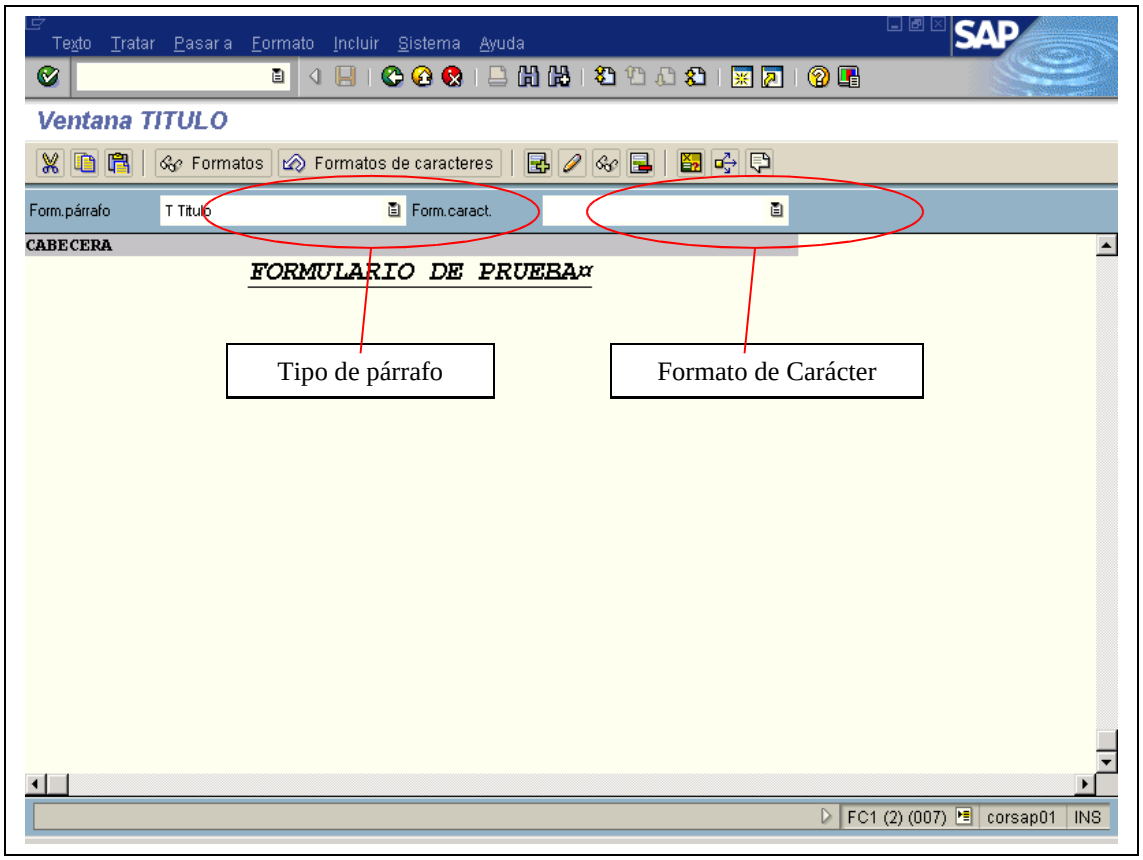
La apariencia final del documento depende de la interacción del programa de control con el formulario. El programa de control inicializa y finaliza el proceso de impresión, transfiriendo los comandos de SAPscript al composer. El composer formatea el documento a partir de la información del diseño del formulario especificado en el programa de control. Si el documento contiene variables, el composer reemplaza el contenido de estas en su posición (fecha, hora...) Ejemplo de una factura:



9.2 Editores de formularios

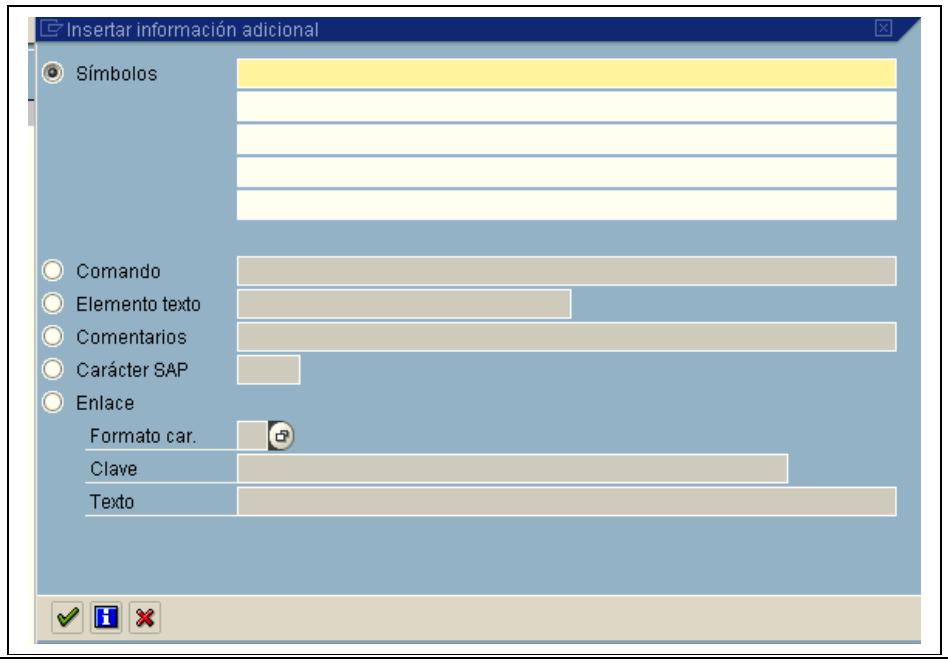
9.2.1 Editor Gráfico

Si entramos al editor de texto, seleccionando primero una ventana y pulsando luego en Elementos de texto veremos la siguiente pantalla:



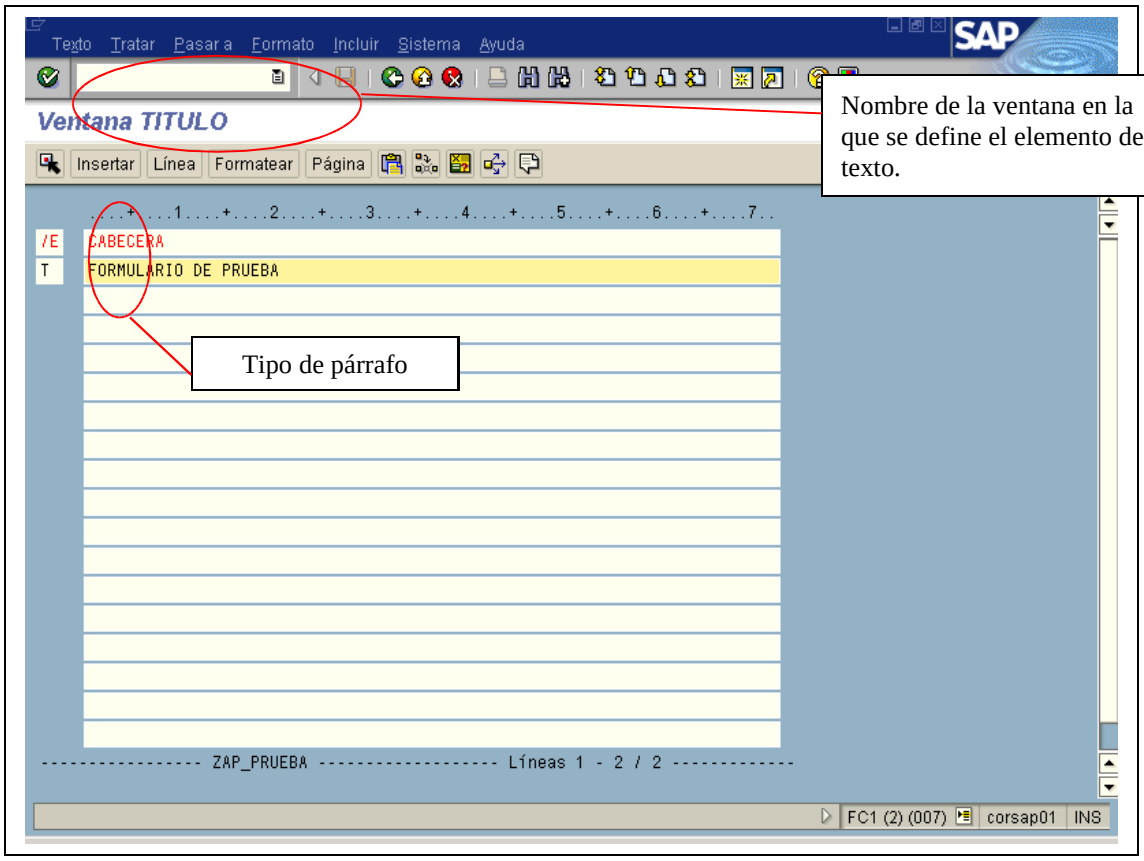
En el campo de Tipo de Párrafo podemos escoger entre los formato de párrafo que hemos definido en dicha ventana al igual que en el campo de Formato de Carácter podemos escoger entre los formatos de carácter que nos hemos definido.

Para introducir comandos, símbolos, elemento de textos, etc. dentro del formulario en el menú Tratar → Comando → Insertar Comando nos permite insertar dicha información adicional.

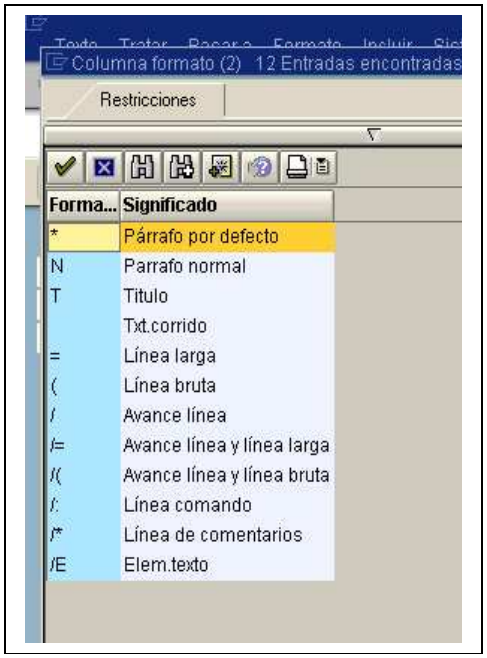


9.2.2 Editor Alfanumérico

Para pasar al editor de texto alfanumérico, desde el menú Pasar seleccionar la opción de Cambiar editor.

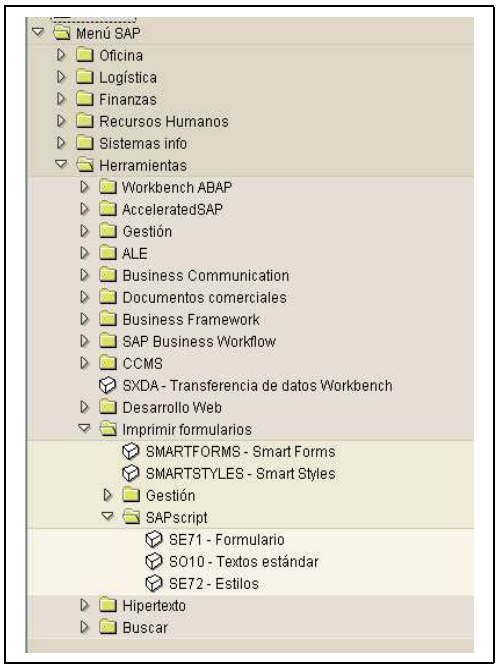


A diferencia del editor gráfico en el tipo de párrafo a parte de tener nuestro formato de párrafo definido anteriormente también podemos informar si la línea en cuestión es un comando, comentario, etc.

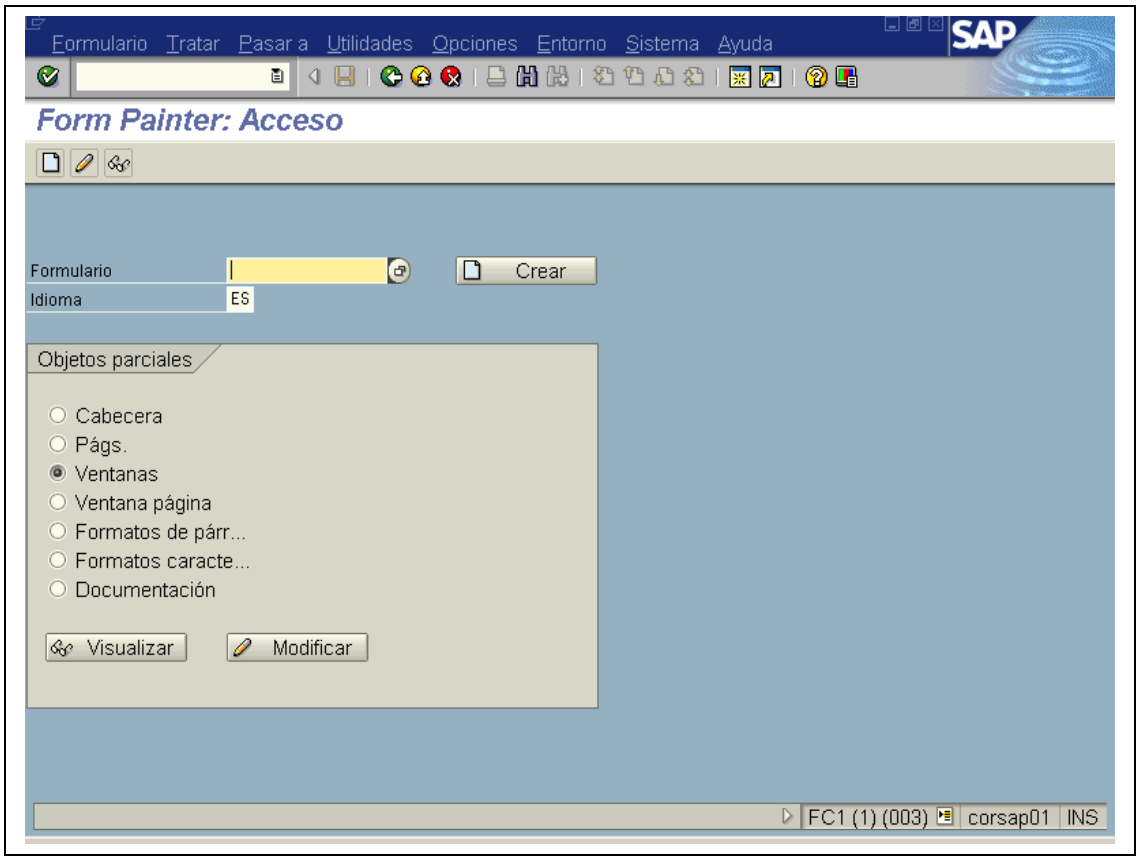


9.3 Estructura de un formulario

Para acceder al editor de formularios se sigue la ruta Herramientas -> Imprimir formularios->SAPscript-> Formulario, o directamente con la transacción SE71.



Accederemos a continuación a la pantalla Form Painter. Podemos crear un nuevo formulario, editar uno ya existente o modificarlo.



Nota: En el caso de no mostrar esta pantalla significará que estamos usando Form Painter gráfico. Es igual que el modo alfanumérico pero más visual. No obstante aquí seguiremos el modo alfanumérico accediendo por el menú->Opciones->Form Painter y desmarcando el flag de modo gráfico.

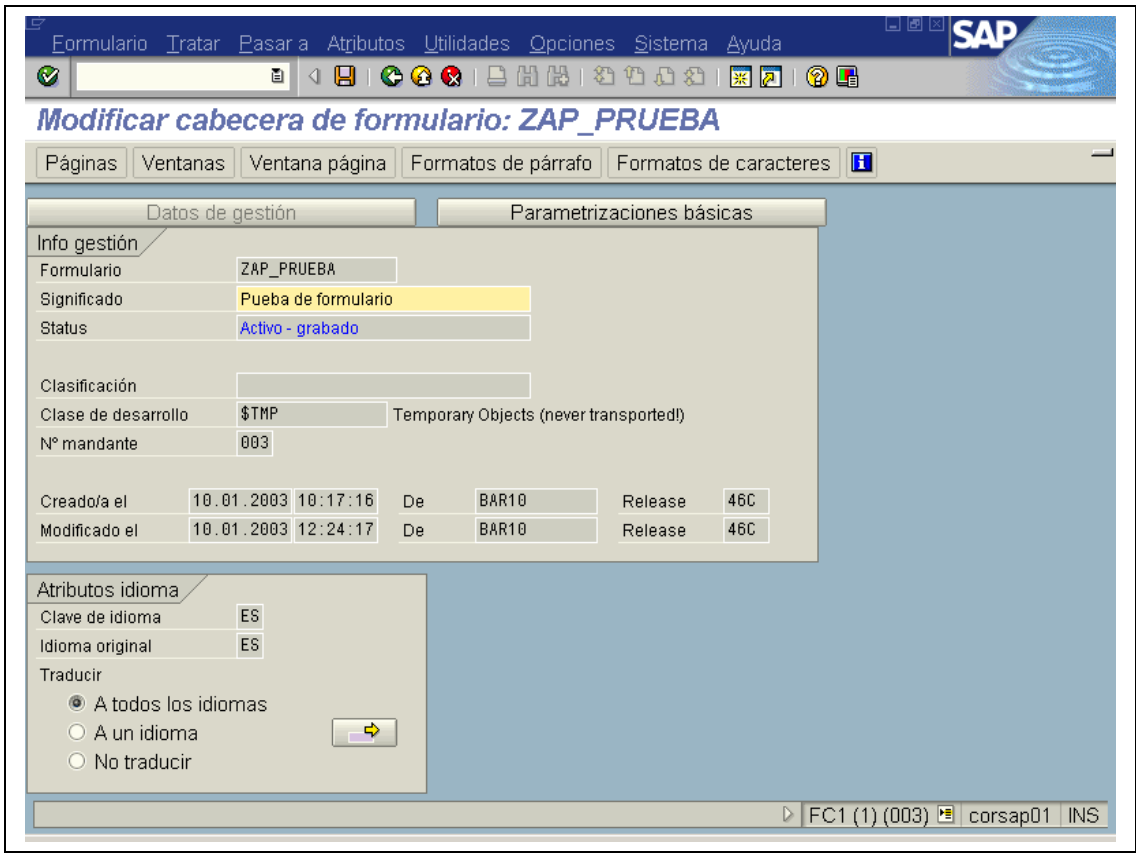
Un formulario especifica la disposición de los siguientes elementos en las páginas de un documento:

- Cabecera
- Páginas
- Ventanas
- Ventana página
- Formatos de párrafo
- Formatos caracteres
- Elementos de texto

9.3.1 Cabecera

La cabecera de un formulario consiste en atributos globales del formulario. Estos pueden ser *datos de gestión* (Nombre del formulario, descripción, clase de desarrollo...) y *parametrizaciones básicas* (formato de página, fuente por defecto, párrafo por defecto...). Cuando se crea un formulario, se accede directamente a la cabecera, la cual sirve para definir los datos generales del formulario.

- *Datos de gestión*

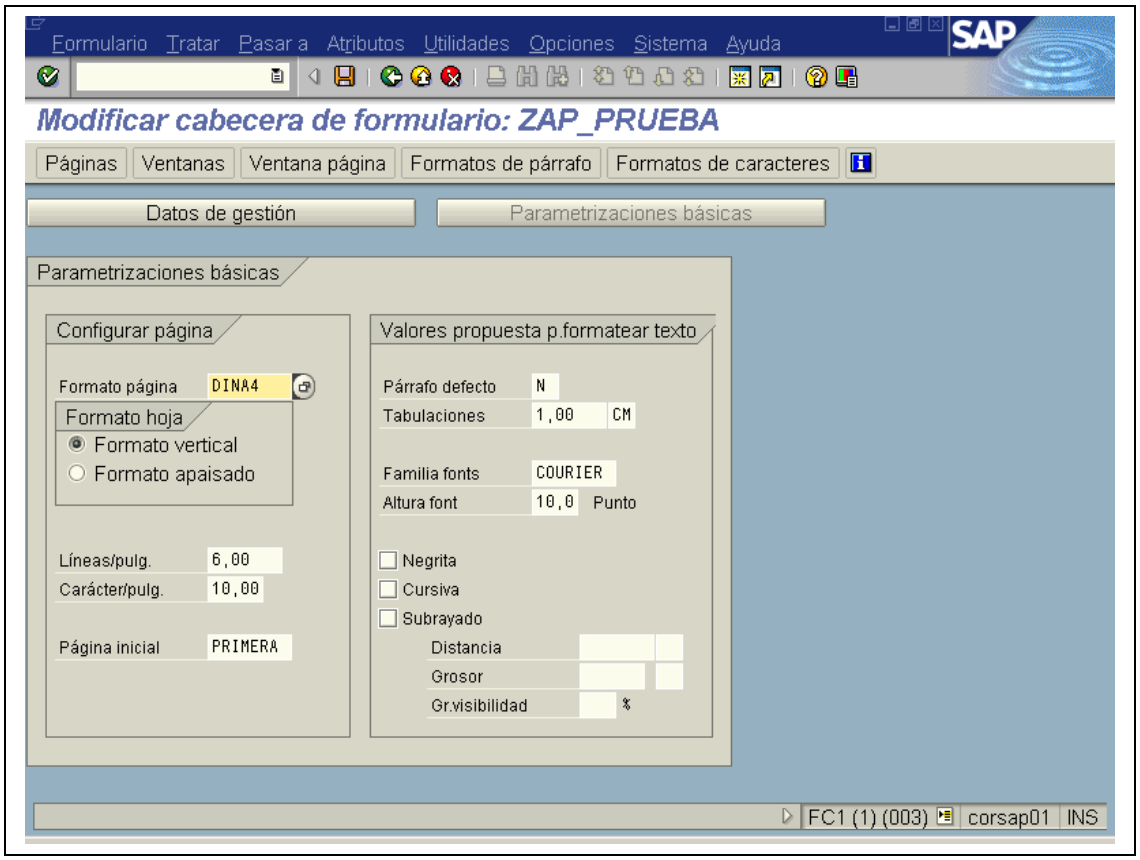


Está ventana esta conformada por dos subdivisiones, info gestión y atributos idioma.

- Info gestión:
  Formulario: El nombre de un formulario no deberá tener más de 16 posiciones y sólo deberán utilizarse letras mayúsculas y cifras, comenzando siempre por una letra.
  Significado: Descripción breve del formulario

- Atributos idioma:
  Se define el idioma principal y si el formulario podrá ser traducido a otros idiomas, o solo podrá mostrarse en el idioma original.

Parametrizaciones básicas



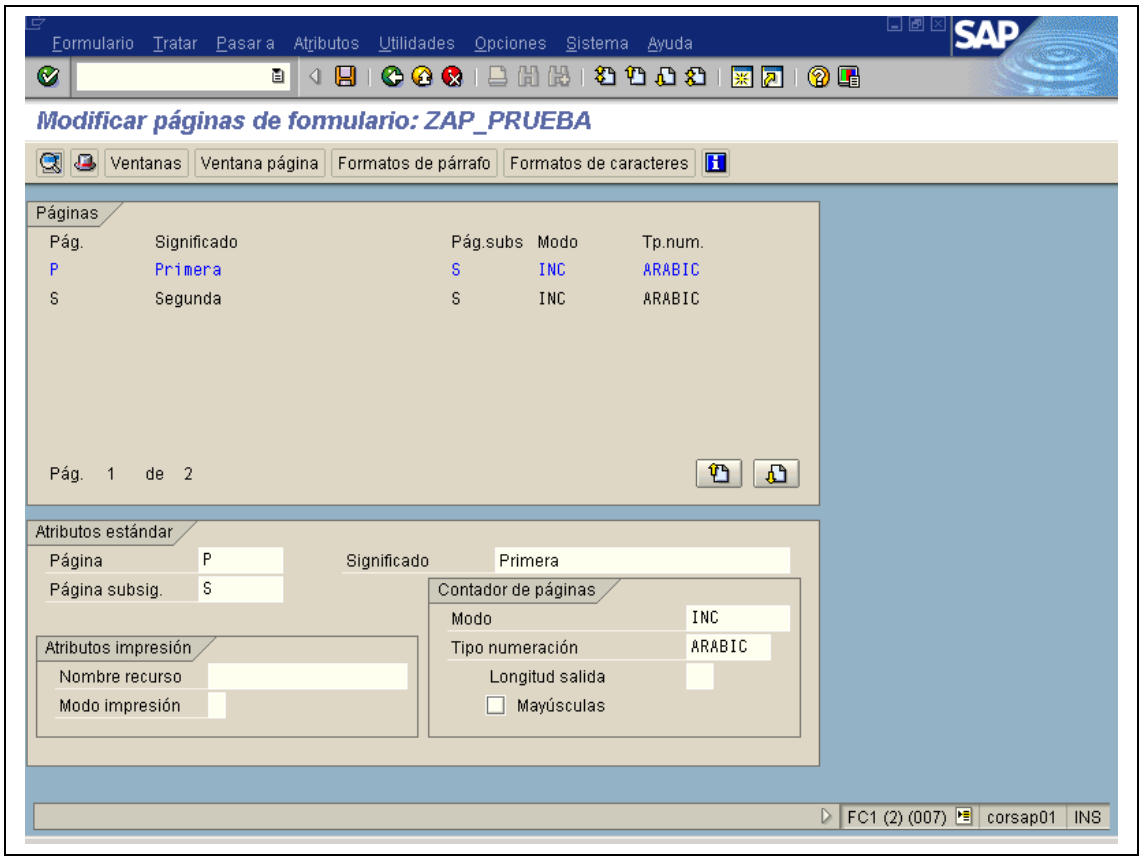
En la ventana de parametrizaciones básicas se distinguen dos divisiones, una conformada por la configuración de la página y la otra con los valores propuestos para el formato del texto

Configurar página:
Formato de página: Indica el tamaño del papel.
Formato hoja: Indica la orientación del papel.
Líneas/pulg: Indica la cantidad de líneas de impresión por pulgada.
Carácter/pulg: Indica la cantidad de caracteres de impresión por pulgada.
Página inicial: Primera página que se va ha imprimir de nuestro formulario.

Valores propuesta p. formatear texto: Se especifican los valores tomados por defecto.
Párrafo por defecto: Si se indica '*' para el párrafo en el editor SAPscript, el sistema tomará el párrafo default para editar el texto.
Tabulaciones: Distancia entre las tabulaciones en un formulario.
Familia fonts, altura font...: Definen el formato de carácter por defecto.

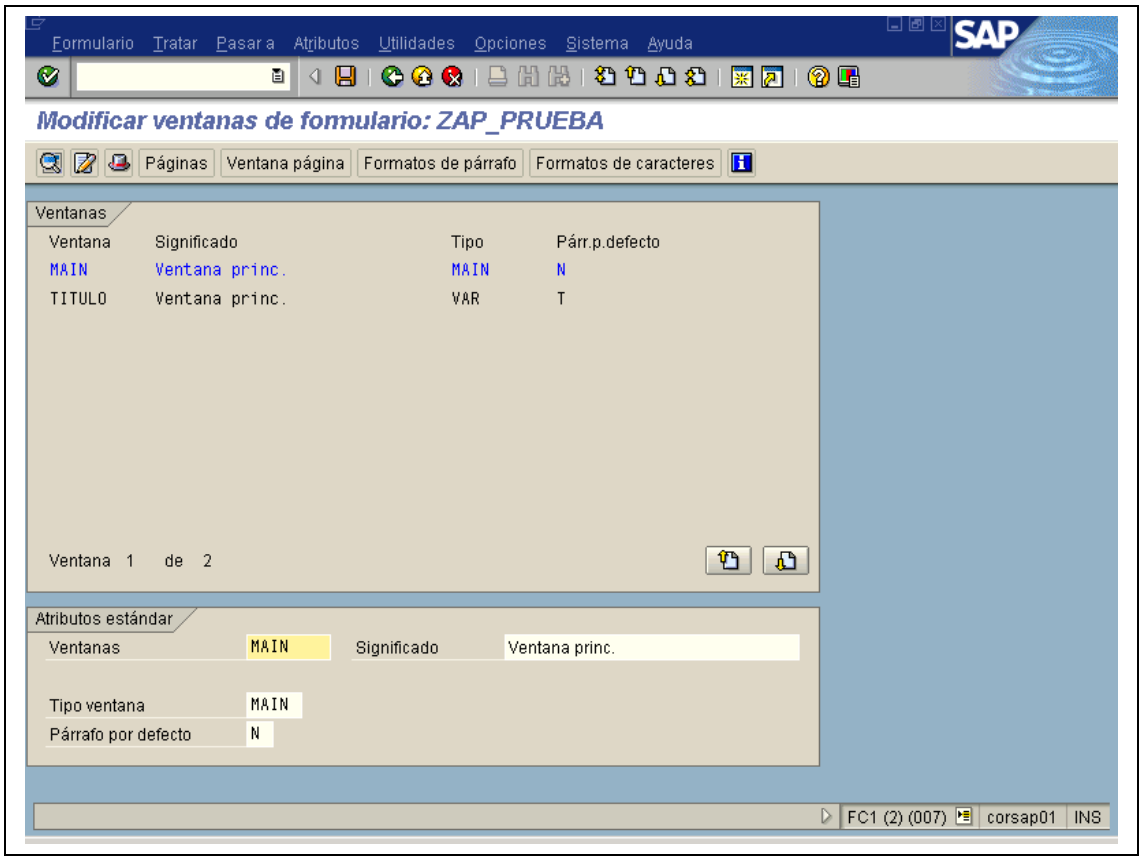
9.3.2 Páginas

Representan las distintas páginas del documento. Estas suelen tener distinto aspecto unas de otras, la primera página de un fax contiene información distinta a la que se mostrara en las páginas siguientes. Para cada formulario se ha de definir al menos una página. En este apartado daremos una descripción a la página, indicaremos cual es la página siguiente y cómo se actualiza el contador de páginas, así como los atributos de impresión como son el modo de impresión (Por defecto, SYMPLEX, DUPLEX o TRIPLEX) y el nombre del recurso que será la bandeja de la impresora de donde tomará el papel (TRY01, TRY02, TRY03).



9.3.3 Ventanas

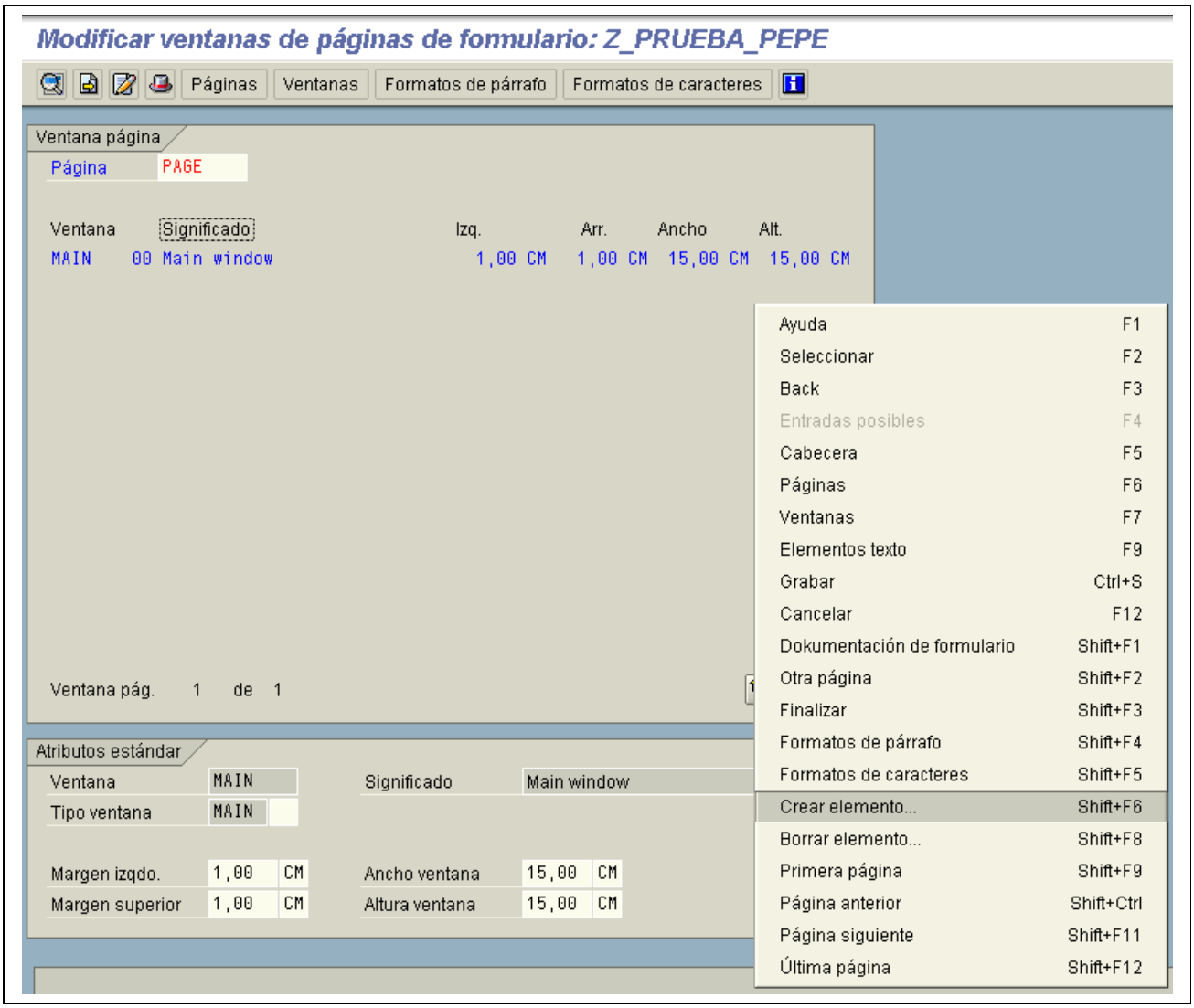
Las ventanas representan áreas que se posicionarán sobre las páginas. En ellas pondremos el párrafo por defecto, el nombre de la ventana, su significado y el tipo de ventana.



Hay 4 tipos de ventanas:
MAIN: Es la ventana principal en la que se escribirá el texto variable, como podría ser el cuerpo de una carta. Esta ventana puede extenderse a más de una página.
VAR: Es una ventana cuyo contenido puede variar. Estas ventanas deben definirse en cada página.
CONST: Define una ventana cuyo contenido no cambia.
GRAPH: Define una ventana con una imagen gráfica.

9.3.4 Ventana página

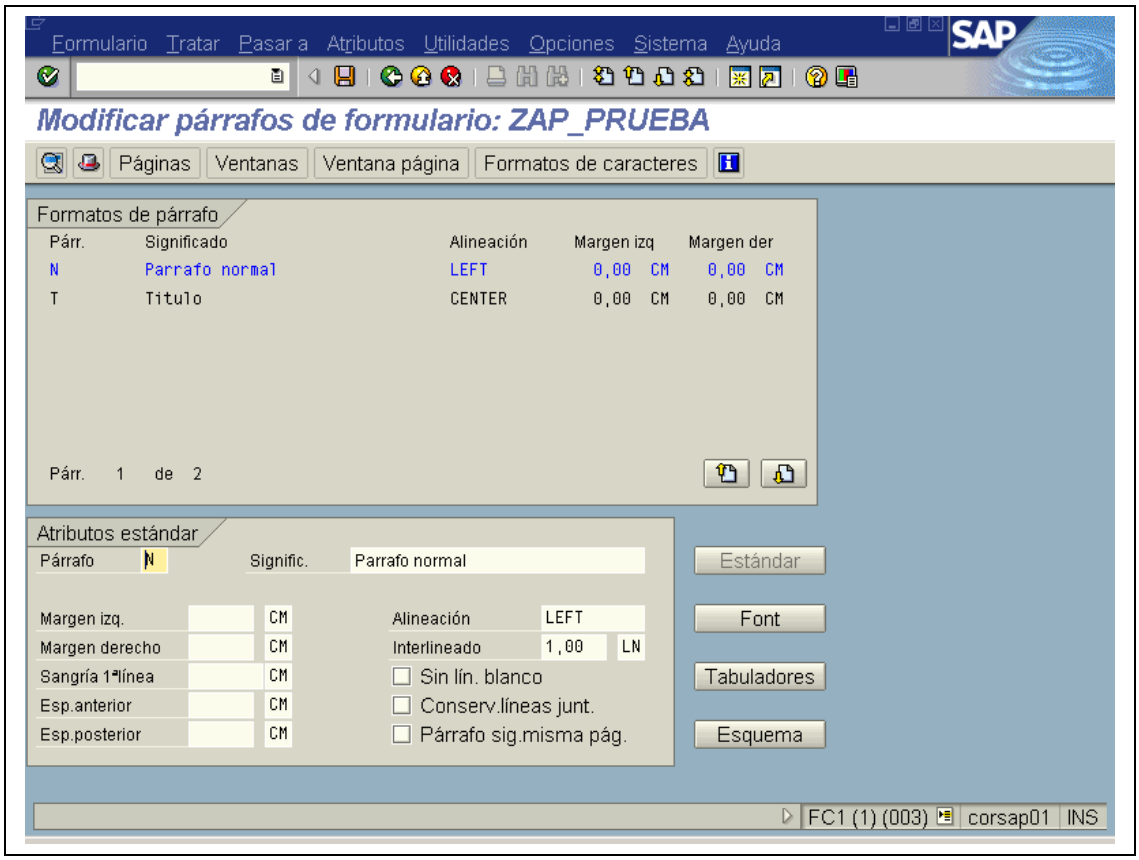
En este apartado, se especifica la posición y tamaño de las ventanas en cada página. Para añadir página, se realiza por medio del botón derecho del ratón en la opción crear elemento.



9.3.5 Formatos de párrafos

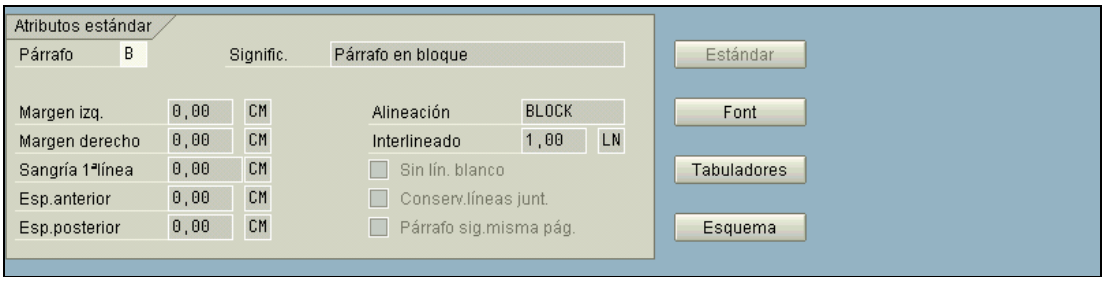
Los formatos de párrafos definen las características de estos, esta pantalla esta dividida en dos partes. En la zona superior se muestra una lista con todos los formatos de párrafo para el formulario, observándose las características generales de cada párrafo.

Los atributos de cada párrafo están divididos en 4 partes:



Para la creación de un nuevo párrafo se añade en la casilla de párrafo en la zona de atributos se añade el nombre que le identificara con uno o dos caracteres

- *Datos estándar*

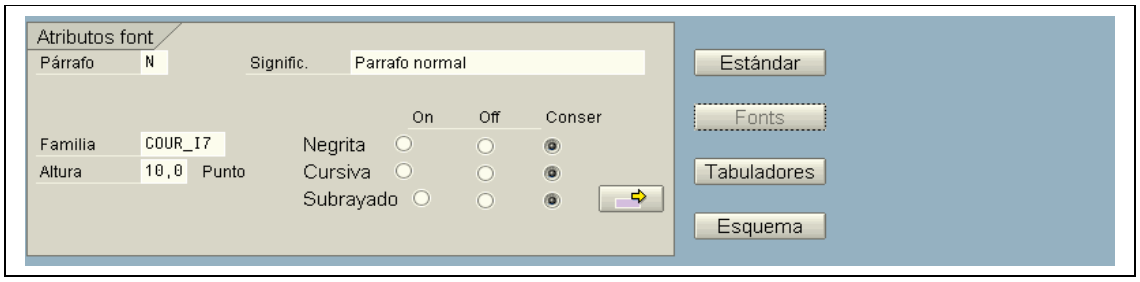


Se define las características propias del párrafo, como márgenes, sangría y alineación.

Párrafo: Define el nombre con el cual se identificara con posterioridad las distintas características del párrafo creado.
Significado: Breve descripción identificativa del párrafo.
Margen izquierdo y derecho: Distancia con la ventana que contenga el párrafo.
Sangría 1ª línea: Sangría.
Esp. anterior / posterior: Espacio de comienzo con respecto al párrafo anterior / posterior.
Alineación: Alineación del párrafo.
Interlineado: Espacio entre cada línea.

- *Datos de Fuentes*

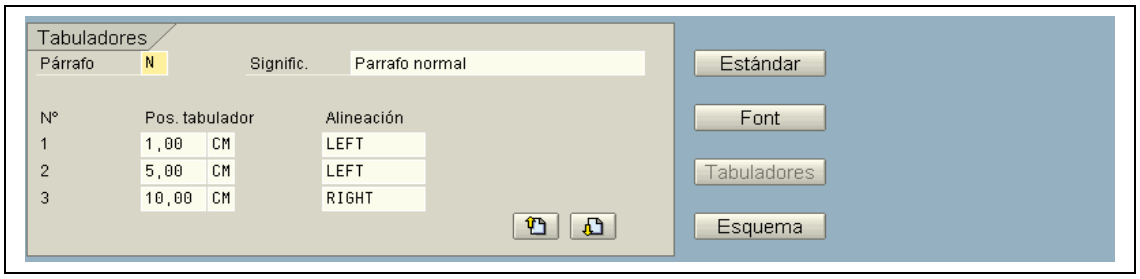
  Contiene las características de la fuente así como su formato.



  Familia: fuente que se usará en el párrafo.
  Altura: Tamaño de la fuente a utilizar
  Negrita, cursiva y subrayado: Como indica el titulo permite la opción de que el párrafo este formateado con alguna de estas características si se marca la opción “On” quedarán activadas, si se selecciona la opción “Conser” se usara el valor que tenía el texto anterior.

- *Datos de tabuladores*

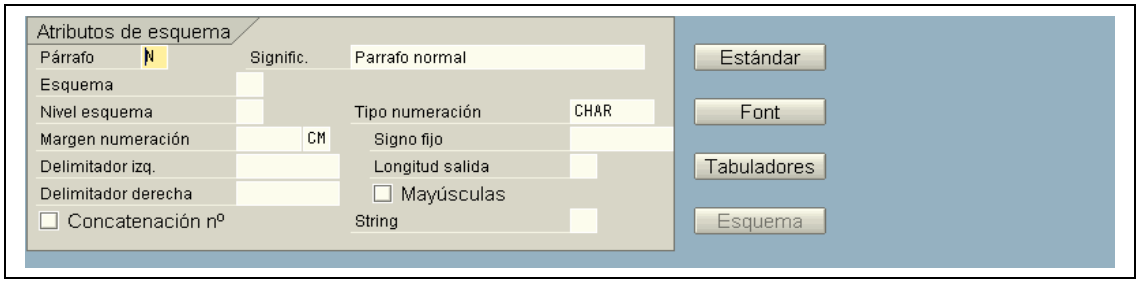
  Aquí se definen las distintas posiciones de tabulación que necesitemos para cada párrafo.



  Le deberemos indicar una posición, ya sea en centímetros (CM), caracteres (CH), milímetros (MM), puntos (PT) y un tipo de alineación: izquierda (LEFT), derecha (RIGHT), centrado (CENTER), al signo (SIGN) o a la coma decimal (DECIMAL)

- *Datos de esquema*

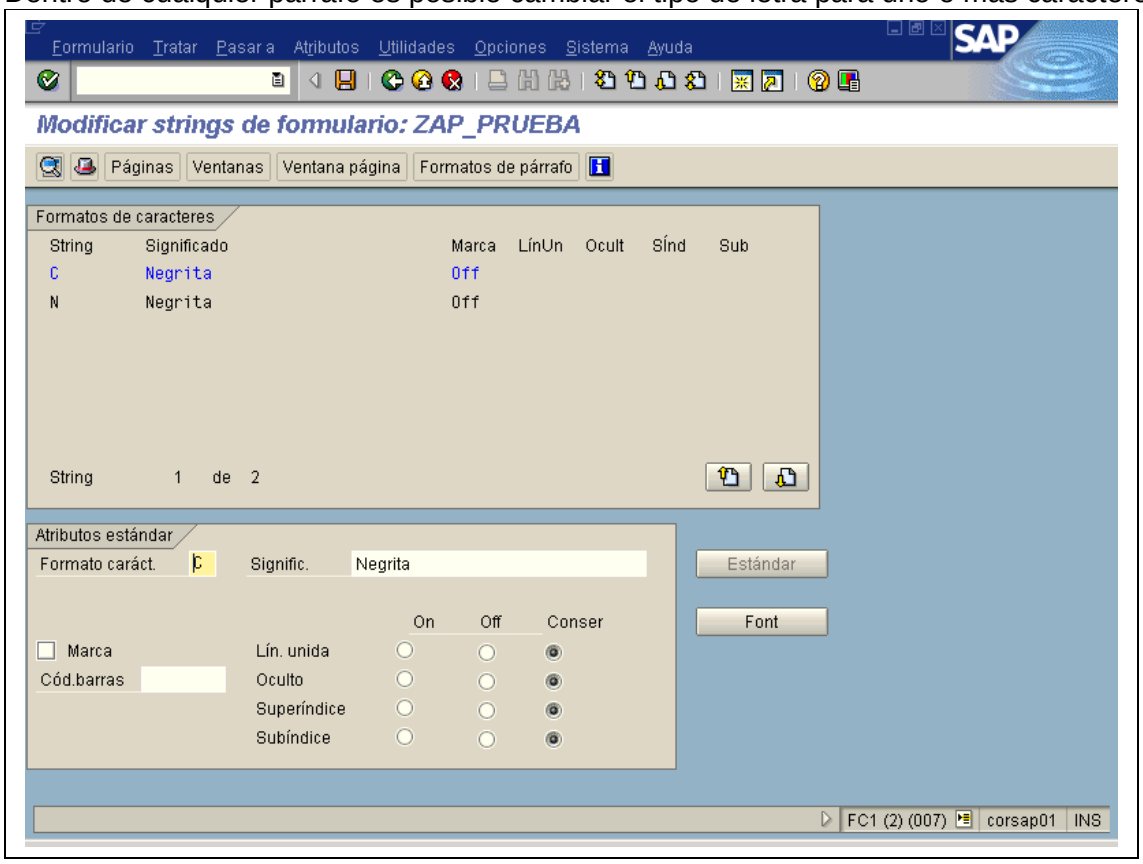
  Es posible definir numeración y marcas automáticas de forma que podamos estructurar texto en capítulos, subcapítulos y secciones.



  El signo fijo será el carácter que se antepondrá siempre al párrafo, si el signo es ‘_’ está se convertirá en espacio en blanco.

9.3.6 Formatos de caracteres

Dentro de cualquier párrafo es posible cambiar el tipo de letra para uno o más caracteres.

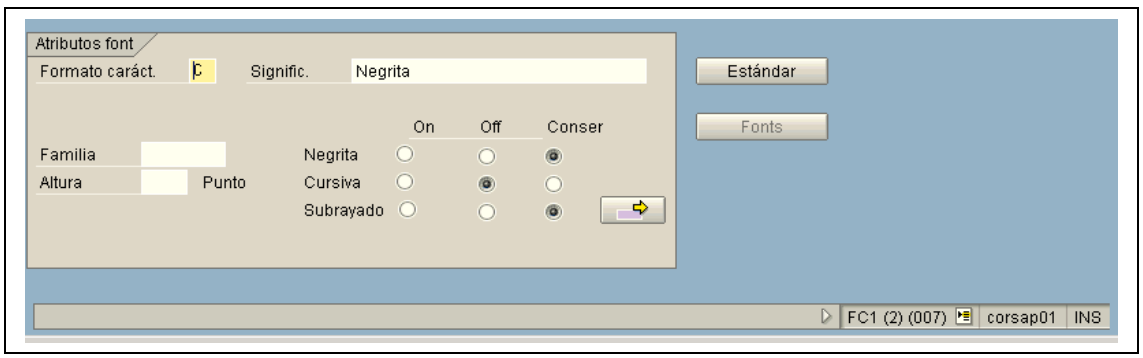


- *Atributos estándar*

  Formato carácter.: Nombre con un máximo de dos letras con el cual se identificara el tipo de formato de carácter creado.
  Signific.: Breve descripción que defina el formato de caracteres.
  Cód. Barras: Se introduce el código numérico de barras para la posterior visualización del mismo.
  Lín. Unida, oculto, superíndice, subíndice: Se especifican las distintas opciones que tendrá el formato de caracteres. “Conser” conservara los parámetros especificados en el texto anterior.

- *Atributos font*

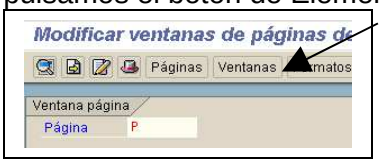
  Contiene los datos de la fuente que se usara en el formato de carácter.



Las características en los atributos Font son los mismos que para los formatos de párrafo.

9.3.7 Elementos de texto

Los elementos de texto son componentes individuales de una ventana. Puede haber más de un elemento de texto por ventana y se distinguen dos tipos. Los elementos de texto pueden contener variables e instrucciones de control SAPscript. Para acceder a ellos nos situamos en la pantalla de Ventana Pagina, seleccionamos la ventana donde queremos insertar los elementos de texto y pulsamos el botón de Elementos de texto (F9).



El programa de impresión accede a los elementos de texto por nombre, los formatea e imprime en la ventana correspondiente.

9.4 SAPScript

9.4.1 Cajas, líneas y sombreados

Se pueden dibujar cajas y líneas en SAPScript mediante los siguientes comandos:
BOX: Dibuja una caja o una línea.
POSITION: Especifica el punto inicial de una caja o línea
SIZE: Especifica la anchura o altura de una caja.

Ejemplo:
Creamos una nueva ventana de tipo CONST de idénticas proporciones que la ventana MAIN que dibuje una caja que rodee las dos columnas de la ventana MAIN del ejemplo.

```
/:  BOX XPOS '0.5' CM WIDTH 8 CM HEIGHT 15 CM FRAME 10 TW.
/:  BOX XPOS '8.5' CM WIDTH 8 CM HEIGHT 15 CM FRAME 10 TW.
```

9.4.2 Comandos de control

SAPScript dispone de una serie de comandos que permiten obtener un control total sobre el texto. Estos comandos son introducidos en el editor de textos igual que una línea normal excepto que el deberemos seleccionar el párrafo /:

NEW-PAGE: Provoca el salto automático de página
PROTECT – ENDPROTECT: Se asegura que el texto introducido entre estos dos comandos aparezca siempre en una misma página.
NEW-WINDOW: Llama automáticamente a la siguiente ventana MAIN definida en una misma página.
DEFINE: Permite crear una constante con un valor dado.
SET DATE MASK: Define el formato de los campos fecha.
SET TIME MASK: Define el formato de los campos hora.
SET COUNTRY: Define el formato de ciertos campos como el punto para los millares adaptados a un país específico.
SET SIGN: Indica la posición del signo.
RESET: Inicializa el contador de un párrafo con numeración.
INCLUDE: Inserta el contenido de otro texto en el texto actual.
STYLE: Cambia el contenido del estilo actual del texto.

ADDRESS – ENDADDRESS: Formatea una dirección de acuerdo con las convenciones postales del país definido por el parámetro COUNTRY.
TOP – ENDTOP: Especifica líneas de texto que siempre aparecerán en la parte superior de la ventana MAIN.
BOTTOM – ENDBOTTOM: Especifica líneas de texto que aparecerán en la parte inferior de la ventana MAIN.
IF – ENDIF: Permite especificar que líneas debería imprimirse cuando se cumplan ciertas condiciones.
CASE: Cubre el caso de múltiples comandos IF anidados.
PERFORM: Permite llamar una rutina de un programa ABAP
PRINT-CONTROL: Llama directamente ciertas funciones de la impresora.
BOX, POSITION, LINE: Dibuja líneas y cajas.
HEX – ENDHEX: Envía a la impresora órdenes en el lenguaje que utiliza la impresora.
SUMMING: Acumula un valor total para un símbolo de programa.

9.4.3 Símbolos SAPScript

La información variable se introduce en los formularios SAPScript mediante “símbolos” o variables que SAP reconoce porque van rodeadas de ‘&’. Ej.: &symbol&.

Símbolos del sistema: variable como fecha, hora…
Símbolos de programa: variables almacenas en aplicaciones SAP como campos del diccionario de datos o variables globales de los programas.
Símbolos estándar definidos en la tabla TTDTG: El valor de estos símbolos es dependiente del lenguaje y puede contener hasta 60 caracteres. SAP mantiene esta tabla con valores estándar.
Símbolos de texto: Aquellos que no corresponden a los tipos de símbolos definidos anteriormente. Son definidos por el usuario en el editor de texto, eligiendo Incluir -> Símbolos ->Texto. O bien definiéndolos con el comando INCLUDE.

9.4.4 Símbolos del sistema

&DATE&	Fecha del sistema. Se imprimirá de acuerdo al tipo de SET DATE MASK definida anteriormente.
&DAY&	Día del sistema.
&MONTH&	Mes
&YEAR&	Año
&NAME_OF_DAY&	Nombre del día de la semana del recogido en el SET DATE MASK.
&NAME_OF_MONTH&	Nombre del mes
&TIME&	Hora según el formato especificado en el SET TIME MASK.
&HOURS&	Hora
&MINUTES&	Minuto
&SECONDS&	Segundo
&PAGE&	Número de la página actual
&NEXTPAGE&	Número de la página siguiente a la actual
&DEVICE&	Tipo del dispositivo de salida (PRINTER, SCREEN, TELEX, ABAP).
&SPACE&	Espacio.
&ULINE&	Línea de subrayado
&VLINE&	Línea vertical

9.4.5 Campos generales de SAPScript

&SAPSCRIPT-SUBRC& Recibe un valore después de ejecutar un comando INCLUDE. (0 si lo encuentra, 4 si no).
&SAPSCRIPT-DRIVER& Nombre del dispositivo de salida (POST, HPL2, PRES).
&SAPSCRIPT-FORMPAGES& Nº total de páginas del form.
&SAPSCRIPT-JOBPAGES& Nº total de páginas de todos los formularios contenido en la petición de impresión actual.
&SAPSCRIPT_CONTER_X& (x = 0…9) Representan 10 variables de contador que se puede utilizar en el texto y formularios para cualquier propósito.

9.4.6 Opciones de formato de los símbolos

Desplazamiento	&symbol+offset&
Longitud de salida	&symbol(length)&
Omisión del signo	&symbol(S)&
Mostrar el signo por la izquierda	&symbol(<)&
Mostrar el signo por la derecha	&symbol(>)&
Omitir los ceros iniciales	&symbol(Z)&
Comprimir los espacios	&symbol(C)&
Número de decimales	&symbol(.2)&
Omitir el indicador de miles	&symbol(T)&
Especificar exponente	&symbol(E2)&
Alinear a la derecha	&symbol(8R)&
Rellenar de caracteres	&symbol(Ff)&
Suprimir valores iniciales	&symbol(I)&
Ignorar rutinas de conversión	&symbol(K)&
Cambiar valor de un contador	&SAPSCRIPT_COUNTER_X(+)&
Textos precedentes	&’pre-text’symbol’post-text’&

9.4.7 Formularios en varios idiomas

Una vez tenemos creado el formulario en un idioma padre, podemos crearlo en otros idiomas. Para ello en la pantalla de mantenimiento de formularios, seleccionamos el formulario que acabamos de crear, seleccionamos el nuevo lenguaje del formulario y seleccionamos crear. Veremos que trabajamos con una copia del formulario anterior en el que lo único que deberemos hacer será traducir los textos y adaptar el formulario, si fuera necesario, a las peculiaridades del nuevo idioma.

Para llamar al formulario en distintos idiomas se realiza mediante la función de abrir formulario, indicando el idioma requerido. Si el formulario no existiera en ese idioma, se abrirá el formulario en el idioma padre.

```
CALL FUNCTION 'OPEN_FORM'
   EXPORTING
      FORM     = 'Z_PRUEBA'
      LANGUAGE = P_IDIOMA
      OPTIONS  = ITCPO
      DEVICE   = 'PRINTER'
      DIALOG   = 'X'
   EXCEPTIONS
```

```
OTHERS   = 1.
```

9.4.8 Inclusión de gráficos

Para incluir gráficos en un formulario, primero este debe estar grabado en SAP en formato TIFF, BMP o en forma de elemento de texto. Luego se incluye en la ventana deseada a través del menú Incluir -> Función gráfica y seleccionando el gráfico. Esto generará una línea de comando como la siguiente:

```
BITMAP IDES_LOGO OBJECT GRAPHICS ID BMAP TYPE BMON DPI 300
```

9.5 Programa de impresión del formulario

Una vez finalizado el diseño del formulario es necesario crear un programa que gestione la impresión del mismo. Para ello hay una serie de funciones Standard de SAP que gestionan todos los parámetros de salida. De todas ellas, las más relevantes son:
OPEN_FORM
WRITE_FORM
CLOSE_FORM

OPEN_FORM: Este modulo de función abre un formulario para su impresión. Esta función ha de ejecutarse antes que cualquier otra función que actúe sobre el formulario (WRITE_FORM, START_FORM, CONTROL_FORM...). Cada vez que se utiliza la función OPEN_FORM es necesario cerrar el formulario (CLOSE_FORM) para que este se imprima. Dentro de un mismo programa puede haber varios pares de llamadas a las funciones OPEN_FORM y CLOSE_FORM.

CLOSE_FORM: Cierra un formulario abierto previamente con la función OPEN_FORM.

WRITE_FORM: El sistema muestra un elemento de texto determinado del formulario. El elemento de texto se especifica en el parámetro exportado ELEMENT. En el parámetro WINDOW se puede especificar el nombre de la ventana de salida.

Estructura ITCPO representa los parámetros de control del formato de salida. Esta estructura se puede utilizar en los módulos de función PRINT_TEXT y OPEN_FORM en el parámetro OPTIONS.

TDPAGESLCT	SAPscript: seleccionar página de impresión
TDPREVIEW	SAPscript: habilitar vista previa
TDNOPREV	SAPscript: deshabilitar vista previa
TDNOPRINT	SAPscript: deshabilitar impresión desde vista previa
TDTITLE	SAPscript: Título de la pantalla de selección
TDPROGRAM	SAPscript: nombre del programa de símbolos de sustitución
TDTEST	SAPscript: visualización previa
TDIEXIT	SAPscript: volver inmediatamente después de la impresión
TDGETOTF	SAPscript: valor de retorno te la tabla OTF, no hay impresión
TDSCRNPOS	SAPscript: posición del OTF en la pantalla
TDDEST	Spool: nombre del dispositivo de salida

TDPRINTE	Spool: nombre del tipo de dispositivo
TDCOPIES	Spool: numero de copias
TDNEWID	Spool: nueva petición
TDIMMED	Spool: petición de impresión inmediata
TDDELETE	Spool: borrar petición después de la impresión
TDLIFETIME	Spool: tiempo de retención de la petición
TDDATASET	Spool: identificación de la petición
TDSUFFIX1	Spool: primer sufijo de la petición
TDSUFFIX2	Spool: segundo sufijo de la petición
TDAUTORITY	Spool: autorización para la petición
TDARMOD	Spool: modo de archivo
TDCOVER	Spool: imprimir portada
TDCOVTITLE	Spool: portada: título
TDRECEIVER	Spool: portada: nombre del destinatario
TDDIVISION	Spool: portada: nombre de la división
TDSCHEDULE	SAPcomm: tipo del tiempo de envío estimado
TDSENDDATE	SAPcomm: fecha de envío solicitada
TDSENDTIME	SAPcomm: hora de envío solicitada
TDTELELAND	SAPcomm: código del país destinatario
TDTELENUM	SAPcomm: número de marcación

Ejemplo de formulario

Formulario

```
---------------------------------------------------------------------------
Formulario          ZPRUEBA
---------------------------------------------------------------------------

Significado         Formulario de prueba

Atributos std.
  Página inicial      PRICIPAL
  Párrafo defecto     DF
  Tabulaciones        1,00 CM
  Formato página      DINA4
  Formato hoja        Formato vertical
  Líneas/pulg.        6,00
  Carácter/pulg.      10,00

Atributos font
  Familia fonts       COURIER
  Altura font         12,0 Punto
  Negrita             no
  Cursiva             no
  Subrayado           no

---------------------------------------------------------------------------
Carácteres    Atributos
---------------------------------------------------------------------------
```

```
 DF        Normal
          Atributos std.
            Marca           no
          Atributos font
            Familia fonts    HELVE
            Altura font      10,0 Punto

 NG        Negrita
          Atributos std.
            Marca           no
          Atributos font
            Familia fonts    HELVE
            Altura font      10,0 Punto
            Negrita          sí

--------------------------------------------------------------------------------
Párrafos     Atributos
--------------------------------------------------------------------------------

 AD        Dirección propia
          Atributos std.
            Interlineado     0.50 LN
            Alineación       Alin.derecha
          Atributos font
            Familia fonts    HELVE
            Altura font      8,0 Punto

 DF        Párrafo por defecto
          Atributos std.
            Interlineado     1.00 LN
            Alineación       alin.izq.
          Atributos font
            Familia fonts    HELVE
            Altura font      10,0 Punto

 NG        Negrita
          Atributos std.
            Interlineado     1.00 LN
            Alineación       alin.izq.
          Atributos font
            Familia fonts    HELVE
            Altura font      10,0 Punto
            Negrita          sí

--------------------------------------------------------------------------------
Ventanas     Atributos
--------------------------------------------------------------------------------

 ADDRESS    Dirección
            Tipo ventana     VAR
            Párr.p.defecto   DF

 CONTROL    Ventana de control
```

```
            Tipo ventana      VAR
            Párr.p.defecto    DF

 D_FISCAL   Datos fiscales
            Tipo ventana      VAR
            Párr.p.defecto    DF

 FACTURA    Datos Factura
            Tipo ventana      VAR
            Párr.p.defecto    DF

 LOGO       Ventana de logo
            Tipo ventana      CONST
            Párr.p.defecto    DF

 MAIN       Ventana princ.
            Tipo ventana      MAIN
            Párr.p.defecto    DF

 MARCO      marco de la ventana main
            Tipo ventana      CONST
            Párr.p.defecto    DF

 MY_ADD     Dirección propia
            Tipo ventana      CONST
            Párr.p.defecto    AD

 PAGO       Forma de pago
            Tipo ventana      VAR
            Párr.p.defecto    DF

---------------------------------------------------------------------------
Páginas      Atributos
---------------------------------------------------------------------------

 PRICIPAL   Primera página
          Atributos std.
            Pág.subsiguiente  SEGUNDA
          Atributos impresión
            Modo impresión    S
          Contad.pág.
            Modo              INC
            Tipo numeración   cifras árabes
          Ventana página
            MAIN              Margen izqdo.       1.25  CM
                              Margen superior     8.89  CM
                              Ancho ventana      18.27  CM
                              Altura ventana     12.47  CM
            ADDRESS           Margen izqdo.      10.49  CM
                              Margen superior     3.95  CM
                              Ancho ventana       9.14  CM
                              Altura ventana      1.98  CM
            CONTROL           Margen izqdo.      16.50  CM
```

```
                    Margen superior     22.00  CM
                    Ancho ventana        3.25  CM
                    Altura ventana       3.00  CM
          D_FISCAL          Margen izqdo.        1.25  CM
                    Margen superior      3.95  CM
                    Ancho ventana        8.89  CM
                    Altura ventana       1.98  CM
          FACTURA           Margen izqdo.        1.25  CM
                    Margen superior      6.91  CM
                    Ancho ventana       18.40  CM
                    Altura ventana       1.48  CM
          LOGO              Margen izqdo.        1.25  CM
                    Margen superior      1.25  CM
                    Ancho ventana        1.25  CM
                    Altura ventana       1.25  CM
          MARCO             Margen izqdo.        1.20  CM
                    Margen superior      8.84  CM
                    Ancho ventana       18.27  CM
                    Altura ventana      12.47  CM
          MY_ADD            Margen izqdo.       10.50  CM
                    Margen superior      1.25  CM
                    Ancho ventana        9.25  CM
                    Altura ventana       2.75  CM
          PAGO              Margen izqdo.        1.25  CM
                    Margen superior     22.00  CM
                    Ancho ventana       14.70  CM
                    Altura ventana       3.00  CM

  SEGUNDA     Página segunda y siguientes
            Atributos std.
              Pág.subsiguiente  SEGUNDA
            Contad.pág.
              Modo              INC
              Tipo numeración   cifras árabes
            Ventana página
              MAIN              Margen izqdo.        1.25  CM
                                Margen superior     11.50  CM
                                Ancho ventana       18.27  CM
                                Altura ventana      10.00  CM

--------------------------------------------------------------------------------
Elementos texto para ventanas:
--------------------------------------------------------------------------------

ADDRESS
*  <ng>DIRECCIÓN POSTAL</>
/: ADDRESS
*  &CALLE&
*  &CIUDAD&
*  &PAIS&
/: ENDADDRESS
*
```

```
CONTROL
*  <ng>CONTROL</>
*

D_FISCAL
*  <ng>DATOS FISCALES</>
*  &C_NIF&
*  &N_RESERVA&

FACTURA
NG NºFACTURA,,COD CLIENTE,,NOMBRE,, ,, ,,FECHA PEDIDO,,FECHA FACTURA
DF &CABECERA-NUMERO_FACTURA&,,&CABECERA-IDCLIENTE&,,&CABECERA-
DESCRIPCION&
   ,,&cabecera-fecha_pedido&,,&Cabecera-fecha_factura&

LOGO
*
/: BITMAP IDES_LOGO OBJECT GRAPHICS ID BMAP TYPE BMON DPI 300
*

MAIN
Elemento ITEM_HEADER
NG POSICION,,CONCEPTO,, ,, ,, ,, ,, ,, ,, ,, CANTIDAD,, ,,PRECIO

Elemento DATOS
DF &i_posicion-posicion&,, ,,&i_posicion-concepto(50)&,,
   &i_posicion-cantidad&,, ,,  &I_POSICION-PRECIO(5)&

Elemento PUBLICIDAD
/: PROTECT
NG ********************************************************************************
NG &C_MENSAJE&
NG ********************************************************************************
/: ENDPROTECT

Elemento TOTAL
DF
DF
DF ,, ,, ,, ,, ,, ,, ,, ,, ,, ,, SUBTOTAL,, &SUBTOTAL&
DF ,, ,, ,, ,, ,, ,, ,, ,, ,, ,, IVA,, ,, ,, &IVA&
NG  ,, ,, ,, ,, ,, ,, ,, ,, ,, ,,TOTAL,, &TOTAL&

MARCO
/: BOX XPOS 0 CM WIDTH '16.57' CM HEIGHT '12.47' CM FRAME 10 TW.

MY_ADD
AD IDES Holding AG
AD Neurottstrasse 16
AD Waldorf, 69190
AD Germany

PAGO
*  FORMA DE PAGO
```

Programa de control

```
REPORT zform .

*&---------------------------------------------------------------------*
* VARIABLES
*&---------------------------------------------------------------------*
DATA BEGIN OF itcpo.
       INCLUDE STRUCTURE itcpo.         "SAPscript Salida interfase
DATA END OF itcpo.

DATA: c_nif(10) TYPE c VALUE '12345678-Z',
      c_mensaje(74) TYPE c,
      n_reserva(5) TYPE n VALUE '12345',
      total(10),
      subtotal(10),
      iva(10).

c_mensaje =
'¡Nueva línea de productos !'.

DATA: BEGIN OF cabecera,
        numero_factura(10) TYPE c VALUE '0025698094',
        idcliente(10) TYPE c VALUE      '5556981254',
        descripcion(50) TYPE c VALUE 'Mario Lopez Alvarez',
        fecha_pedido LIKE sy-datum VALUE '20030210',
        fecha_factura LIKE sy-datum VALUE '20030214',
      END OF cabecera.

DATA: BEGIN OF i_posicion OCCURS 0,
        posicion(3) TYPE c,
        concepto(50) TYPE c,
        cantidad TYPE i,
        precio(5) TYPE c,
      END OF i_posicion.

* Variables para comunicarnos con el formulario
DATA: calle(40),
      ciudad(20),
      pais(20).


* Configuración de la impresora
PERFORM configurar_impresora.


* Abrimos el formulario
CALL FUNCTION 'OPEN_FORM'
     EXPORTING
          form     = 'ZPRUEBA'
*         LANGUAGE = P_IDIOMA
          options  = itcpo
          device   = 'PRINTER'
```

```
*         DIALOG   = 'X'
          dialog = space           " Sin diálogo
      EXCEPTIONS
          OTHERS   = 1.

IF sy-subrc NE 0.
  WRITE /'error al abrir formulario'.
  STOP.
ENDIF.

* Carga el contenido de las variables
PERFORM llenar_variables.

* Muestra el cuerpo del formulario
PERFORM mostrar_cuerpo_form.

CALL FUNCTION 'WRITE_FORM'
     EXPORTING
          window = 'MARCO'
     EXCEPTIONS
          OTHERS = 1.

* Imprimimos la ventana dirección
calle = 'C/ Modesto Lafuente 23'.
ciudad  = '28003, Madrid'.
pais = 'SPAIN'.
CALL FUNCTION 'WRITE_FORM'
     EXPORTING
          window = 'ADDRESS'
     EXCEPTIONS
          OTHERS = 1.

IF sy-subrc NE 0.
  WRITE /'error al escribir en ventana DIRECCION'.
  STOP.
ENDIF.



CALL FUNCTION 'CLOSE_FORM'.
*&---------------------------------------------------------------------*
*&      Form  CONFIGURAR_IMPRESORA
*&---------------------------------------------------------------------*
*       Configura los atributos de la impresora
*----------------------------------------------------------------------*
*  -->  p1
*  <--  p2
*----------------------------------------------------------------------*
FORM configurar_impresora.
  itcpo-tdpageslct  = space.          "Todas las páginas
  itcpo-tdnewid     = 'X'.            "Crea nuevo spool
  itcpo-tdcopies    = 1.                        "1 copia
  itcpo-tddest      = 'LP01'.         "Nombre de la impresora
```

```
  itcpo-tdpreview = 'X'.                " Visualización previa
  itcpo-tdcover     = space.              "No portada
  itcpo-tdimmed     = 'X'.               "Imprime inmediatamente
  itcpo-tddelete    = 'X'.              "Borra después de imprimir
  itcpo-tdcovtitle  = 'Ejemplo Formularios'.
  itcpo-tdtitle     = 'Ejemplo Formularios'.


ENDFORM.                    " CONFIGURAR_IMPRESORA
*&---------------------------------------------------------------------*
*&      Form  LLENAR_VARIABLES
*&---------------------------------------------------------------------*
*       Da valores a las variables que se mostrarán en el formulario
*----------------------------------------------------------------------*
*  -->  p1
*  <--  p2
*----------------------------------------------------------------------*
FORM llenar_variables.

* Llena la tabla interna de posiciones
  i_posicion-posicion = '1'.
  i_posicion-concepto = 'Papel fotocopiadora (2500 hojas)'.
  i_posicion-cantidad = 5.
  i_posicion-precio = ' 8.50'.
  APPEND i_posicion.

  i_posicion-posicion = '2'.
  i_posicion-concepto = 'Boligrafo tinta azul'.
  i_posicion-cantidad = 100.
  i_posicion-precio = '50.00'.
  APPEND i_posicion.

  i_posicion-posicion = '3'.
  i_posicion-concepto = 'Portaminas 0.5'.
  i_posicion-cantidad = 50.
  i_posicion-precio = '99.35'.
  APPEND i_posicion.

  i_posicion-posicion = '4'.
  i_posicion-concepto = 'Caja grapas (500)'.
  i_posicion-cantidad = 25.
  i_posicion-precio = '12.50'.
  APPEND i_posicion.

  i_posicion-posicion = '5'.
  i_posicion-concepto = 'Hojas transparencias'.
  i_posicion-cantidad = 15.
  i_posicion-precio = '90.50'.
  APPEND i_posicion.

*  TOTALES
  total = '302.59 Euros'.
  iva = ' 41.74'.
```

```
  subtotal = '250.85 Euros'.

ENDFORM.                    " LLENAR_POSICIONES
*&---------------------------------------------------------------------*
*&      Form  mostrar_cuerpo_form
*&---------------------------------------------------------------------*
*       text
*----------------------------------------------------------------------*
*  -->  p1        text
*  <--  p2        text
*----------------------------------------------------------------------*
FORM mostrar_cuerpo_form .

* Imprimos la ventana principal
  CALL FUNCTION 'WRITE_FORM'
       EXPORTING
          window  = 'MAIN'
          element = 'ITEM_HEADER'
       EXCEPTIONS
          OTHERS  = 1.

* Muestra las posiciones
  LOOP AT i_posicion.
    CALL FUNCTION 'WRITE_FORM'
         EXPORTING
            window  = 'MAIN'
            element = 'DATOS'
         EXCEPTIONS
            OTHERS  = 1.

  ENDLOOP.

* Muestra los totales
  CALL FUNCTION 'WRITE_FORM'
       EXPORTING
          window  = 'MAIN'
          element = 'TOTAL'
       EXCEPTIONS
          OTHERS  = 1.

* Imprime la zona de publicidad
  CALL FUNCTION 'WRITE_FORM'
       EXPORTING
          window  = 'MAIN'
          element = 'PUBLICIDAD'
       EXCEPTIONS
          OTHERS  = 1.

ENDFORM.                    " mostrar_cuerpo_form
```

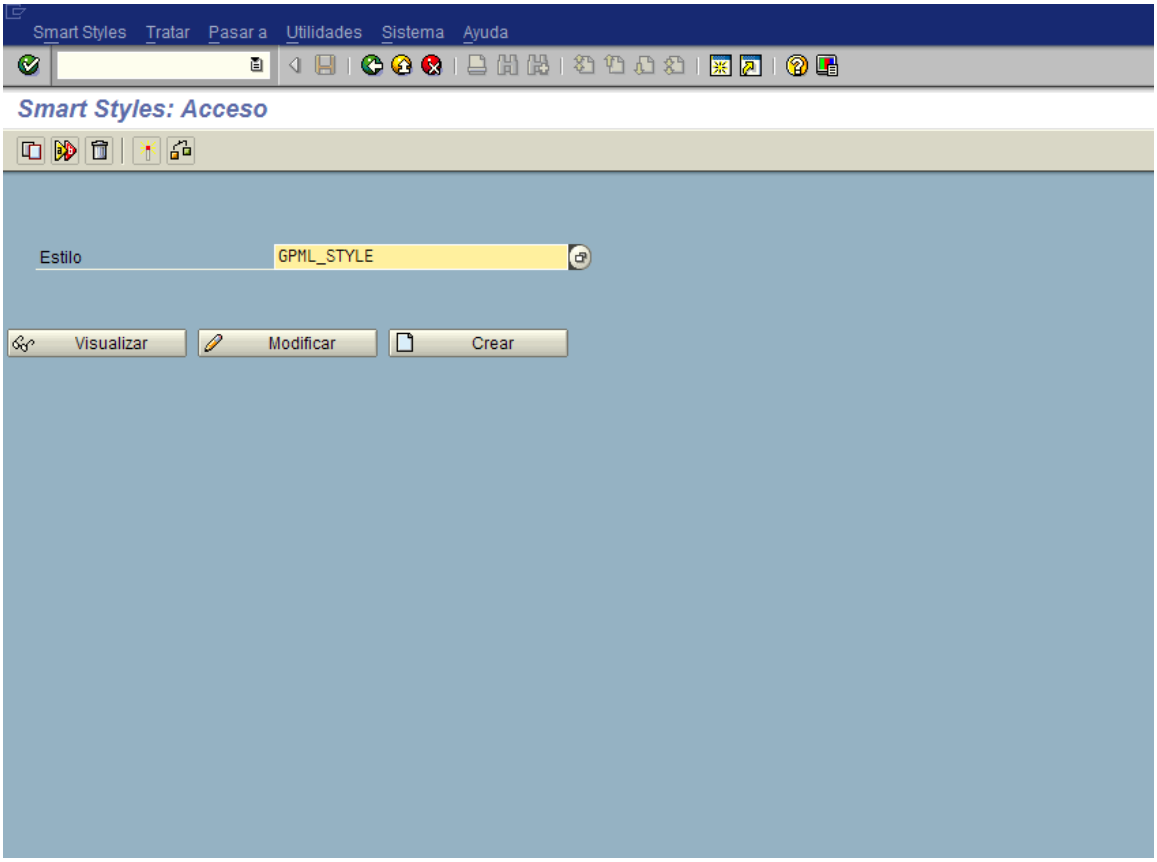
9.7. Smartforms

Se trata de una herramienta utilizada para la impresión y envío de informes e información tabulada y formateada a través de fax o correo electrónico.

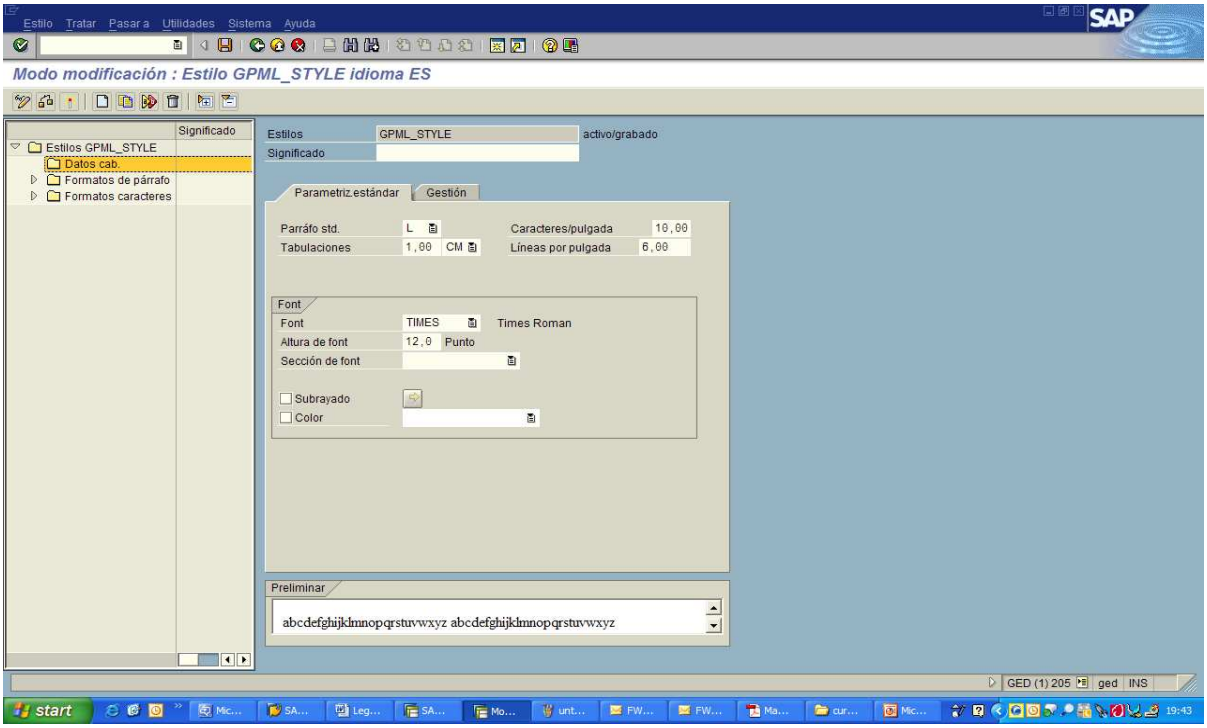
Para el tratamiento de los formularios creados con SMARTFORMS se utilizan dos transacciones:

- Una transacción para definir el estilo del formulario llamada SMARTSTYLES en la cual se definen los tipos de párrafos, tipos de caracteres, las fuentes que se van a usar, el tamaño que tendrán, las tabulaciones, ...

  La pantalla que aparece al invocar a la transacción SMARTSTYLES es la siguiente:

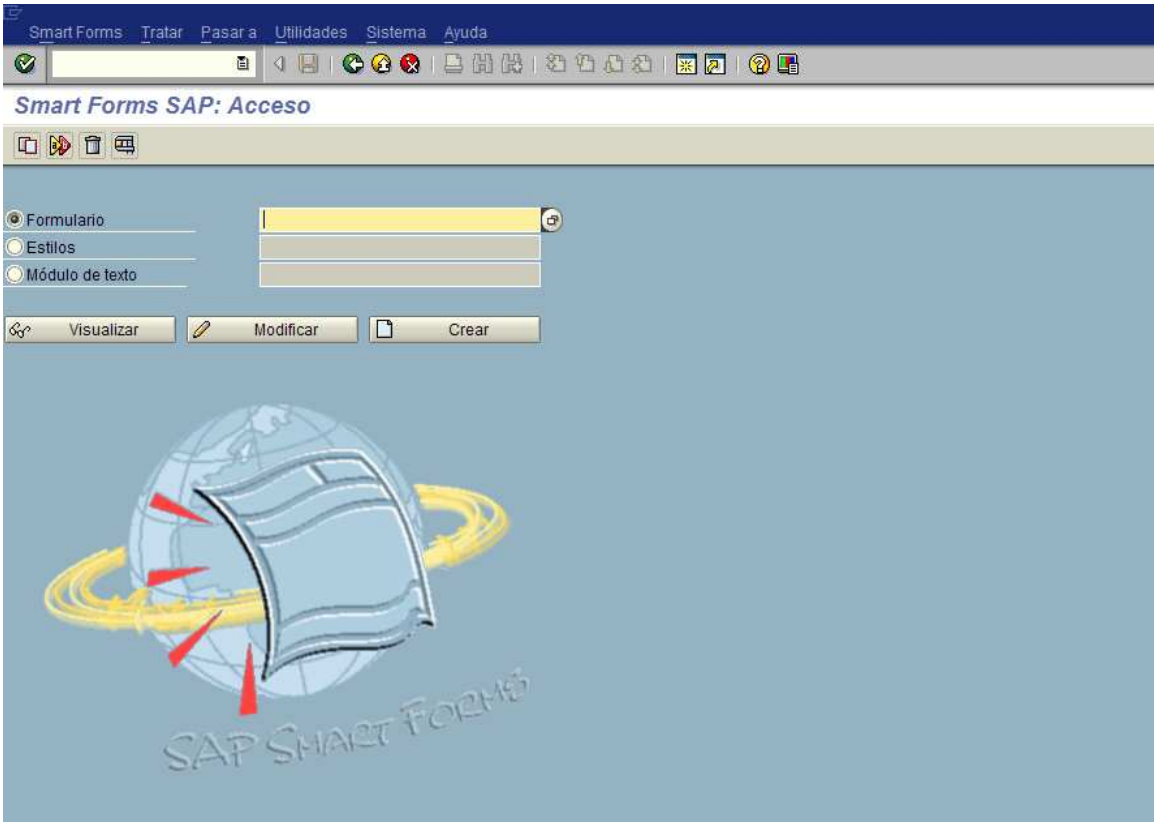


A continuación se muestra la pantalla de modificación de un estilo elegido en la transacción SMARTSTYLES.

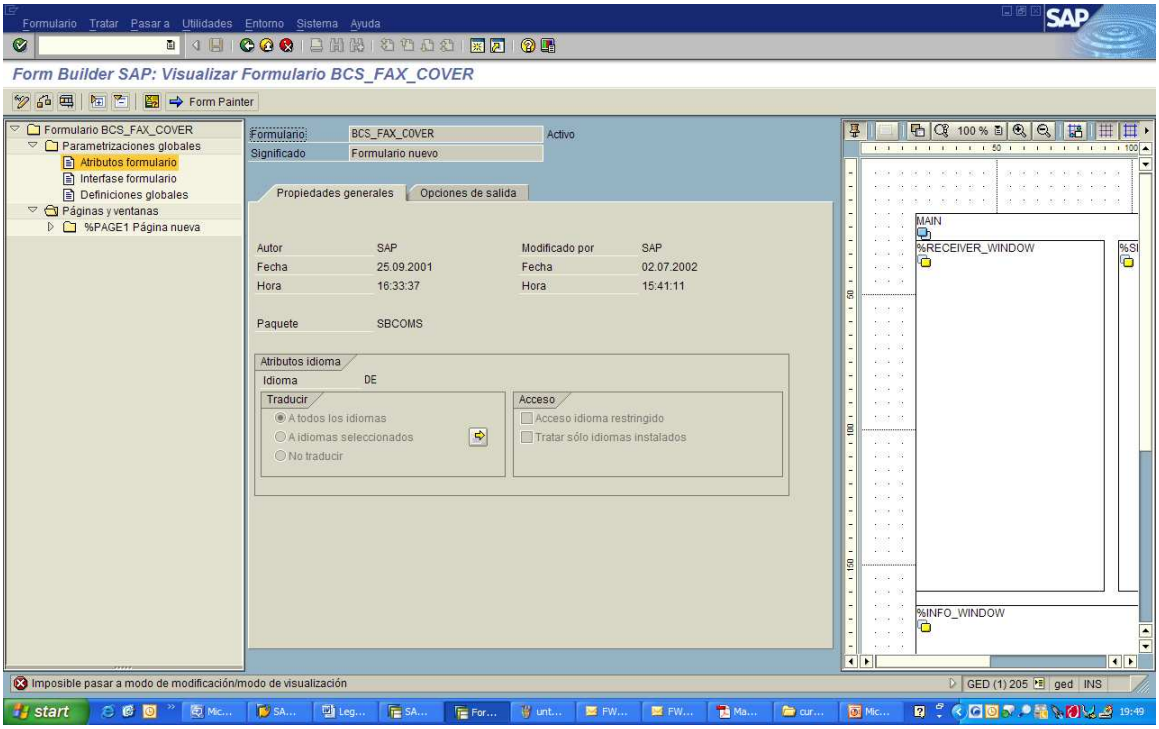


- La transacción para crear el formulario es la SMARTFORMS en la cual se definen las ventanas y su contenido. El estilo utilizado por el formulario es el creado con la transacción SMARTSTYLES.

  La pantalla que aparece al invocar a la transacción se muestra a continuación. En ella se introduce el formulario que se desea editar, crear o visualizar.



A continuación se muestra una pantalla con la visualización de un formulario tomado como ejemplo:



EJERCICIOS

Ejercicio 1

Construya una máquina para dividir valores enteros.

Debe poder introducir los dos valores a dividir en pantalla de selección, como parámetros.

Escriba el resultado con dos posiciones decimales.

Ejercicio 2

Determine los cuadrados de números enteros.

Los valores Desde-Hasta deberán indicarse como parámetros en la imagen de selección.

Ejercicio 3

Cree un mecanismo de cálculo para las cuatro operaciones fundamentales de la aritmética con dos valores enteros.

Ambos valores, así como el operador, deberán entrarse como parámetros en la imagen de selección.

Edite el resultado con dos decimales.

Ejercicio 4

Escriba un programa que genere el listado muestra adjunta.

Los operadores válidos son: " + - * / ".

Edite los números con dos decimales.

Muestra del listado:

OP	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00
1,00	1,00 OP 1,00							
2,00	1,00 OP 2,00							
3,00	1,00 OP 3,00							
4,00	1,00 OP 4,00							
5,00	1,00 OP 5,00							
6,00	1,00 OP 6,00							
7,00	1,00 OP 7,00							
8,00	1,00 OP 8,00							

Ejercicio 5

Cree un programa que genere un listado como el adjunto.

Para ello utilice los campos del sistema: SY-DATUM, SY-UZEIT y SY-UNAME

Mantenga la cabecera del programa.

Muestra del listado: _____

Este listado ha sido generado

el 27.10.1997
a las 12:23:34
por JMS

Ejercicio 6

Cree con ayuda de la tabla TABNA una lista de direcciones.

Ofrezca la posibilidad de limitar el listado a un sólo país, que se indicará como parámetro en la imagen de selección del report.

Ejercicio 7

Introducir por pantalla dos fechas y un número entero.

El programa deberá comprobar si entre esas dos fechas han pasado el número de meses que hemos introducido.

NOTA: No utilizar variables de tipo entero excepto el parámetro que indica el número de meses.

Ejercicio 8

Crear un listado de programas de nuevo desarrollo (los que empiecen por “Z”) donde aparezcan aquellos datos que se consideren útiles.

Tabla utilizada: TRDIR.

Datos que consideraremos útiles:

Nombre del programa.

Clase del programa.

Tipo de programa.

Grupo de autorización.

Autor.

Fecha de creación.

Ultimo cambio por.

Fecha cambio.

Lengua.

Status.

Saldrán ordenados por orden alfabético, por tipo y clase de programa.

Ejercicio 9

Cree un listado del volumen de negocio por país y el total por país.

Cada país deberá empezar en una nueva página.

Deberán listarse todas las entradas.

En la primera página deberá aparecer el total general del volumen de negocio de todos los países.

Nota para la programación

Lea para ello la tabla TABNA y guarde en una tabla interna los campos COUNTRY, ID, NAME1 y los volúmenes de negocio.

Determine el total del volumen de negocio por país al procesar la tabla interna.

Ejercicio 10

Cree un listado de los volúmenes de negocio por país con indicación de porcentajes.

Cada país deberá empezar en una nueva página.

Clasifique dentro de un país por orden descendente de volumen de negocio.

Ejercicio 11

Pase la tabla TABNA a una tabla interna y edite los campos TABNA-COUNTRY, TABNA-ID, TABNA-NAME1 y TABNA-SALES.

Borre todas las entradas de la tabla interna con un volumen de negocio inferior a 50.000.

Lea la entrada de la tabla interna con las claves: PAIS = ‘GB’ e ID = ‘00000003’, multiplique por 3 y modifique la entrada de la tabla.

Edite la tabla interna.

Ejercicio 12

Apartado a.

Dado un país, dato que entra el usuario por pantalla, listar sus clientes. Saldrán ordenado por ciudad, y dentro de esta, ordenados alfabéticamente por nombre.

En la cabecera del listado se especificará el código del país y su descripción.

Al final de cada ciudad, aparecerá el total de clientes de dicha ciudad.

Apartado b.

Repetir el apartado anterior pero introduciendo un rango de países.

Al final de cada país poner el total de clientes de dicho país.

El listado tendrá el siguiente formato:

cada país aparecerá en una página distinta

indicándose el código del país y su descripción

Apartado c.

Para cada cliente mostrar las ventas anuales (UMSA1)

Al final de cada ciudad, mostrar la suma de ventas anuales.

Al final de cada país mostrar la suma de ventas anuales.

Tabla de descripción de países: T005T

Ejercicio 13

Copie el report del Ejercicio 10 y modifíquelo para que el cálculo del porcentaje se realice en un módulo de función.

Ejercicio 14

Copie el report del Ejercicio 9 y modifíquelo:

La tabla interna sólo debe contener los campos COUNTRY, ID y NAME1.

Omita el cálculo y visualización de sumas.

Liste el contenido de la tabla interna desde un subprograma.

Ejercicio 15

Copie el report del Ejercicio 10 y modifíquelo para que el cálculo del porcentaje se realice en una subrutina.

Ejercicio 16

Dado un rango de materiales, escogido por el usuario, listar dichos materiales y su descripción.

La tabla usada es: MAKT.

La pantalla de selección deberá estar incluida en un frame y contener el siguiente comentario: “Introduzca un rango de materiales”, a continuación dejará dos líneas en blanco y luego pedirá el rango de materiales.

Guardar los materiales seleccionados en una tabla interna y listarlos por número de material.

Eliminar de la tabla aquellos registros cuya primera palabra de descripción empiece por “slug”, y volver a listarlos.

Ejercicio 17

Realizar un listado de datos bancarios de proveedores utilizando la base de datos lógica KDF.

Las tablas que utilizaremos serán la LFA1 y la LFBK.

Poner el usuario y la fecha en la parte superior derecha de la página.

Ejercicio 18

Cree un listado con totales por sociedad de los proveedores (vea el listado de muestra).

Para cada proveedor, posiciónese en una nueva página.

Liste el campo LFC3-SOLLL (suma de contabilización debe) por proveedor, para cada sociedad (LFB1-BUKRS) y para cada ejercicio(LFC3-GJAHR). Liste la suma de contabilizaciones debe por sociedad (vea el listado de muestra 2).

A continuación, liste en una nueva página los sociedades para cada proveedor (vea el listado muestra 3).

El listado debe tener una hoja de cubierta, así como una hoja adicional al final (vea los listados muestra 1 y 4).

Nota para la programación

- Trabajar con la base de datos lógica KDF.
- Debe incluir los siguientes eventos en su programa:

```
START-OF-SELECTION
TOP-OF-PAGE
GET LFA1
GET LFB1
GET LFC3
GET LFB1 LATE
GET LFA1 LATE
END-OF-SELECTION
```

Listado muestra 1

Volúmenes anuales de los proveedores 1

Sumas por sociedad de los proveedores
Resumen de sociedades por proveedor
y hoja de resumen

Listado muestra 2

Volúmenes anuales de los proveedores 2

Zabel N. cuenta.: 00000001
Lerehoster, 26
7197 Melibrenn

Sociedad: 0001

1991	24.314.40-
1992	1.000.90-
Suma sociedad 0001	25.315.30-

Listado muestra 3

Volúmenes anuales de los proveedores 3

Para el proveedor Zabel N. cuenta.: 00000001
se han procesado las
siguientes sociedades:
0001
0002

Listado muestra 4

Volúmenes anuales de los proveedores 4

Se han procesado 5 proveedores

Ejercicio 19

Escriba un report que cree un listado con direcciones de proveedores y datos de documentos.

Haga una clasificación fija por país, cuenta y sociedad, así como por uno de los campos siguientes, como último campo de clasificación:

BSIK-BUDAT

BSIK-WAERS

BSIK-DMBTR

El último campo de clasificación debe introducirse como parámetro del report.

Ejercicio 20

Crear facturas por cliente con los datos siguientes:

Número de Factura: 000020200			Fecha: 10-06-1998	
Cliente: Juan Pérez Gomez		Ciudad: D.F.	Teléfono: 5-546-68-90	
Cantidad	Cve. del producto	Descripción	Precio	Importe
5	2354-90	Cuaderno rayado de 100 hojas	12.00	60.00
Fecha a Pagar: 30-06-1998		Fecha de Pago: 15-06-1998	Importe Total:	60.00

Ejercicio 21

Realiza un download de una serie de programas teniendo como parámetros de entrada la ruta del directorio donde se guardan los programas y ficheros de los que se debe hacer el download.

Ejercicio 22

Realiza un upload de una serie de programas teniendo como parámetros de entrada la ruta del directorio en donde se encuentran los ficheros y el nombre de los mismos.

Ejercicio 23

Creación de entradas nuevas en la tabla *zclient* y utilizando un archivo plano para obtener los datos.

Ejercicio 24

Realizar un upload de impresión del código de un programa teniendo como parámetros de entrada el nombre del programa que se va a imprimir.

Ejercicio 25

Realizar un Batch Input para la transacción FD01.

Características: Esta transacción es usada para crear clientes en SAP. Se debe tomar los datos de un fichero y cear todos los clientes. Debe aparecer al final un reporte con el número de clientes asignados por SAP de los clientes creados exitosamente y o mensaje con los posibles errores.

Hacerlo con CALL TRANSACTION y con CALL FUNCTION.

Transacción: FD01. Programa SAPMF02D

- Pantalla 105

Campos: RF02D-BUKRS Compañía

RF02D-KTOKD Sector.

- Pantalla 110

Campos: KNA1-NAME1 Nombre

KNA1-SORTL Término de búsqueda

KNA1-ORT01 Ciudad.

KNA1-PSTLZ Código postal

KNA1-LAND1 Idioma

KNA1-SPRAS Idioma.

KNA1-TELF1 Teléfono

KNA1-TELFX Fax

Comando: Enter

- Pantalla 120

  Campos:

  Comando: Enter

- Pantalla 130

  Campos:

  Comando: Enter

- Pantalla 210

  Campos: KNB1-AKONT Cuenta

  Comando: Enter

- Pantalla 215

  Campos:

  Comando: Enter

- Pantalla: 220

  Campos:

  Comando: Enter.

- Pantalla: 230

  Campos:

  Comando: Grabar F11.

El fichero a utilizar es el siguiente:

CAMPO	TAMAÑO
Compañía	4
Sector	4
Nombre	20

Término de búsquda	10
Ciudad	20
Código postal	5
País	3
Idioma	1
Teléfono	8
Fax	8
Cuenta	10

Ejercicio 26

⇒ Realizar un Batch Input que a partir de un fichero secuencial que contiene datos de clientes:

- Cree nuevos registros de clientes en el maestro de clientes, mediante la transacción FD01.
- Modifique registros de clientes ya existentes, mediante la transacción FD02.

La estructura del fichero deberá tener la siguiente estructura:

Campo	Longitud	
Customer	10	
Company Code	4	(1000)
Account Group	4	(DEBI)
Name	8	
Search term	3	
City	7	
Postal Code	5	
Country	2	(ES)
Language	1	(S)
Reconciliation account	6	(120000)

El programa debe acceder al maestro de clientes (KNA1) para comprobar si existe el cliente en proceso, y así simular vía Batch Input la creación (FD01) o modificación (FD02) de los clientes del fichero.

⇒ Comprobar en la tabla KNA1 que las operaciones se han realizado correctamente.

Ejercicio 27

Realizar un Batch Input que a partir de un fichero secuencial que contiene datos de proveedores, cree nuevos registros en el maestro de proveedores utilizando la transacción FK01.

El nombre del fichero de proveedores, así como el path hasta llegar a él, serán introducidos como parámetros.

La estructura del fichero será la siguiente:

Company Code(4) : 1000

Account Group(4) : KRED

Name(8)

Search Term(3)

Street(10)

City(7)

Postal Code(5)

Country(2) : ES

Languaje(1) : S

Reconciliation Account(5): 31000

Planning Group(2): A1

Realizar el ejercicio con la técnica del CALL TRANSACTION (en invisible), y en caso de error escribir los proveedores en el fichero de texto:

<nombre_fichero_proveedores>.err'.

Ejercicio 28

Realizar un Batch Input utilizando la técnica del CALL TRANSACTION que simule al editor de programas (transacción SE38).

El programa debe pedirnos un rango de programas de los cuales se hará un Display y se imprimirá su código.

Los programas se encuentran en la tabla TRDIR.

Se ejecutará en modo invisible y al final se dará un mensaje si todo ha salido correctamente, en caso contrario saldrá el mensaje correspondiente al error.

Ejercicio 29

⇒ Realizar un Batch Input para simular la creación de nuevos clientes en el registro maestro.

Los datos están en el fichero "FICH1.TXT".

Los campos existentes para cada registro son los siguientes:

Company Code

Account Group

Name

Search term

City

Postal Code

Country

Language: S

Reconciliation account

⇒ Definir en un pool de mensajes los mensajes que indiquen si el programa se ejecuta correctamente o, en caso contrario, nos informen de los errores que se producen.

⇒ Una vez creado el programa:

- ejecutarlo

- procesar la sesión que se obtiene

- comprobar en la tabla KNA1 que se han creado los clientes del fichero.

MENSAJES:

I011: Error apertura group.

I015: Apertura group OK.

I012: Error apertura Fich1.

I016: Apertura Fich1 OK.

I017: Ha finalizado la lectura de Fich1.

I018 Leido un nuevo registro &

I013 Error al cerrar Fich1.

I014 Error al cerrar sesión.

I019 Cierre de sesión OK.

Ejercicio 30

Genere un listado con datos de proveedores.

Permita que el usuario posicione el cursor sobre una línea, y tras pulsar la tecla PF13, obtenga información bancaria para el proveedor seleccionado, en un listado de ramificación.

Asegúrese de que una selección no válida de línea no tenga ningún efecto.

Los nombres de los campos que contienen información bancaria son:

- Código del banco: LFBK-BANKL
- Número de cuenta: LFBK-BANKN
- Código del país del banco: LFBK-BANKS

Ejercicio 31 Listado interactivo de movimientos de proveedores.

Generar un listado interactivo de proveedores, que contenga la siguiente información :

Un listado básico de proveedores (tabla LFA1) , donde se indique el código de proveedor y el nombre de proveedor. A partir de este listado básico, podremos realizar una serie de funciones :

- Obtener otros datos generales del proveedor. (con **F13**).

  En una ventana sobre el listado de proveedores, visualizaremos :

  Sociedad del proveedor. (en la tabla LFB1).

  Calle. (en la tabla LFA1).

  Ciudad. (en la tabla LFA1).

  Región. (en la tabla LFA1).

País. (en la tabla LFA1).

Código Postal. (en la tabla LFA1).

Teléfono. (en la tabla LFA1).

- Obtener los datos bancarios del proveedor. (con **F14**).

  En una ventana sobre el listado de proveedores, visualizaremos :

  Código de banco. (en la tabla LFBK).

  Código de cuenta. (en la tabla LFBK).

  Código del país del banco. (en la tabla LFBK).

- Obtener los documentos de cada proveedor (con **Seleccionar** o **Doble-Click**).

  En un listado de ramificación, visualizaremos :

  Número de documento. (en la tabla BSIK).

  Fecha de contabilización. (en la tabla BSIK).

  División. (en la tabla BSIK).

  Importe. (en la tabla BSIK).

Ejercicio 32 Listado interactivo de movimientos de proveedores (continuación)

Vamos a dar nuevas funcionalidades al listado de proveedores.

Permitiremos que una vez visualizados los documentos de un proveedor (con doble-click), estos se puedan clasificar y visualizar nuevamente, sin salir del listado de ramificación en el que estamos.

Con **F16** clasificará los documentos por fecha de contabilización.

Con **F17** clasificará los documentos por importe.

Si pulsamos un doble-click sobre un documento, entonces visualizaremos el detalle de cada documento. Los campos a listar serán :

Número de documento. (en la tabla BSIK).

Fecha de contabilización. (en la tabla BSIK).

Sociedad (compañía). (en la tabla BSIK).

Año fiscal. (en la tabla BSIK).

Posición dentro del doc contable. (en la tabla BSIK).

Tipo de documento. (en la tabla BSIK).

Moneda. (en la tabla BSIK).

Importe. (en la tabla BSIK).

Ejercicio 33. Realizar una transacción. (Mantenimiento de Empleados), que Crea, Modifica y Visualiza los datos generales de una tabla de empleados, que es creada en el diccionario de datos.

PASOS A REALIZAR :

A) Crear una Tabla de empleados.

B) Crear un Programa Module Pool.

C) Crear una transacción.

D) Crear un Status de superficie. (Menú).

E) Crear Pantallas de la transacción.

F) Codificar un mantenimiento de empleados.

A) **Crear una Tabla de empleados ZEMP** con el siguiente formato.

ZEMP-MANDT	**x**	CLNT C	3	Mandante
ZEMP- ZEMPLE	**x**	CHAR C	4	Código de empleado
ZEMP-NAME1		CHAR C	30	Nombre
ZEMP- STRAS		CHAR C	30	Calle y número
ZEMP- ORT01		CHAR C	25	Ciudad
ZEMP- REGIO		CHAR C	3	Región
ZEMP- LAND1		CHAR C	3	Clave país
ZEMP- ZSEXO		CHAR C	1	Sexo
ZEMP- ZSAPU		CHAR C	1	Es usuario SAP ?
ZEMP- TELF1		CHAR C	16	Número de teléfono

Donde los elementos de datos se llaman igual que los campos.

Y donde aquellos campos que empiecen con 'Z', tendrán que definir un elemento de datos nuevo, ya que no existen el diccionario de datos.

B) Crear un Programa Módule Pool. Por ejemplo ZMANTEMP.

C) Crear una transacción. Por ejemplo ZMEO. Dándole el nombre del module Pool recién creado ZMANTEMP y el número de la primera pantalla de la transacción (100).

D) Crear un Status de superficie. (Menú).

Desde el menú Empleados : DISP Visualizar

MODI Modificar

NEW Crear

DEL Borrar

FIN Salir

En la Barra de botones de aplicación tendremos los siguientes botones cada uno de ellos con su correspondiente icono :

Visualizar

Modificar

Crear

Borrar

En teclas de Función dispondremos de :

F5 = Visualizar

F6 = Modificar

F7 = Crear

F8 = Borrar

Definiremos dos Títulos :

Z01 = MANTENIMIENTO DE EMPLEADOS (Pantalla inicial)

Z02 = MANTENIMIENTO DE EMPLEADOS (Entrada de datos)

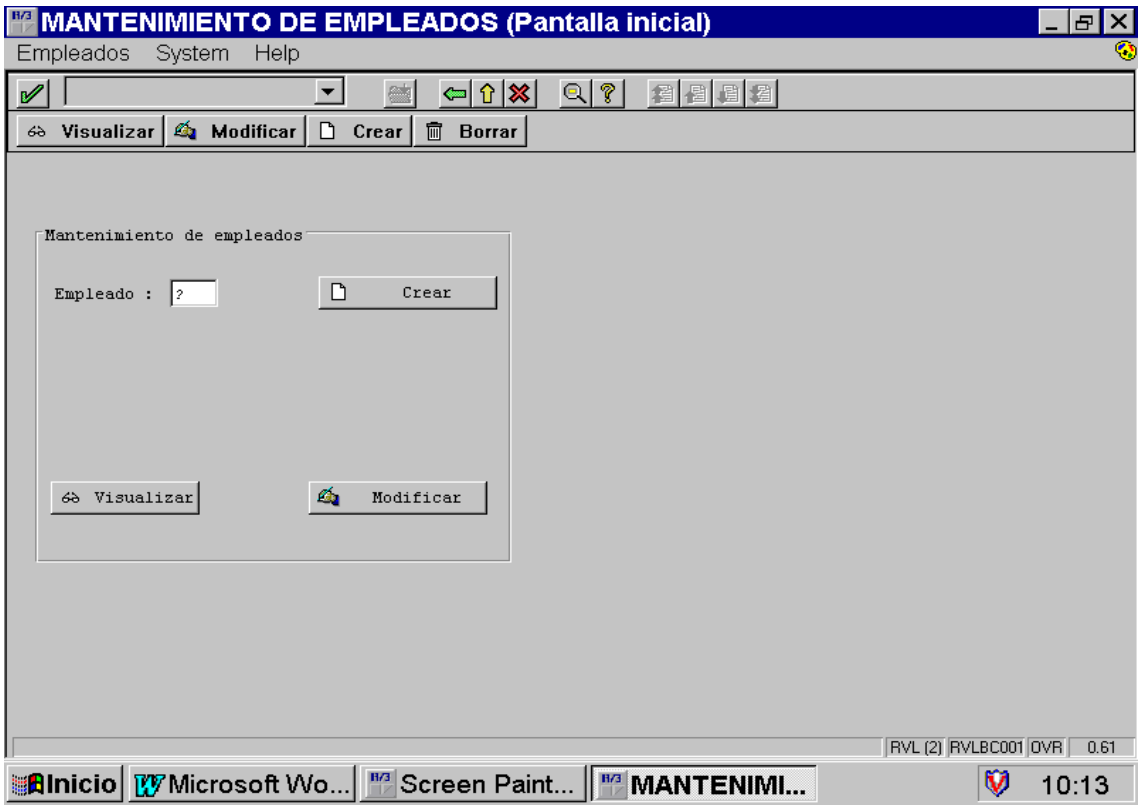
E)Crear las Pantallas de la transacción.

Crearemos dos pantallas :

0100 PANTALLA INICIAL TRANSACCION ZME0

0200 PANTALLA DE DATOS DE LA TRANSACCION ZME0

Puedes ver las Pantallas 0100 y 0200 en la siguiente pagina



Región y País tienen un chequeo automático en tablas del diccionario de datos. En caso de que se introduzcan valores incorrectos se permitirá modificar los dos campos simultáneamente. Además dispondremos de una ayuda de valores posibles sobre ambos campos.

Al lado del campo Teléfono, se dibujará un icono representativo.

El campo Sexo estará formado de un grupo de dos radio-buttons excluyentes entre si.

F) Instrucciones para la codificación del mantenimiento de empleados.

1) Control del Menú.

En la pantalla 0100, desactivamos la función de Grabar.

En la pantalla 0200, desactivamos las funciones : Crear, Borrar, Visualizar y Modificar. En caso de que estemos en Consultando un empleado, también desactivaremos la función Grabar.

Tendremos que permitir salir de una pantalla (con BACK , CANCEL o EXIT) aunque queden campos obligatorios por completar.

2) Visualizar empleado. En caso de que no exista el empleado dar un mensaje.

3) Crear un empleado. Introducir datos de empleado y grabar.

4) Modificar los datos de un empleado.

Tendremos que bloquear el empleado para que ningún otro usuario pueda realizar modificaciones simultáneamente, y mantenerlo bloqueado hasta que salga de la pantalla.

Será necesario crear un objeto de bloqueo en el diccionario de datos.

5) Borrar un empleado de la tabla de empleados.

Previamente al borrado de un empleado pediremos confirmación con la función: 'POPUP_TO_CONFIRM_STEP'.

Nota para la creación de tablas necesarias en algunos ejercicios:

Tablas para la base de datos de ejecicios.

Clientes-proveedores : ID, Nombre, apellidos, ciudad, opcion, telefono, Sexo, Fecha nacimiento.

Opcion: C: Clientes, D: distribuidores, P: pesonal.

Productos: ID Producto, descripcion, cantidad, precio unitario.

Facturas: ID Factura, ID cliente, fecha factura, fecha a pagar, fecha de pago.

Posiciones: ID factura , posicion, id producto, cantidad, importe.

ANEXO 1:NOTAS PARA LA PROGRAMACIÓN ABAP/4

Estas son algunas notas que se deberían seguir en los desarrollos ABAP. El seguimiento o no de ellas para nada influirá en el perfecto funcionamiento de los programas, pero si servirá bastante para standarizar de cierta forma la manera de programar. Esto minimizará el tiempo de comprensión y de modificación de programas que no hemos hecho nosotros mismos, y hará más fácil estas modificaciones.

1) Comentarios del principio. Deben ir en el siguiente formato:

```
************************************************************************
* Nombre de la empresa
*
* Programa: NOMBRE MAYUSCULAS   Autor: Nombre Apellido   Fecha: 04/09/1998
*
* DESCRIPCION: Título, el mismo que lleve el programa en atributos, y breve descripción
*de lo que hace y como lo hace el programa.
*
* Modificación: Fecha de modificación       Autor:
* Pedida por: Nombre del que autorizó la modificación
* DESCRIPCION: Breve descripción de la modificación realizada
*
************************************************************************
```

2) Los nombres de los programas deberían ser de la siguiente forma: **ZAABXXX**, donde:

   - **Z** es obligatorio para la creación de nuevos objetos.
   - **AA** son las iniciales del módulo para el que se crea el programa.
   - **B** depende del tipo de programa de la siguiente forma:
     - **R** si es un reporte.
     - **B** si es un batch-input o batch-output.
     - **T** para los programas module pool de las nuevas transacciones.
     - **I** para los programas de impresión.
   - **XXX** número consecutivo empezando en 001.

   Ejemplo: **ZMMR001**: Seria el primer report del módulo de MM.

   Los programas que sean includes llevan una nomenclatura distinta: **ZINCXXX** donde todo es fijo menos **XXX** que es un numero consecutivo empezando en 001.

3) Los nombres de las tablas deben ser: **ZTAAXXX**, donde:

   - **ZT** son obligatorios.
   - **AA** son las iniciales del módulo para el que se crea la tabla.
   - **XXX** número consecutivo empezando en 001.

4) Los nombres de los formularios deben ser los más explicativos posibles ya que nos permite un nombre largo.

5) Los select deben de estar de la siguiente manera:

```
Select * from TABLA
        Where       campo1 = variable1   and/or
                    campo2 = variable2   and/or
                    campon = variablen.
Endselect. " TABLA
```

6) No poner mas de 35 líneas dentro de un select- endselect. Usar procedimientos. Igual para loop-endloop y todo tipo de bucles.

7) No poner mas de 2 ó 3 líneas en caso de tener que usar select anidados. Usar procedimientos. Igual para loop-endloop y todo tipo de bucles.

8) Después de un select single usar siempre la comprobación de la variable del sistema SY-SUBRC.

9) Explicar para que se usan las variables declaradas, en la misma línea de la declaración.

10) Poner una solo instrucción en cada línea.

11) Intentar usar lo máximo posible los nombres de variables de SAP.

12) Al declarar una tabla interna o estructura explicar antes de la declaración para que se usa, así como el significado de cada campo declarados la tabla interna es usada para un report de salida se debería poner la longitud de salida del campo declarado, para así facilitar las posiciones de salida. Si mismo poner al final del nombre de la tabla "**_i**" para identificar rápidamente que estamos haciendo referencia a una tabla interna. (Importante no aplica igual para registros).

13) En la llamada de un procedimiento debería haber un pequeño comentario explicando que hace y/o por que se llama dicho procedimiento.

14) En un programa batch-input siempre es necesario el llenado de pantallas. Cada llenado de una pantalla distinta debe hacerse en un procedimiento, por breve que este sea. El nombre de estos módulos debe ser **llena_pantalla_xxxx**, donde **xxxx** debe ser el número de pantalla (Dynpro) que estemos rellenando. Es muy importante recordar que no es lo mismo el dynpro 100 que el dynpro 0100.

15) Al crear un procedimiento usar los comentarios y separadores standard de SAP. Estos son:

```
*&---------------------------------------------------------------------*
*&      Form  NOMBRE DEL FORM EN MAYUSCULAS
*&---------------------------------------------------------------------*
*  Entrada:
*          Parámetros de entrada. " Descripción
*
* Salida:
*          Parámetros de salida. " Descripción
*
* DESCRIPCION: Breve descripción del procedimiento
*---------------------------------------------------------------------*
form nombre del form.

                Código del form.

Endform. " NOMBRE DEL FORM EN MAYÚSCULAS.
```

16) Los nombres de los procedimientos deben ser explicativos y relacionados con la operación.

17) El orden del código de los programas debería ser.

Comentarios iniciales. Ver punto 1).

1. Definición del programa, clase de mensajes, etc.
2. Definición de tablas del diccionario de datos.
3. Definición de tablas internas y estructuras.

4. Definición de variables.
5. Definición de parámetros de entrada.
6. Eventos y código.
7. Definición de procedimientos y módulos.

Nota: Todos estos apartados deberían ir señalizados al principio con el nombre del mismo.

Comentar el código lo máximo posible con comentarios concretos y aclaratorios, sin dejar que éstos en vez de ser aclaratorios empeoren el entendimiento del programa. No hay que olvidar que un mal comentario es mucho pero que la ausencia del mismo.

18) Usar siempre las plantillas de programación de report (ZRERIVMEX) y de batch-input (ZBIRIVMEX), así como intentar reutilizar código usado en otros programas/proyectos. Para ello intentar hacer procedimientos lo mas estándards y reutilizables posibles. Documentándolos muy bien y comunicándolo al resto del grupo ABAP. Muy importante en este punto es intentar crear, mantener, ampliar u usar una biblioteca de procedimientos creada por nosotros.

Algunas recomendaciones más.

Poner mucha atención en donde y cuando poner los clear de variables tablas.

Usar clear tabla siempre después de un append tabla.

Dejar espacios en blanco entre grupos de intrucciones con algún significado concreto

Tabular el programa ayudándose de la opción Pritty printer del editor.

Recomendaciones antes de empezar a codificar un programa.

Estas son algunas notas que se deberían seguir, para empezar a codificar todos los programas. Igual que las notas anteriores no son totalmente imprescindibles, el seguimiento de estas, pero si facilitarán en una buena codificación final.

1. Entender perfectamente que se quiere hacer. Para ello leer atentamente las especificaciones funcionales, preguntando a los responsables de estas, cuando sea necesario para su total entendimiento.
2. Definir las entradas de parámetros cuando sea necesario. Diseñar la posible pantalla o pantallas, con la obligatoriedad de los campos cuando sea necesario.
3. Buscar relaciones entre las tablas implicadas en el programa. Sacar llaves y obtener pequeño esquema entre las tablas relacionadas para encontrar la mejor forma de búsqueda de datos.
4. Cuando sea necesario definir bien la salida, por pantalla, fichero, etc. Definir cabeceras y todas las partes que sean necesarias.
5. Buscar datos necesarios para simular el programa en desarrollo. Cuando esto no sea posible, pedir a la persona indicada que nos lo proporcione. Esta parte es fundamental para Batch-input, para poder hacer el seguimiento on-line de la transacción que se va ha usar.
6. Codificar.
7. Probar en desarrollo.
8. Notificar al consultor funcional o encargado de informática que ya está listo el programa y esperar la aprobación del mismo.

ANEXO 2 TYPE CONVERSION TABLE

Conversion table **(f -> g)** depending on the types of **f** and **g**:

C -> C Left-justified transfer; padded with blanks if necessary.

C -> D The field f must be an 8-character date in YYYYMMDD format.

C -> F The character string in f must be a valid representation of a floating point number (see DATA).

C -> N Only numeric characters are valid here. They are moved to g, right-justified and padded with zeros on the left.

C -> T The field f must contain a 6-character time specification in MMSS format.

C -> P The field f must contain a decimal number, i.e. a sequence of numeric characters with optional signs that includes no more than one decimal point; there may be blanks on either side. If g is too short, an overflow error can occur.

C -> X The field f must contain a hexadecimal character string (i.e. the only valid characters are 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F). The number to be converted is treated as a hexadecimal number rather than a decimal number,

e.g.: C'15' -> X'15'.

It is transported left-justified to g and either padded with zeros or truncated;

e.g.: C'AB' -> X'AB00'.

if is processed up to the first blank.

Examples:

C'ABC' -> X'ABC0', C'ABC0' -> X'ABC0'

C'ABC D' -> X'ABC0', C' AB' -> X'0000'

D -> C Left-justified transfer without conversion.

D -> D Transfer without conversion.

D -> F As for D -> P and then P -> F.

D -> N As for D -> C and then C -> N.

D -> P Inverse of P -> D.

D -> T Not supported: error message.

D -> X Inverse of X -> D.

F -> C f is converted to <mantissa>e<exponent> format and moved to g.

e.g.: F'-3.142' -> C'-3.14200000000000E+00'

The value of the mantissa is 0 and, depending on the amount, greater than or equal to 1.0 and smaller than 10.0.

Both the mantissa and the exponent always appear with signs.

If g is too short, the mantissa is rounded.

e.g.: F'3.152' -> C'+3.2E+00'

The length of g should be at least 6, otherwise it (g) is filled with asterisks (*).

F -> D See F -> N.

F -> F Transfer without conversion.

F -> N f is rounded as with F -> P and then treated like a P field.

F -> P f is rounded, e.g. F'-3.512' -> P'-4'.

F -> T See F -> N.

F -> X See F -> N.

N -> C f is treated like a C field; leading zeros remain.

N -> D As for N -> C and then C -> D.

N -> F As for N -> P and then P -> F.

N -> N Right-justified transfer; on the left, padded with zeros or truncated.

N -> P f is packed and moved to g with a positive sign (+). If g is too short, an overflow error can occur.

N -> T As for N -> C and then C -> T.

N -> X As for N -> P and then P -> X.

P -> C f is moved to g with trailing sign and, if required, a decimal point.

e.g.: P'-1234567' -> C'12345.67-'

Notes:

1) One position is always reserved for the sign and, in the event of a positive number, a blank is output.

2) Leading zeros are output as blanks.

3) If g is too short, the blank representing the sign in the case of positive numbers is omitted; if this is insufficient, the number is truncated on the left - this is indicated by an asterisk (*).

Examples (the P field f has the length 2, the C field g the length 3):

P'123' -> C'123', P'-123' -> C'*3-'

4) If you do not want to reserve a position for the sign, use the WRITE TO statement with the addition NO-SIGN.

5) To convert with leading zeros and without formatting characters, use the UNPACK statement.

P -> D The value in f is the absolute date (i.e. the number of days since 30.12.1899) and is moved to g in the YYYYMMDD format.

P -> F The field f is moved to g as a floating point number.

P -> N Right-justified transfer without sign; padded with zeros on the left.

P -> P If g is too short, an overflow error can occur.

P -> T The value in f is an absolute time (i.e. the number of seconds since midnight modulo 24 hours = 86,400 seconds) and is moved to g in HHMMSS format.

P -> X The value in f is stored in g as a hexadecimal number.

e.g.: P'15' -> X'0F'.

Negative numbers are represented by the two's complement.

e.g.: P'-153' -> X'FF67'.

If g is too short, the number is truncated on the left.

T -> C As for D -> C.

T -> D Not supported: error message.

T -> F As for T -> P and then P -> F.

T -> N As for T -> C.

T -> P Inverse of P -> T.

T -> T Transfer without conversion.

T -> X Inverse of X -> T.

X -> C f is converted to hexadecimal format.

e.g.: X'0F' -> C'0F'

X -> D The value in f is an absolute date (i.e. the number of days since 30.12.1899) and is moved to g in YYYYMMDD format. (See also P -> D).

X -> F As for X -> P and then P -> F.

X -> N As for X -> P and then P -> N.

X -> P f is a hexadecimal number and is moved to g in decimal packed form.

e.g.: X'0F' -> P'15'

If the length of f is greater than 4, only the last 4 bytes are processed.

If g is too short, an overflow error can occur.

X -> T The value in f is an absolute time (i.e. the number of seconds since midnight modulo 24 hours = 86,400 seconds) and is moved to g in HHMMSS format. (See also P -> T.)

X -> X Left-justified transfer; filled with X'00', if necessary.

ANEXO 3 VARIABLES DEL SISTEMA

VARIABLE	TIPO		L	
SYST-ABCDE	CHAR	C	26	CONSTANT: Alfabeto (A,B,C,...)
SYST-APPLI	RAW	X	2	Aplicaciones SAP
SYST-BATCH	CHAR	C	1	Batch activo (X)
SYST-BATZD	CHAR	C	1	SUBMIT batch: Diario
SYST-BATZM	CHAR	C	1	SUBMIT batch: mensual
SYST-BATZO	CHAR	C	1	SUBMIT batch: único
SYST-BATZS	CHAR	C	1	SUBMIT batch: inmediatamente
SYST-BATZW	CHAR	C	1	SUBMIT batch: semanal
SYST-BINPT	CHAR	C	1	Batch input activo (X)
SYST-BREP4	CHAR	C	4	SUBMIT batch: Nombre de raíz del report de llamada
SYST-BSPLD	CHAR	C	1	SUBMIT batch: Salida de lista en SPOOL
SYST-CALLD	CHAR	C	1	CALL mode active (X)
SYST-CALLR	CHAR	C	8	IMPRIMIR: ID para funciones de diálogo
SYST-CCURS	DEC	P	5	Tipo cambio/Campo resultado CURRENCY CONVERT
SYST-CCURT	DEC	P	5	Tipo de cambio en tabla de aplicación CURRENCY CONVERSION
SYST-CDATE	DATS	D	8	Fecha de tipo de cambio de CURRENCY CONVERSION
SYST-CFWAE	CUKY	C	5	Utilización interna
SYST-CHWAE	CUKY	C	5	Utilización interna
SYST-COLNO	INT4	X	4	Columna actual en la creación de la lista
SYST-CPAGE	INT4	X	4	Número de página actual
SYST-CPROG	CHAR	C	8	RUNTIME: Programa principal
SYST-CTABL	CHAR	C	4	Tabla de tipo de cambio en CURRENCY CONVERSION
SYST-CTYPE	CHAR	C	1	Tipo de cambio 'M','B','G' de CURRENCY CONVERSION
SYST-CUCOL	INT4	X	4	Posición del cursor (columna)
SYST-CUROW	INT4	X	4	Cursor position (line)
SYST-DATAR	CHAR	C	1	Indicador: Datos recibidos
SYST-DATUM	DATS	D	8	SYSTEM: Fecha del día
SYST-DAYST	CHAR	C	1	¿ Horario de verano activo ?
SYST-DBCNT	INT4	X	4	Cantidad elementos en conjunto tratado para operaciones BD
SYST-DBNAM	CHAR	C	2	Base de datos lógica en report ABAP/4
SYST-DBSYS	CHAR	C	10	SYSTEM: Sistema de base de datos
SYST-DCSYS	CHAR	C	4	Sistema de diálogo
SYST-DEBUG	CHAR	C	1	Utilización interna
SYST-DSNAM	CHAR	C	8	RUNTIME: Nombre del set de datos para salida en SPOOL
SYST-DYNGR	CHAR	C	4	Grupo de dynpros del dynpro actual
SYST-DYNNR	CHAR	C	4	Número de la imagen en pantalla actual
SYST-ENTRY	CHAR	C	72	Utilización interna
SYST-FDAYW	INT1	X	1	Día de semana en el calendario de fábrica
SYST-FDPOS	INT4	X	4	Lugar de hallazgo de un string
SYST-FFILE	CHAR	C	8	INTERNO: Flatfile (USING/GENERATING DATASET)
SYST-FLENG	INT4	X	4	Utilización interna (longitud de campo)
SYST-FMKEY	CHAR	C	3	Menú de códigos de funciones actual
SYST-FODEC	INT4	X	4	Internal use (field decimal places)
SYST-FOLEN	INT4	X	4	Utilización interna (longitud de salida de campo)
SYST-FTYPE	CHAR	C	1	Utilización interna (tipo de campo)

SYST-GROUP	CHAR	C	1	INTERNO: Concatenación
SYST-HOST	CHAR	C	8	Nombre de la máquina
SYST-INDEX	INT4	X	4	Cantidad de repeticiones de bucles
SYST-INPUT	CHAR	C	1	Utilización interna
SYST-LANGU	LANG	C	1	Clave de idioma para entrar al Sistema SAP
SYST-LDBPG	CHAR	C	8	PROGRAM: Programa ABAP/4 de base de datos para SY-DBNAM
SYST-LILLI	INT4	X	4	Número de la línea de lista actual
SYST-LINCT	INT4	X	4	Cantidad de líneas de lista
SYST-LINNO	INT4	X	4	Línea actual en la creación de una lista
SYST-LINSZ	INT4	X	4	Ancho de línea de la lista
SYST-LISEL	CHAR	C	255	INTERACT.: Línea seleccionada
SYST-LISTI	INT4	X	4	Número de la línea de lista actual
SYST-LOCDB	CHAR	C	1	Existe base de datos local
SYST-LOCOP	CHAR	C	1	Operación local en base de datos
SYST-LOOPC	INT4	X	4	Cantidad de líneas LOOP en steploop de dynpro
SYST-LPASS	CHAR	C	4	Utilización interna
SYST-LSIND	INT4	X	4	Número de la lista de bifurcación
SYST-LSTAT	CHAR	C	16	INTERACT.: Información de status por nivel de lista
SYST-MACDB	CHAR	C	4	PROGRAM: Nombre del fichero para el acceso con matchcode
SYST-MACOL	INT4	X	4	Number of columns from SET MARGIN
SYST-MANDT	CHAR	C	3	Número de mandante para entrar al Sistema SAP
SYST-MARKY	CHAR	C	1	Letra de línea actual para MARK
SYST-MAROW	INT4	X	4	Cantidad de líneas de instrucción SET MARGIN
SYST-MODNO	CHAR	C	1	Cantidad de modos alternativos
SYST-MSGID	CHAR	C	2	ID de mensaje
SYST-MSGLI	CHAR	C	60	INTERACT.: Línea de mensaje (línea 23)
SYST-MSGNO	NUMC	N	3	Número del mensaje
SYST-MSGTY	CHAR	C	1	Tipo de mensaje (E,I,W,etc.)
SYST-MSGV1	CHAR	C	50	Variable en mensaje
SYST-MSGV2	CHAR	C	50	Variable en mensaje
SYST-MSGV3	CHAR	C	50	Variable en mensaje
SYST-MSGV4	CHAR	C	50	Variable en mensaje
SYST-NEWPA	CHAR	C	1	Utilización interna
SYST-NRPAG	CHAR	C	1	Utilización interna
SYST-ONCOM	CHAR	C	1	Internal: ON COMMIT flag
SYST-OPSYS	CHAR	C	10	SYSTEM: Sistema operativo
SYST-PAART	CHAR	C	16	IMPRESION: Edición
SYST-PAGCT	INT4	X	4	Límite de página de lista en instrucción REPORT
SYST-PAGNO	INT4	X	4	RUNTIME: Página actual en creación de lista
SYST-PAUTH	NUMC	N	2	Utilización interna
SYST-PDEST	CHAR	C	4	IMPRIMIR: Dispositivo de salida
SYST-PEXPI	NUMC	N	1	IMPRIMIR: Tiempo de permanencia en SPOOL
SYST-PFKEY	CHAR	C	8	RUNTIME: Status de teclas-F actual
SYST-PLAYO	CHAR	C	5	Utilización interna
SYST-PLAYP	CHAR	C	1	Utilización interna
SYST-PLIST	CHAR	C	12	IMPRESION: Nombre de la orden SPOOL (nombre de lista)
SYST-PNWPA	CHAR	C	1	Utilización interna
SYST-PRABT	CHAR	C	12	IMPRIMIR: Departamento en la portada
SYST-PRBIG	CHAR	C	1	IMPRIMIR: Portada de selección
SYST-PRCOP	NUMC	N	3	IMPRIMIR: Cantidad de ejemplares
SYST-PRDSN	CHAR	C	6	IMPRIMIR: Nombre del set de datos SPOOL

SYST-PREFX	CHAR	C	3	Prefijo ABAP/4 para jobs batch
SYST-PRI40	CHAR	C	1	Utilización interna
SYST-PRIMM	CHAR	C	1	IMPRESION: Salida inmediata
SYST-PRINI	NUMC	N	1	Utilización interna
SYST-PRLOG	CHAR	C	1	Utilización interna
SYST-PRNEW	CHAR	C	1	IMPRESION: Nueva orden SPOOL (lista)
SYST-PRREC	CHAR	C	12	IMPRIMIR: Destinatario
SYST-PRREL	CHAR	C	1	PRINT: Delete after printing
SYST-PRTXT	CHAR	C	68	IMPRIMIR: Texto para portada
SYST-REPI2	CHAR	C	8	Utilización interna
SYST-REPID	CHAR	C	8	PROGRAM: Nombre de un programa ABAP/4
SYST-RSTRT	CHAR	C	1	Internal use
SYST-RTITL	CHAR	C	70	IMPRIMIR: Título de report del programa de impresión
SYST-SAPRL	CHAR	C	4	SISTEMA: Release SAP
SYST-SCOLS	INT4	X	4	Columnas en la pantalla
SYST-SFNAM	CHAR	C	30	Sin utilizar
SYST-SFOFF	INT4	X	4	Utilización interna
SYST-SLSET	CHAR	C	14	Nombre de SELECTON-SETS
SYST-SPONO	NUMC	N	5	RUNTIME: Número SPOOL para salida de una lista
SYST-SPONR	NUMC	N	5	RUNTIME: Número SPOOL de instrucción TRANSFER
SYST-SROWS	INT4	X	4	Líneas en la pantalla
SYST-STACO	INT4	X	4	INTERACT.: Lista visualizada a partir de la columna
SYST-STARO	INT4	X	4	INTERACT.: Lista visualizada a partir de línea
SYST-STEPL	INT4	X	4	Número de la línea LOOP en step dynpro
SYST-SUBCS	CHAR	C	1	INTERNO: Status call del report
SYST-SUBRC	INT4	X	4	Código de retorno posterior a determinadas operaciones
SYST-SUBTY	RAW	X	1	ABAP: Forma de llamada en SUBMIT
SYST-SYSID	CHAR	C	8	System: SAP System ID
SYST-TABID	CHAR	C	8	Utilización interna
SYST-TABIX	INT4	X	4	RUNTIME: Línea actual de una tabla interna
SYST-TCODE	CHAR	C	4	SESSION: Código de transacción actual
SYST-TFDSN	CHAR	C	8	RUNTIME: Nombre del set de datos para extractos de datos
SYST-TFILL	INT4	X	4	Cantidad actual de entradas en la tabla interna
SYST-TITLE	CHAR	C	70	PROGRAM: Título del programa ABAP/4
SYST-TLENG	INT4	X	4	Tamaño de la línea de una tabla interna
SYST-TLOPC	INT4	X	4	Utilización interna
SYST-TMAXL	INT4	X	4	Cantidad máxima de entradas en la tabla interna
SYST-TNAME	CHAR	C	30	Nombre de la tabla interna después de un acceso
SYST-TOCCU	INT4	X	4	Parámetro occurs en tablas internas
SYST-TPAGI	INT4	X	4	Indicador para almacenar tabla interna en bloque paging
SYST-TSTIS	INT4	X	4	Utilización interna
SYST-TTABC	INT4	X	4	Número de la última línea de tabla interna leída
SYST-TTABI	INT4	X	4	Offset de tablas internas en el área de roll
SYST-TVAR0	CHAR	C	20	RUNTIME: Variable de texto para elementos de texto ABAP/4
SYST-TVAR1	CHAR	C	20	RUNTIME: Variable de texto para elementos de texto ABAP/4
SYST-TVAR2	CHAR	C	20	RUNTIME: Variable de texto para elementos de texto ABAP/4
SYST-TVAR3	CHAR	C	20	RUNTIME: Variable de texto para elementos de texto ABAP/4
SYST-TVAR4	CHAR	C	20	RUNTIME: Variable de texto para elementos de texto ABAP/4
SYST-TVAR5	CHAR	C	20	RUNTIME: Variable de texto para elementos de texto ABAP/4
SYST-TVAR6	CHAR	C	20	RUNTIME: Variable de texto para elementos de texto ABAP/4
SYST-TVAR7	CHAR	C	20	RUNTIME: Variable de texto para elementos de texto ABAP/4

SYST-TVAR8	CHAR	C	20	RUNTIME: Variable de texto para elementos de texto ABAP/4
SYST-TVAR9	CHAR	C	20	RUNTIME: Variable de texto para elementos de texto ABAP/4
SYST-TZONE	INT4	X	4	Diferencia de tiempo con 'Hora media de Greenwich' (UTC)
SYST-UCOMM	CHAR	C	70	Interact.: Command field function entry
SYST-ULINE	CHAR	C	255	CONSTANT: Línea de subrayado (_____...)
SYST-UNAME	CHAR	C	12	SESSION: Nombre de usuario según entrada al Sistema SAP
SYST-UZEIT	TIMS	T	6	SYSTEM: Hora
SYST-VLINE	CHAR	C	1	CONSTANT: raya vertical
SYST-WAERS	CUKY	C	5	Moneda de sociedad tras leer segmento B
SYST-WILLI	INT4	X	4	Número de la línea de ventana actual
SYST-WINCO	INT4	X	4	Posición de cursor en la ventana (columna)
SYST-WINDI	INT4	X	4	Indice de la línea de ventana actual
SYST-WINRO	INT4	X	4	Posición de cursor en la ventana (línea)
SYST-WINSL	CHAR	C	79	INTERACT.: Línea en ventana seleccionada
SYST-WINX1	INT4	X	4	Coordenada de ventana (columna izquierda)
SYST-WINX2	INT4	X	4	Coordenada ventana (columna derecha)
SYST-WINY1	INT4	X	4	Coordenada ventana (línea izquierda)
SYST-WINY2	INT4	X	4	Coordenada de ventana (línea derecha)
SYST-WTITL	CHAR	C	1	Indicador para cabecera estándar de página
SYST-XCODE	CHAR	C	70	Código OK ampliado
SYST-XFORM	CHAR	C	30	Utilización interna (form SYSTEM-EXIT)
SYST-XPROG	CHAR	C	8	Utilización interna (programa SYSTEM-EXIT)

VARIABLES DEL SISTEMA EN REPORTING INTERACTIVO

SY-LSIND Número de la lista en proceso (lista básica = 0).

SY-CPAGE Página actual dentro de la lista en proceso.

SY-STARO Primera línea visualizada de esta página.

SY-STACO Primera columna visualizada (SCROLL LIST)

Las siguientes variables del sistema cambian únicamente si el usuario modifica el tamaño de una ventana.

SY-SROWS Número de líneas del listado en la ventana actual.

SY-SCOLS Número de columnas en el listado con la ventana actual.

Variables de línea.

SY-LISEL Contenido de una línea del listado.

SY-LILLI Número de la línea en curso.

SY-LISTI Número de la lista (LSIND) para la cual SY-LILLI está determinada.

La opción CURRENT LINE de la sentencias READ LINE y MODIFY LINE siempre se refiere a la línea <SY-LILLI> de la lista <SY-LISTI>.

Los siguientes variables se refieren a la posición del cursor en la lista.

SY-CUROW Número de la línea donde se encuentra el cursor.

SY-CUCOL Número de la columna donde se encuentra el cursor.

ANEXO 4 PRINCIPALES TABLAS

Este anexo presenta las principales tablas con que cuenta un sistema SAP R/3. Como tablas principales entendemos aquellas que contienen datos maestros y cuyos datos son vitales para la empresa. Por lo tanto, serán aquellas que más a menudo se verán envueltas en la programación en ABAP/4. La interpretación funcional de sus campos y contenidos variará mucho en función de los módulos instalados y del customizing realizado, pero su gran importancia hará que siempre estén presentes y que deban utilizarse en la programación para cualquier módulo.

Sin embargo, no debe entenderse este documento como exhaustivo. SAP R/3 es un sistema con miles de tablas y las aquí listadas deben ser consideradas una pequeña parte. Sólo la experiencia y una estrecha colaboración con los analistas funcionales llevarán a un conocimiento más profundo de las tablas del sistema.

Tablas:

- **T001**: Tabla que contiene los datos maestros de las diferentes sociedades dadas de alta en el sistema.
- **T002**: Claves de los idiomas presentes en el sistema.
- **T003**: Claves de los diferentes tipos de documentos que el sistema acepta, así como información sobre sus sistema de numeración.
- **T004**: Tabla con los diferentes planes de cuentas que contiene el sistema.
- **T005**: Información sobre todos los países dados de alta.
- **T006**: Unidades de medida disponibles.
- **TCURC**: Códigos de moneda de R/3, así como los alternativos e ISO.
- **BKPF**: Tabla que contiene los datos de cabecera de todos los documentos contables del sistema.
- **BSEG**: Tabla con los datos de todas las posiciones de cada uno de los documentos contables que se encuentran en la tabla BKPF.
- **KNA1**: Tabla con los datos maestros de los clientes a nivel general, es decir, aquellos datos que son independientes de la sociedad.
- **KNB1**: Tabla que contiene aquellos datos de los clientes que varían en función de la sociedad.
- **LFA1**: Tabla con datos análogos a la KNA1, pero refiriéndose a los proveedores.
- **LFB1**: Datos de proveedores a nivel de sociedad.
- **SKA1**: Tabla con los datos maestros de las cuentas de mayor.
- **SKB1**: Tabla con los datos de las cuentas de mayor a nivel de sociedad.
- **SKC1A**: Estructura que contiene, una vez llenada por la base de datos lógica o por la función READ_SCK1A, las cantidades en el debe, en el haber y los saldos desglosados por periodos contables de una cuenta de mayor.
- **MARA**: Datos maestros de los materiales dados de alta en el sistema.
- **MARC**: Datos centro p.material
- **EKKO**: Cabecera del documento de compras
- **EKPO**: Posicion del documento de compras
- **RBKP**: Cabecera documento factura recibida
- **RSEG**: Posición de documento factura recibida
- **VBAK**: Tabla con los datos de cabecera de los pedidos realizados.
- **VBAP**: Tabla con los datos de todas las posiciones de los pedidos contenidos en la tabla anterior.
- **LIKP**: Datos de cabecera de los documentos de entrega.
- **LIPS**: Datos de las posiciones de los documentos de entrega contenidos en LIKP.

- **VBRK**: Tabla que contiene los datos de cabecera de las facturas creadas por el sistema.
- **VBRP**: Tabla con las posiciones de todas las facturas existentes en el sistema.

Ayuda en la búsqueda de tablas:

Encontrar donde guarda el sistema determinados datos no es tarea fácil. Sin embargo, estas pequeñas pistas nos pueden simplificar la tarea:

- Las tablas que contienen extensiones de los datos de una tabla tienen nombres similares, variando únicamente la última letra. De esta forma, podemos obtener todas las tablas que hacen referencia a países tecleando T005* en el diccionario de datos y pulsando F4. Otro ejemplo lo tenemos en las tablas que contienen las descripciones de un código de R/3, cuyo nombre suele ser el mismo que la tabla con los datos maestros pero acabado en T. Si T005 es la tabla con los diferentes códigos de países, T005T contiene los nombres de esos países.
- Aquellas tablas asociadas a otras tablas o que contienen datos relacionados suelen tener nombres cuyas dos primeras letras son iguales. Así, las tablas con datos referidos a clientes empezarán por KN, como KNA1 y KNB1.
- En caso de conocer un campo que forma parte de la tabla que estamos buscando, podemos comprobar las tablas de las que forma parte mediante la opción de buscar referencias en el diccionario de datos. Esta opción, sin embargo, puede generarnos un resultado muy grande del cual sea difícil obtener la información que necesitamos.
- Existe también la posibilidad de consultar al diccionario de datos mediante la descripción de las tablas. De esta forma podemos obtener todas las tablas en cuya descripción existe una determinada palabra. Recordemos que se pueden utilizar los asteriscos para representar cualquier cadena y, por lo tanto, una consulta con ‘*cliente*’ sobre las descripciones de las tablas obtendrá todas las tablas que contengan la palabra ‘cliente‘ en su descripción.

ANEXO 5 PRINCIPALES TRANSACCIONES

Este anexo presenta las principales transacciones que un programador ABAP tendra que utilizar para desarrollar su trabajo. No son las unicas, pero si las mas importantes y asiduas.

- **SE10**: Transport Organizer. Permite transportar ordenes a test y a productivo
- **SE11**: Data Dictionary. Para crear y modificar tablas, vistas, dominios, elementos…
- **SE16**: Data browser. Permite ver los datos que hay en las distintas tablas
- **SE24**: Browser de clases. Nos permite crear, visualizar y modificar transacciones.
- **SE30**: Analisis de tiempo ejecución de programas, funciones y transacciones
- **SE35**: Split Screen. Visualizacion o comparación de dos programas en pantalla separada.
- **SE37**: Biblioteca de funciones, para crear y modificar modulos de funcion.
- **SE38**: Editor ABAP, para crear y modificar programas, preferiblemente de tipo ejecutable.
- **SE41**: Menu Painter, donde podremos diseñar menus y barras de herramientas para los programas.
- **SE51**: Screen Painter. Permite crear pantallas para nuestros programas de dialogo.
- **SE71**: Form Painter. Nos deja crear formularios de tipo SapScript para la impresión de documentos.
- **SE80**: Object Navigator. Creación y modificacion de programas de dialogo, clases de desallorro, grupos de funciones y clases.
- **SE91**: Actualizacion de mensajes. Para crear clases de mensajes y mensajes.
- **SE93**: Transacciones. Desde aquí se pueden crear transacciones.
- **BAPI**: Navegador de Bapis, para acceder y visualizar todas las bapis del sistema.
- **SMARTFORMS**: Diseño de formularios de tipo Smartform.
- **SM35**: Batch Input. Grabacion y ejecución de batch inputs.
- **SM37**: Jobs. Monitorización de Jobs. Creación, visualizacion y control de errores.
- **SP01**: Control de salida del spool. Ordenes de impresión SAP.