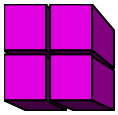


NORMAS PARA LA PROGRAMACIÓN EN COBOL

NORMAS PARA LA PROGRAMACIÓN EN COBOL

Introducción

Este documento contiene una serie de normas y recomendaciones de uso general en la instalación para estandarizar la programación en lenguaje Cobol, de modo que se faciliten las tareas de comprensión y mantenimiento del código generado.

1#

NORMAS DE PROGRAMACIÓN EN COBOL

CODIGO **TEMA**

NPRCOB05 Consideraciones Generales

Se consideran una serie de apartados a tener en cuenta:

- De forma general, se recomienda estructurar el código de manera que sea fácil su seguimiento, utilizando nombres de párrafos y variables que tengan un significado lógico.

- Es imprescindible que se siga un método de programación estructurada. De acuerdo a la metodología de desarrollo adoptada por Banco Santander, se empleará el método **Warnier** o **Bertini .**

  Este método es eficiente incluso en los programas de lógica compleja. De hecho cuanto mas complicado es un programa, mas ayudan las técnicas de programación estructurada en el proceso de codificación. Además, el Warnier o el arbol programático de Bertini supone una parte **muy importante** de la documentación del mismo.

- Modularizar en párrafos las sentencias comunes, para facilitar el mantenimiento de los programas. Es decir, todos aquellos párrafos que tienen un código similar se estructurarán unificando la parte común en un único párrafo que será llamado desde el resto de los párrafos. Con esto se pretende disminuir el impacto de las modificaciones dentro de los programas.

  Del mismo modo, las estructuras de datos comunes a varios programas (registros, áreas de enlace, etc.) se escribirán como COPY's a incluir en los programas.

- Una cuestión de interés para el mantenimiento de las aplicaciones es que una modificación repercuta en el mínimo número de cambios posibles. En este sentido, hay que utilizar de forma obligatoria las COPY's estándar que existen en la instalación.

- Crear las definiciones de registros en la librería de COPYS, siguiendo el estándar de nomenclaturas, para que dichos registros puedan ser compartidos por el resto de los programas

- No realizar SORT internos en el programa, sino en JCL.

- Un programa debe tener **un solo** punto de salida, es decir, una única sentencia STOP RUN, GOBACK o RETURN.

1

NPRCOB01

- En cuanto a la inclusión de **comentarios** se tendrán en cuenta las siguientes consideraciones:

  - En vez de crear comentarios que describan solo procesos técnicos, debe tenderse a incorporar comentarios descriptivos de la función.

  - Los comentarios que se incluyan deben ser claros, concisos y suficientemente señalados para hacerlos fácilmente distinguibles de las instrucciones del programa.

- Verificar siempre los resultados de la compilación, de cara a eliminar todo campo no referenciado en PROCEDURE.

NORMAS DE PROGRAMACIÓN EN COBOL

CODIGO **TEMA**

NPRCOB10 Identification Division

- PROGRAM-ID : Debe coincidir el nombre del programa.
- AUTHOR : Completar siempre el apartado.Se deberá indicar el nombre de la persona que lo programa (nombre y apellidos) y el nombre de la empresa a la que pertenece.
- Describir, antes de dicha identificación y con * (asterisco) en columna 7, las funciones del programa resumidamente, con especial mención a la parte de procedimiento o lógica del programa (siempre que se pueda, evitar que la descripción exceda de 12 líneas).
- No incluir resto de claúsulas, tales como DATE COMPILER, etc ...
- No incluir comentarios de las modificaciones efectuadas al programa original. Dichos comentarios se deben incluir en los **motivos** que se solicitan en Librarian al efectuar el pase del programa a producción.

NORMAS DE PROGRAMACIÓN EN COBOL

CODIGO	TEMA
NPRCOB15	Environment Division

º SPECIAL-NAMES : Es obligatorio el uso de la claúsula "DECIMAL POINT IS COMMA", para utilizar la coma como separador decimal.

NORMAS DE PROGRAMACIÓN EN COBOL

CODIGO	TEMA
NPRCOB20	Data Division

Se consideran los siguientes apartados:

Apartado 1 - File Section

Apartado 2 - Working-Storage Section

- 2.1 Estructura de la Working-Storage
- 2.2 Recomendaciones en el uso de Campos Numéricos
- 2.3 Niveles 88 - Campos de Control

Apartado 3 - Linkage Section

NPRCOB20 - Apartado 1

File Section

- Delante de cada sentencia FD se pondrá un comentario relativo al fichero (su función y proposito, particularidades, ...).

- Ordenar las FD's en la misma secuencia que los estamentos SELECT de la Environment Division.

- Dejar una linea libre entre las FD's.

- En la sentencia FD solo se definirá el nivel 01 del registro asociado, dejando la definición completa del registro para la Working-Storage.

- Para estandarizar el nombre de los registros asociados a ficheros, se recomienda definir el registro de nivel 01 de la FD como: ´REG-FFFFFFFF`, siendo ´FFFFFFFF` el nombre del fichero indicado en la FD, que corresponderá al quinto cualificador del fichero (Ver Estándares TSO - "Librerias y ficheros de aplicación").

- La codificación de las siguientes claúsulas es obligatoria:

  - ´BLOCK CONTAINS 0 RECORDS`: Utilizarla en todo tipo de ficheros, para dejar al JCL la gestión del bloque óptimo según tipo de dispositivo (el factor de bloqueo es dado en tiempo de ejecución en el JCL).Se codificará siempre con cero.

  - ´RECORDING MODE IS (modo)`: En (modo) se pondrá F, V o U según que los registros sean de longitud fija, variable o indefinida, respectivamente. Lo indicaremos sólo en el caso de que sea distinto de F.

  - ´LABEL RECORD STANDARD`: Solo se pondrá OMITTED para cintas dirigidas a otras instalaciones.

NPRCOB20 - Apartado 2

Working-Storage Section

- Cada campo interno del programa debe tener un solo uso, y su nombre debe sugerir esta utilización. No utilizar nombres que no identifiquen claramente la función del campo (tales como I, J, CAMPO-1, PEPE, etc.).

- La numeración de los niveles comenzará por 01, continuando por multiplos de cinco: 01, 05, 10, 15, ...

- La abreviación ´PIC` es estándar. Toda cadena de caracteres de la clausula PICTURE se escribirá entre parentesis. Por ejemplo, PIC S9(4)V99, en vez de S9999V99.

- Se procurará alinear en la misma columna, por un lado, las clausulas PIC y, por otro, las VALUE.

- No se utilizará el nivel 77, sustituyéndolo por el nivel 01.

- Constantes y variables deben denominarse con prefijos,del modo siguiente:

Constante Numérica	**CN-XX....**
Constante Alfanumérica	**CA-XX....**
Variable Numérica	**VN-XX....**
Variable Alfanumérica	**VA-XX....**

- Todas las tablas se han escribir con índice INDEXED BY, para su utilización posterior en la Procedure.

- Es muy importante poner todas las tablas juntas al final de la WORKING, para controlar desbordamientos (y prever índices para evitarlos).

- No se utilizarán los literales en la PROCEDURE DIVISION. Cuando se necesite una constante se debe definir en la WORKING-STORAGE con su valor, no estando su nombre relacionado con el valor sino con el significado de la constante.

- Se debe usar siempre las constantes figurativas en lugar de los literales correspondientes, tanto en VALUE's como en MOVE's.

CORRECTO	INCORRECTO
ZEROS	0,000

- Las variables que se inicialicen una sola vez durante el programa, se deben definir en la WORKING-STORAGE Section, utilizando una claúsula VALUE, en vez de utilizar sentencias MOVE o SET en la PROCEDURE DIVISION. La claúsula VALUE sólo necesita un paso en compilación, en cambio la MOVE, además de la compilación utiliza tiempo de ejecución, en cada paso de programa.

- Por razones de eficiencia, las variables numéricas que sean utilizadas como índices se definirán en formato binario S9(4) COMP o S(9) COMP (2 y 4 bytes respectivamente).De esta manera se facilita la labor del Cobol, que, para trabajar con índices, necesita convertirlos a binario.

- No hace falta completar las lineas de listado hasta las 132 posiciones con FILLER PIC X (nn) VALUE SPACES, ya que el mover un campo alfanumérico de menor longitud a uno de mayor provoca el “rellenado” a espacios de los excedentes del campo receptor, ademas se evita la tarea de contar las posiciones que restan hasta las 132, evitando asi posibles errores de compilación.

NPRCOB04 - Apartado 2.1

Estructura de la Working-Storage

- Los campos se agruparán en bloques en función de su naturaleza y caracteristicas. Para ello se utilizará la siguiente estructura y nomenclatura, incluidos los comentarios que anteceden a cada bloque, y procurando mantener el orden aquí expuesto:

```
*****************************************************************************************************
*                                          VARIABLES                                                *
*****************************************************************************************************
01   VN-VARIABLES-NUMERICAS.
     05 VN-.....

01   VA-VARIABLES-ALFANUMERICAS.
     05 VA-.....


*****************************************************************************************************
*                                          CONSTANTES                                               *
*****************************************************************************************************

01   CN-CONSTANTES-NUMERICAS.
     05 CN-.....

01   CA-CONSTANTES-ALFANUMERICAS.
     05 CA-.....


*****************************************************************************************************
*                                          CONTADORES                                               *
*****************************************************************************************************

01   CO-CONTADORES.
     05 CO-.....


*****************************************************************************************************
*                                          ACUMULADORES                                             *
*****************************************************************************************************
01   AC-ACUMULADORES.
     05 AC-.....


*****************************************************************************************************
*                                          CAMPOS DE CONTROL                                        *
*****************************************************************************************************
01   CC-CAMPOS-CONTROL.
     05 CC-.....


*****************************************************************************************************
*                                          INDICES                                                  *
*****************************************************************************************************
01   IN-INDICES.
     05 IN-.....


*****************************************************************************************************
```

```
*                                          PUNTEROS                                          *
*************************************************************************************************************
01   PT-PUNTEROS.
     05 PT-.....

*************************************************************************************************************
*                                          LITERALES                                         *
*************************************************************************************************************
01   LI-LITERALES.
     05 LI-.....

*************************************************************************************************************
*                                          MENSAJES                                          *
*************************************************************************************************************
01   MM-MENSAJES.
     05 MM-.....
```

A continuación de las áreas anteriores se incluirá, siempre que se necesite, la siguiente información:

- Definición de ficheros: un nivel 01 por cada fichero. El nombre del nivel 01 será igual al del registro de la FD, añadiendo ´W-`. Esto es:

  01 REG-**FFFFFFFF**-W.

  (**FFFFFFFF** es el nombre que se dió al fichero en la FD)

- Tablas internas de Cobol (en las que se utilice OCCURS). Se indicará que es una tabla con el prefijo ´TB-`. Es conveniente reflejar, en el nombre del registro de la tabla (que acabará en ´-REG` y del índice (que acabará en ´-IND`), la tabla a la que pertenecen. Por ejemplo:

```
01  TB-CUENTAS.
    05  TB-CUENTAS-REG  OCCURS 10 TIMES
                         INDEXED BY TB-CUENTAS-IND.
        10 TB-CUENTAS-CUENTA      PIC X(5).
        10 TB-CUENTAS-TIPO        PIC X(2).
        10 TB-CUENTAS-SALDO       PIC 9(10).
```

- Máscaras de edición de informes:

  ´CAB**n**-**FFFFFFFF** . .` para las cabeceras,
  ´DET**n**-**FFFFFFFF**. . .` para las líneas de detalle,
  ´TOT**n**-**FFFFFFFF**. . .` para las líneas de total,

  donde ´**n**` es un número correlativo que comenzará por 1 para las cabeceras, líneas de detalle y líneas de total, respectivamente.y **FFFFFFFF** el nombre que se le dió al fichero en la FD.

- COPY's e INCLUDE's de registros, tablas DB2, rutinas, etc., agrupándolas.

- Declaración de cursores DB2 (DECLARE). Se hará siempre en la WORKING. La nomenclatura de cursores será: ´CSR-**XXXXXXXX**-**nn**`, donde **XXXXXXXX** es el nombre de la vista y ´**nn**` una secuencia alfanumérica. Si se trata de un JOIN: ´CSR-**J**-**YYY. . .**-nn`, donde ´**J**` indica JOIN e **YYY. . .** un nombre libre (por ej., el nombre de una de las tablas, el nombre abreviado de todas ellas, o cualquier otro nombre que se considere adecuado).

- Precediendo a cada uno de estos grupos de información o a cada nivel 01, se pondrán comentarios visibles, a modo de cabecera, que sirvan de título a lo que va a continuación.

  Ejemplo:

```
*****************************************************************************
*                   DEFINICION DE REGISTROS DE ENTRADA
*****************************************************************************
*                   (Comentario opcional si se considera oportuno)
*****************************************************************************
01  REG-ENTRADA1-W.
    05 . . ./. . .
```

- Para facilitar la lectura y el mantenimiento del código, no continuar el literal de una clausula VALUE en la línea siguiente. En su lugar, codificar un nuevo FILLER y continuar el literal.

  Ejemplo:

```
01  MM-MENSAJES.
    05 MM-ERROR-DEPARTAMENTO.
       10 FILLER       PIC X(45)        VALUE
          ´INFORME NUM. N. NO EXISTE EL CODIGO DE DEPART`.
       10 FILLER       PIC X(30)        VALUE
          ´AMENTO INTRODUCIDO`.
```

NPRCOB20 - Apartado 2.2

Recomendaciones en el uso de Campos Numéricos

Siempre y cuando no intervengan en cálculos, se deberán utilizar campos alfanuméricos.

- Utilizar la picture (USAGE) y longitud adecuada en campos numéricos:
  El compilador realiza conversiones de formato de campos numéricos en las sentencias MOVE, comparaciones y operaciones aritméticas para adaptarlos entre sí y para procesar las sentencias, con el consiguiente gasto de memoria y tiempo de ejecución.
  Es usual realizar operaciones de conversión innecesarias por el uso de pictures diferentes en los operadores y resultado cuando, en la mayoria de los casos, se podría evitar.
  Las operaciones con campos numéricos son mas eficientes cuando el campo receptor tiene el mismo tipo, longitud, posición de signo y posiciones decimales, que el campo origen. Por tanto, se evitará mezclar tipos de datos diferentes en las operaciones.

- Consideraciones sobre el rendimiento de las diferentes pictures: de mayor a menor: COMP, COMP-1, COMP-2, COMP-3, DISPLAY.

- Consideraciones sobre algunas pictures:

  . DISPLAY: Para operaciones aritméticas es ineficiente, ya que el compilador lo convierte a empaquetado para realizar la operación y luego reconvierte el resultado a DISPLAY. Se suele usar para mostrar datos. Ocupa un byte por dígito.

  . PACKED-DECIMAL (COMP-3): Se suele usar en áreas de trabajo.
  El número de dígitos debe ser impar para un uso más eficiente. (Ocupa un byte por cada dos dígitos mas ½ byte para el signo).
  En operaciones aritméticas, el compilador convierte el dato a COMP si es menor o igual a 9 dígitos.

  . BINARY (COMP, COMP-4): Se suele utilizar en índices y campos aritméticos.
  Es la representación más eficiente para campos de menos de 10 dígitos; en operaciones aritméticas, el compilador convierte el dato a COMP-3 si es mayor de 18 dígitos.

  El número de bytes que ocupa un campo binario es:

  . 2 bytes si 0 < **N** < = 4;
  . 4 bytes si 4 < **N** < = 9;
  . 8 bytes si 9 < **N** < = 18;

  donde ´**N**` representa el número de dígitos.

  La clausula SYNCHRONIZED en campos binarios alinea a media palabra o palabra completa, lo cual mejora el rendimiento.

Las operaciones de redondeo son más complejas.

- COMA FLOTANTE (COMP-1 y COMP-2): Para trabajar con muchos dígitos significativos.

º Por lo tanto se recomienda definir los campos numéricos que se vayan a usar en cálculos aritméticos como COMP (si <= 9 dígitos) o COMP-3 (si > 9 dígitos) y con signo.

º Cuando se realicen cálculos con campos DISPLAY o se emplean dos campos en diferente formato, la eficiencia se incrementa si los datos se convierten una vez, en lugar de varias veces.

Ejemplo:

```
05  A  PIC 9(2).
05  B  PIC 9(2)    COMP-3.

MOVE A TO B     ==>  Convertimos A en un formato COMP-3, y a partir de
                     entonces, efectuamos las operaciones sobre B.
ADD CTN-1 TO B
ADD CTN-9 TO B.
```

en vez de:

```
ADD CTN-1 TO A. ==> Se ha de realizar una conversión en cada operación.
ADD CTN-9 TO A.
```

º Evitar OVERFLOW en las operaciones aritméticas, que puede dar lugar a resultados impredecibles. Para ello:

- Calcular la longitud necesaria para los campos intermedios y de resultado.
- Hacer el test ON SIZE ERROR, si es necesario. Hay que tener en cuenta que incorporar esta cláusula requiere un tiempo de proceso extra, ya que el compilador genera código adicional, para que, durante la ejecución, se compruebe el OVERFLOW.

º Al utilizar COMPUTE, puede existir truncamiento si hay resultados intermedios de gran longitud. Cuando los campos decimales tienen mas de 30 dígitos, el campo es truncado a 30, con lo que se pueden obtener resultados finales erroneos.
Si se quiere un resultado intermedio que exceda a 30 dígitos, se puede emplear COMA FLOTANTE (COMP-1, COMP-2).

NPRCOB20 - Apartado 2.3

Niveles 88 - Campos de Control

- Cuando un campo puede tomar valores o rangos significativos, estos se indicarán expresamente a través de niveles 88. Los nombres asociados a los niveles 88 se utilizarán obligatoriamente en la Procedure Division en lugar de hacer referencia explicita al valor del campo.

- Es conveniente que los nombres de los niveles 88 incluyan claramente la condición a la que se refieren, y, en su caso, al campo de control al que pertenecen. Y en caso de que las condiciones sean de afirmación/negación, incluir el prefijo SI/NO al nombre de la condición para que no haya lugar a confusiones (en este caso usar los valores lógicos ´S` para SI y ´N` para NO). Por ejemplo:

```
05  CC-TIPO-CUENTA          PIC X(1).
    88  TIPO-CUENTA-ACTIVO          VALUE ´A`.
    88  TIPO-CUENTA-PASIVO          VALUE ´P`.


01 CC-FIN-FICHEROS

   05  CC-FIN-FFFFFFFF              PIC X(1) VALUE  'N'.
       88  FIN-FFFFFFFF             VALUE ´S`.
       88  NO-FIN-FFFFFFFF          VALUE ´N`.
```

- Habrá de limitarse en lo posible el uso de campos de control. Si el campo de control debe contener un valor inicial, se inicializará en la Working-Storage.

NPRCOB20 - Apartado 3

Linkage Section

Será de aplicación en esta sección todo lo dicho para la Working-Storage Section.

```
**********************************************************
WORKING STORAGE SECTION.
        01  LK-PARAMETROS.
            05 LK-......
            05 LK-.....
        -----
        -----
        -----


PROCEDURE DIVISION USING LK-PARAMETROS.


**********************************************************
```

Todos los campos irán con prefijos **LK.**

NORMAS DE PROGRAMACIÓN EN COBOL

CODIGO **TEMA**

NPRCOB25 Procedure Division

Se consideran los siguientes apartados:

Apartado 1 - Estructura de la Procedure Division

Apartado 2 - Sentencias

Apartado 3 - Puntuación Simbología

Apartado 4 - Nomenclatura de párrafos

Apartado 5 - Estructura y jerarquia de párrafos

Apartado 6 - Utilización de la sentencia PERFORM

6.1 Bucles

Apartado 7 - Utilización de sentencias condicionales

Apartado 8 - Uso de variables, constantes, niveles 88 y campos de control

Apartado 9 - Tablas Cobol

Apartado 10 - Integridad Referencial, cardinalidad

NPRCOB25 - Apartado 1

Estructura de la Procedure Division

º La Procedure Division debe estructurarse en tres bloques bien diferenciados:

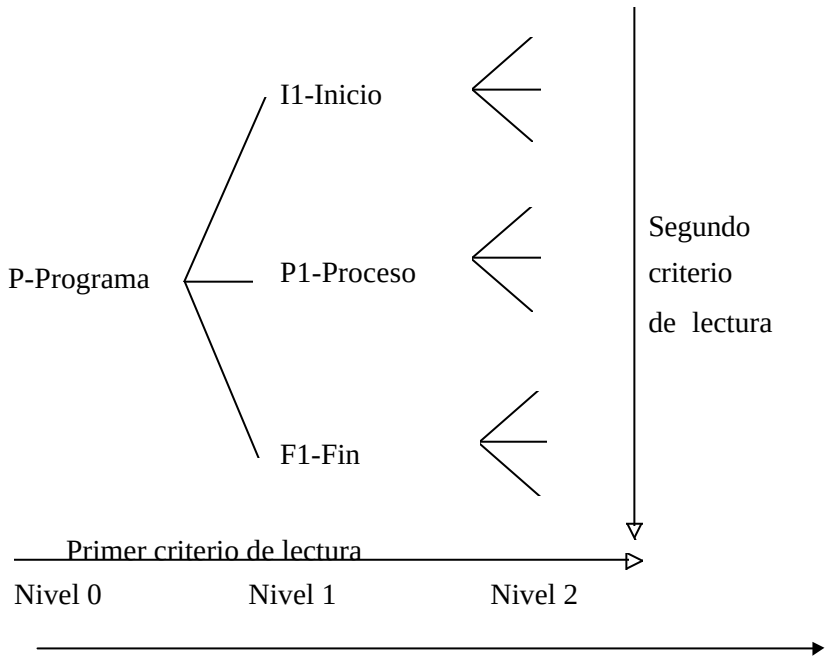
- ♦ **ESTRUCTURA**
- ¨ **TRATAMIENTOS**
- ♦ **TRATAMIENTOS AUXILIARES**

º La codificación de dichos bloques será secuencial y en el orden establecido.

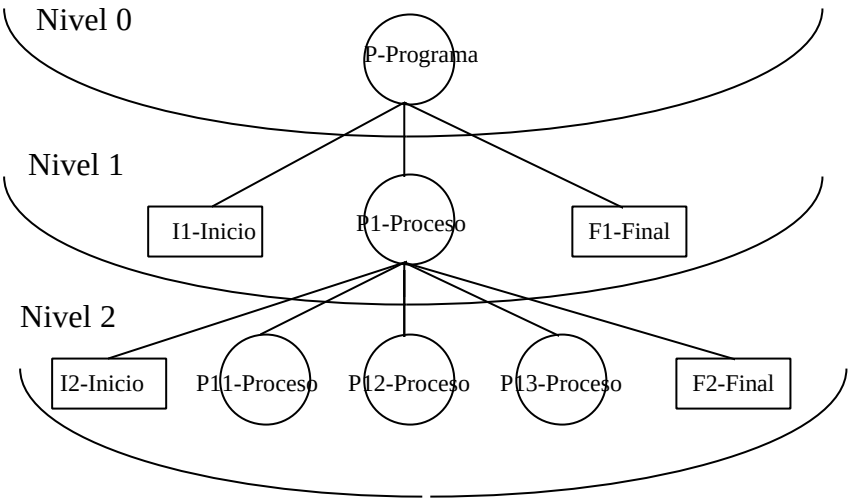
ESTRUCTURA

º Reflejará la lógica del diseño Warnier o del árbol programático de Bertini, según las correspondientes metodologías. En dicha estructura se reflejarán los niveles del diseño según ejemplo:

WARNIER

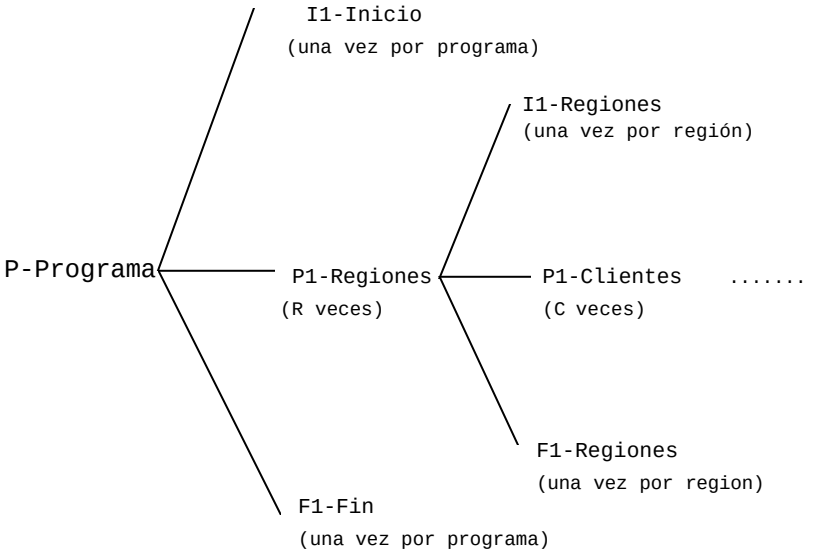


BERTINI

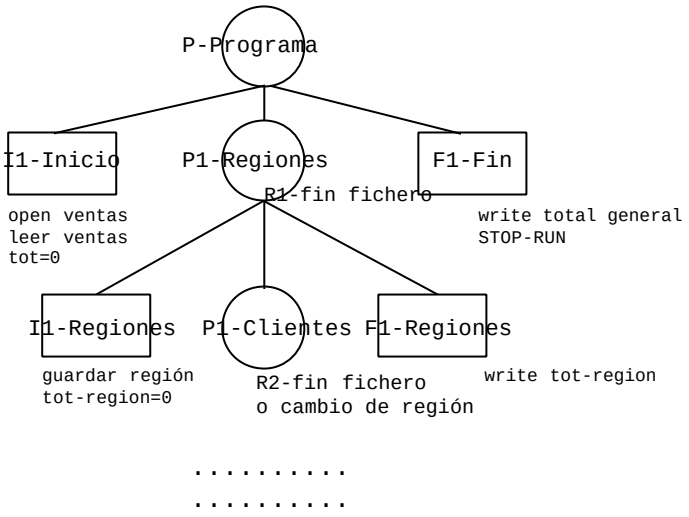


Ejemplos:

WARNIER



BERTINI



º El orden de codificación de párrafos del bloque de ESTRUCTURA, se realizará siguiendo los siguientes criterios:

WARNIER

1. De izquierda a derecha por niveles
2. De arriba a abajo por niveles.

BERTINI

1. De arriba a abajo por niveles
2. De izquierda a derecha por niveles.

El bloque de la ESTRUCTURA no debe contener otro tipo de instrucciones más que : PERFORM, PERFORM UNTIL, PERFORM VARYING, STOP RUN, GOBACK, o cualquier combinación de ellas con IF,ELSE.

```
PROCEDURE DIVISION
**************************************************
*******                ESTRUCTURA                **
**************************************************
****                      NIVEL 0 *
**************************************************

P-SKC740
  PERFORM I1-INICIO
  PERFORM P1-PROCESO1
```

```
   ...PERFORM P1-PROCESO2
      PERFORM F1-FIN
   ...STOP RUN (o GOBACK)
    ********************************************************
    ****                              NIVEL 1 **
    ********************************************************
    P1-PROCESO1
        PERFORM P11-
        IF (condición)
            PERFORM P12-
        ELSE
         ....PERFORM P13-
         PERFORM P14-
    P1-PROCESO2
.....PERFORM P15-
       IF (condición)
          PERFORM P16-                    UNTIL (condición)
       PERFORM P17-
```

TRATAMIENTOS

- Debe codificarse como segundo bloque de PROCEDURE DIVISION después del de ESTRUCTURA.
- Contiene los párrafos con el último nivel de sentencias.
- No debe contener, por tanto,sentencias PERFORM out-line, salvo las de llamadas a Tratamientos Auxiliares.

```
**********************************************************
***                    TRATAMIENTOS                    **
**********************************************************
I1-INICIO
   OPEN (ficheros)
   READ (ficheros)
        (primera lectura de ficheros, las restantes se harán desde los
          "tratamientos auxiliares")


P11-.....
   MOVE ...
   IF.....
   END-IF
   COMPUTE
   PERFORM A1-CALCULAR-CONTRAVALOR
   IF....
      MOVE
   END-IF

P12-....
      ......
      ......
      .  ...

P13-....
      ......
      ......
      .  ...

P14-....
      ......
      ......
      .  ...

P15-....
      ......
      ......
      .  ...

P16-....
      ......
      ......
      .  ...

F1-FIN-....
  CLOSE (ficheros)
```

TRATAMIENTOS AUXILIARES

º Debe codificarse como último bloque en la estructura de PROCEDURE DIVISION.

º Debe ajustarse a las siguientes reglas:

- Unicamente deben codificarse tratamientos auxiliares que por su carácter repetitivo deban llamarse ***desde varios párrafos distintos de TRATAMIENTOS***

- No utilice nunca la inclusión de TRATAMIENTOS AUXILIARES para:
  - ♦ Párrafos que por mal diseño de la estructura y los tratamientos se le quiera dar solución en este bloque.
  - ♦ Montar estructuras de ***TRATAMIENTOS*** en este bloque de auxiliares.

```
**********************************************************
***              TRATAMIENTOS AUXILIARES                 **
**********************************************************

A1-FETCH-CURSOS.

   EXEC SQL
      FETCH CR-CURSOS
      INTO :TOA01-CURSOCOD
   END-EXEC
   MOVE SQLCODE TO DB2-SQLCODE
....//....

A2-LEER-CURSOS.

 READ FENTRAD4
         AT END SET FIN-FICURSOS TO TRUE.
  A2-LEER-CURSOS-EXIT.
EXIT.
```

El orden de los tratamientos se hará por orden "*envolvente*" del último nivel de estructura, tanto en Warnier como en Bertini.

Será obligatorio el uso de la técnica de *lectura adelantada,* esta técnica, básica en cualquier diseño estructurado, consiste en distinguir, para cada fichero secuencial de entrada, entre:

- Primer registro
- Resto de registros

La lectura del primer registro se efectuará **una sola vez** en el programa y se debe hacer **inmediatamente después de abrir el fichero**.
La lectura del resto de los registros se efectuará **inmediatamente después** de terminar el tratamiento del registro anterior.

NPRCOB25 - Apartado 2

Sentencias

- No codificar más de una sentencia por línea. Si una sentencia requiere múltiples líneas, endentar las líneas de continuación al menos tres caracteres a la derecha, rompiendo a nivel de las claúsulas si es posible, para facilitar la lectura del código. las nuevas sentencias comenzarán en la columna 12.

- La lectura de varios estamentos es más fácil si se alinean verticalmente.
  Ejemplo:

```
MOVE  SPACES  TO NOMBRE-CAMPO-1
                 NOMBRE-CAMPO-2

OPEN OUTPUT       FICHERO-1
                  FICHERO-2
             FICHERO-3
```

- Está prohibida la utilización de sentencias GO TO y ALTER, ya que rompen la estructuración del programa.

- No es recomendable el uso de los verbos ACCEPT (usar sólo lo imprescindible), BASIS, CANCEL, ENTRY, ni de MOVE (o ADD o SUBTRACT...) CORRESPONDING.

- No utilizar SORT, ni MERGE ni RELEASE internos.

- No usar DISPLAY ni DISPLAY UPON

- Los parámetros STATE,FLOW,COUNT y SYMDMP, y las facilidades TRACE y EXIBIT, sólo se usarán en pruebas.

- Procurar hacer una sola sentencia OPEN al inicio del programa, y una CLOSE al final del mismo, previa al STOP RUN, para todos los ficheros del programa. Así mismo, utilizar un sólo párrafo READ y un sólo párrafo WRITE para cada tipo de acceso a cada fichero.

- Las CALL deben ser dinámicas.

- Si se accede a variables del sistema como DATE, DAY, TIME, CURRENT DATE o TIME-OF-DAY, obtener el valor de la variable una sóla vez en la rutina de inicialización, y especificarlo en un campo de trabajo de la WORKING STORAGE para ser referenciado posteriormente.

- Se recomienda la utilización del verbo SEARCH para las búsquedas en tablas. Si la tabla está clasificada y tiene más de 100 elementos usar SEARCH ALL. Para cualquier tamaño de tabla poner los datos más frecuentes en los primeros elementos de la tabla, si es posible.

- Es aconsejable utilizar la sentencia COMPUTE; es una instrucción eficiente y fácil de leer. En las expresiones aritméticas complejas el COMPUTE genera automáticamente todas las variables intermedias con la longitud apropiadas.

- Utilizar siempre las sentencias que disponen de cierre, como IF y END-IF, EVALUATE y END-EVALUATE, READ y END-READ, SEARCH y END-SEARCH...

- No utilizar COPYS en PROCEDURE, a excepción de las obligatorias de los esqueletos y estándares de ARQUITECTURA. Utilizar en su lugar módulos dinámicos.

- Las llamadas a módulo siempre se codificarán de forma dinámica.

Puntuación y Simbología

- No utilizar comas.

- Se recomienda no poner puntos de final de sentencia,más que allá donde sea obligatorio. Con esto se evitarán problemas de lógica en los programas, debidos a terminaciones no deseadas de las sentencias que requieren un fin de sentencia (estas se terminarán con su END- correspondiente: END-IF, END-READ, END-EVALUATE, END-PERFORM, etc...). Por lo tanto en la PROCEDURE, sólo se pondrán puntos:

  - al final de los nombres de párrafo
  - al final de la última instrucción de cada párrafo

- Cuando se codifique una sentencia que necesite un delimitador del tipo END- ..., éste se escribirá en solitario en la última línea de la sentencia y en la misma columna que el verbo inicial de la misma.

- Es aconsejable hacer uso de la palabras reservadas Cobol, tales com SPACES, ZERO, LOW-VALUES, HIGH-VALUES, etc.

- No utilizar los símbolos de comparación, sino sus nombres correspondientes, salvo que se prevea que la impresora que vamos a utilizar reconozca tales símbolos. Es decir, poner A EQUAL B o A GREATER B en vez de A=B o A>B, respectivamente.

- Usar letras mayúsculas, tanto en el código como en los comentarios.

NPRCOB25 - Apartado 4

Nomenclatura de párrafos

- Se comenzará la codificación de la Procedure Division con un párrafo llamado "P-ddpptnnn" donde 'ddpptnnn' es el nombre del programa. Este es el párrafo raiz del programa, dentro del cual se distinguirán tres PERFORM: 'I1-...', 'P1-...' y 'F1-...', como se muestra en el siguiente ejemplo:

```
PROCEDURE DIVISION
******************************************************
*
******************************************************
P-PLMEC001

        PERFORM I1-INICIO

        PERFORM P1-PROCESO

        PERFORM F1-FIN

        STOP RUN
```

  - En el párrafo 'I1-...' se harán las acciones iniciales del programa, como abrir ficheros , inicializar contadores, recuperar variables del sistema (fecha hora,terminal,...), leer el primer registro, etc.
  - En el párrafo 'P1-...' se desarrollará el grueso del programa. Si es un PERFORM UNTIL fin-fichero, es necesario llevar leído el primer registro.
  - En 'F1-...' se harán las últimas acciones del programa: cerrar ficheros, cursores, etc.

- Para llamar a los párrafos, usar nombres claros y descriptivos de su función, haciendo uso, si se estima razonable, de los treinta caracteres que, como máximo está permitido utilizar.

- El nombre de un párrafo estará formado por:

  a) Un prefijo alfanumérico que será indicativo del lugar que ocupa el párrafo en la jerarquía de módulos del programa (o lo que es lo mismo, del camino lógico seguido hasta llegar al módulo en cuestión).

  b) Un nombre descriptivo que debe indicar, sin ambiguedad, la función del párrafo. Estará separado del prefijo por un guión.

- Las reglas de formación del prefijo del párrafo son las siguientes:

  - El prefijo comenzará con una letra, que indica la función general del párrafo:

I: Para párrafos pertenecientes al Inicio del programa.

P:Para párrafos pertenecientes al Proceso Principal del programa.

F:Para párrafos pertenecientes al Final del Programa.

A:Para párrafos que se emplean como tratamientos auxiliares y que pueden ser llamados desde cualquier punto del programa una o más veces.

- Para cada tipo de letra del prefijo (I, P, F, A) habrá una numeración independiente, a partir del 1 en ascendente.

º La misión del prefijo de los procedimientos es ofrecer información acerca de la complejidad del programa y de la jerarquía de los módulos, no de orden de ejecución de los mismos. Prefijos demasiado largos indican un número excesivo de niveles en el árbol del programa, lo cual incrementa notablemente la complejidad de la lógica y el mantenimiento del mismo.
Como regla general, en un programa de complejidad baja o moderada (objetivo de todo buen diseño) el prefijo no debería tener una longitud mayor de 5 o 6 dígitos.

º Antes de hacer la llamada a los párrafos principales, se documentará, como información,el nombre de cada uno de los párrafos a los cuales llama, acompañado de una pequeña descripcion de lo que hace cada uno. Ejemplo:

```
************************************************************
* P1-SUCURSALES
* P11-GUARDAR LA CLAVE INFERIOR ENTRE MAESTRO Y MOVIMIENTOS
* P12-TRATAMIENTO DE LAS CLAVES DE SUCURSALES Y CUENTAS
* P13-GRABACION DEL MAESTRO RESULTANTE
************************************************************
P1-SUCURSALES .

   PERFORM P11-TRATAR-SUCURSALES
   PERFORM P12-TRATAR-CLAVES UNTIL .....
   PERFORM P13-GRABAR-TOTAL-SUCURSAL
```

º No utilizar el parámetro SECTION (Nombre-Párrafo SECTION). Así se evitará que una sentencia PERFORM pueda ejecutar varios párrafos.

NPRCOB25 - Apartado 5

Estructura y jerarquía de párrafos

- Es muy aconsejable que el tamaño de los párrafos no sea muy extenso. Para evitarlo, debe tenderse a una mayor modularización del código, procurando que cada párrafo tenga un sentido lógico propio.

- Los párrafos de la Procedure Division tendrán una estructura jerarquizada en forma de árbol, con una raíz única.

- Un párrafo debe tener un único punto de entrada y un único punto de salida. Análogamente, el programa acabará en un punto único.

- No se utilizarán los finales de párrafo ni la sentencia EXIT. Los párrafos acabarán en su última instrucción, terminada en punto. Para evitar errores en la compilación, se dejará una línea al final de cada párrafo que contendrá un punto.

#

NPRCOB25 - Apartado 6

Utilización de la sentencia PERFORM

- Un módulo o párrafo SOLO puede ser ejecutado mediante invocación, utilizando la sentencia PERFORM.

- Un módulo puede hacer PERFORM a sus descendientes, pero NUNCA a sus ascendientes ni a sí mismo.

- Dado que no se usan los finales de párrafo, ni la sentencia EXIT, no se utilizará la claúsula THRU del PERFORM. De aquí se deduce que un PERFORM puede invocar a un, y sólo un, párrafo; si bien el mismo párrafo puede ser llamado desde varios puntos del programa. En este caso se aplicaría un tratamiento de “Tratamientos Auxiliares”.

DADN88

Bucles

- Para la ejecución de bucles iterativos se utilizará la opción UNTIL del PERFORM. En este caso se codificará, antes del PERFORM, las instrucciones de inicialización de condiciones y variables que sean procedentes.

- Finalizar los bucles con un operador relacional distinto de IGUAL, para evitar la posibilidad de entrar en un bucle sin fin. Ejemplo:

```
PERFORM Pxxx-PARRAFO
        UNTIL CON-BUCLE > CTN-99
```

en vez de:

```
PERFORM Pxxx-PARRAFO
        UNTIL CON-BUCLE = CTN-100
```

- Uso del PERFORM ... UNTIL 'in-line' o 'out-line':
- En el perform 'in-line', las sentencias a iterar están incluidas dentro de la propia sentencia perform, y no en un párrafo aparte. Se utilizará sólo en el caso de que sea pequeño el número de sentencias,no se le invoque desde otro punto y se encuentre en el último nivel de la estructura (por ejemplo:para inicializar tablas, cargar las mismas, si acaso no se puede hacer con MOVES, en el caso de búsqueda se utilizará mejor el SEARCH). En este caso no se permiten los anidamientos de PERFORM's. Por ejemplo:

```
PERFORM VARYING CON-BUCLE FROM 1 BY 1
        UNTIL CON-BUCLE  > CTN-100
              MOVE ...
                    COMPUTE ...
        MOVE ...

END-PERFORM
```

En este caso no se permiten los anidamientos de PERFORM's.

La ventaja del perform 'inline' es que el compilador no tiene que generar código de llamada a rutina, salto y retorno implícito, lo cual es más eficiente.
Tiene la desventaja de que, al introducir bucles dentro de párrafos, se puede complicar la lógica o estructuración del programa, **por lo cual se hace desaconsejable.**
- En el perform 'out-line', las sentencias están agrupadas en un párrafo aparte, lo que favorece la estructuración del programa según la metodología Warnier, y la claridad del código. Ejemplo:

```
PERFORM P11-TRATAR-EMPRESA
        VARYING CON-BUCLE FROM 1 BY 1
              UNTIL     CON-BUCLE  > CTN-100
.................
************************
P11-TRATAR-EMPRESA
************************

MOVE ...
  COMPUTE ...
  MOVE ...
```

Otra ventaja de perform ‘out-line’ es que el grupo de sentencias al que invoca puede ser ejecutado desde distintos sitios dentro del programa, lo cual no es posible con el perform ‘in-line’.

Se prohibe el uso de TEST AFTER en un bucle. El TEST BEFORE es la opción por defecto.

Utilización de sentencias condicionales

º Evitar en lo posible el exceso de anidamientos de IF, ya que dificulta enormemente la compresión de la lógica del programa. Sustituirlos siempre que sea posible por la sentencia EVALUATE / END-EVALUATE; esta es una instrucción que aclara bastante la lógica y contribuye a estructurar la programación, facilitando además añadir nuevas condiciones. (Aunque el coste de ejecución es algo mayor para el EVALUATE, la claridad en el programa compensa dicho coste).

Ejemplo:

```
IF NOMBRE-CAMPO-1  = CAMPO-A
        (acción A)
ELSE
    IF NOMBRE-CAMPO-1 = CAMPO-B
        (acción B)
    ELSE
        IF NOMBRE-CAMPO-1 = CAMPO-C
            (acción C)
        ELSE
            (acción D)
        END-IF
    END-IF
END-IF
```

La siguiente estructura representa la misma secuencia lógica:

```
EVALUATE NOMBRE-CAMPO-1
        WHEN CAMPO-A
                (acción A)
        WHEN CAMPO-B
                                (acción B)
        WHEN CAMPO-C
                (acción C)
        WHEN OTHER
                (acción D)
END-EVALUATE
```

º La sentencia EVALUATE permite contemplar un amplio abanico de condiciones gracias a los parámetro ALSO, TRUE, FALSE, ANY, THRU ... Ejemplo:

```
EVALUATE TRUE ALSO FALSE
        WHEN condición-1 ALSO expresión-aritmética-1
                (acción Y)
                ..../...
        WHEN OTHER
                (acción Z)
END-EVALUATE
```

º Mientras sea posible, utilizar expresiones positivas en vez de negativas. Ejemplo:

- Expresión positiva a usar:

```
IF NOMBRE-CAMPO-1   =  NOMBRE-CAMPO-2 OR
   NOMBRE-CAMPO-1   =  NOMBRE-CAMPO-3
```

- Expresión negativa a evitar:

```
IF NOMBRE-CAMPO-1  NOT =  NOMBRE-CAMPO-2 AND
   NOMBRE-CAMPO-1  NOT =  NOMBRE-CAMPO-3
```

- Las condiciones complejas, con AND's y OR's , se evitarán en lo posible. De no poderse eludir, se estructurarán y agruparán con paréntesis de modo que resulten fácilmente legibles.

- Cada IF se terminará con una instrucción END-IF.

- Evitar el uso de ELSE NEXT SENTENCE. No codificar ELSE para ramificaciones por defecto. Esto hará más fácil de comprender el código.

- Si dentro de un IF hay multitud de lineas, es aconsejable poner PERFORM en su lugar y desarrollarlas en párrafos aparte. (Lo mismo para un WHEN de un EVALUATE).

- No utilizar la claúsula THEN de la sentencia IF, ya que no es obligatoria ni contribuye a una mejor compresión de la lógica.

- Cuando se utilicen IF anidados, endentarlos hacia la derecha para facilitar su lectura. Las instrucciones IF, ELSE, END-IF correspondientes a un mismo IF, irán en la misma columna, y las instrucciones que contenga ese IF irán endentadas cuatro columnas a la derecha. Las claúsulas ELSE y END-IF se pondrán en una línea separada. Ejemplo:

```
IF condición-1
   instrucción-11
   IF condición-2
      instrucción-21
      IF  condición-3
          instrucción-31
      ELSE
             instrucción-32
      END-IF
    ELSE
        instrucción-22
    END-IF
END-IF
```

- Para un grupo de sentencias condicionales conectadas por AND's, se pondrá primero aquella que más probabilidad tenga de ser cierta, luego la segunda menos probable, y así hasta la más probable.

- Para un grupo de sentencias condicionales conectadas por OR's, se pondrá primero aquella que más probabilidad tenga de ser cierta, luego la segunda más probable, y así hasta la menos probable.

- En el EVALUATE las condiciones son evaluadas en el orden en que hansido codificadas. Por tanto, es conveniente colocarlas en orden decreciente de probabilidad de satisfacción (es dfecir, la más probable en el primer WHEN, y así sucesivamente), excepto para el WHEN OTHER, que debe ser siempre la última condición.

- Las sentencias aritméticas se sacarán de las sentencias condicionales. Las operaciones aritméticas dentro de una sentencia condicional sólo son realizadas con una precisión máxima de seis dígitos decimales.

- Para mejorar el rendimiento, definir con igual PICTURE los campos que se comparen en sentencias condicionales.

  Para evitar problemas de incompatibilidad de campos en comparación, utilice, siempre que sea posible, las variables alfanuméricas. Tener en cuenta ciertas limitaciones que impone el terminal inteligente, como comparar ARQ-OPERACION con constantes, por ejemplo no se admite IF ARQ-OPERACION = CA-A (siendo CA-A una constante alfanumérica con valor 'A'), en este caso debe hacerse IF ARQ-OPERACION = 'A'. Para más información, ver apartado **NORMAS**, en "*Normas de Programación con Arquitectura On-line" NPRAOL02 "Normas generales de Programación".*

NPRCOB25 - Apartado 8

Uso de variables, constantes, niveles 88 y campos de control

- Se deben inicializar todas las variables de la WORKING-STORAGE, que no tengan claúsula VALUE, al inicio del programa con el verbo INITIALIZE.

- De forma general, se prohiben los MOVE de valores constantes directos; es preferible hacer MOVE de variables definidas en la WORKING, que contengan los valores especificados. Por ejemplo, en vez de codificar “MOVE ‘SE HA PRODUCIDO UN ERROR DB2’ TO MSG-SALIDA” , se hará “MOVE CTA_ERROR-DB2 TO MSG-SALIDA”. Esto facilita el mantenimiento de los programas, pues, de tener que cambiar algún literal, sólo se hará una vez y en la WORKING.

- Para activar una de las condiciones de un nivel 88 es obligatorio utilizar el mandato SET, evitando hacer directamente un MOVE sobre el campo. Por ejemplo, será preferible hacer “SET SI-ENCONTRADO TO TRUE” que “MOVE ‘S’ TO CC-ENCONTRADO”.

- En las sentencias MOVE se recomienda que el campo emisor sea de igual naturaleza y longitud que el campo receptor. Si el campo a mover tiene una longitud más corta que el campo receptor, el compilador genera un MOVE muy eficiente para los datos, seguido por un ineficiente MOVE de espacios a las posiciones restantes, utilizando DOS sentencias MOVE en lugar de UNA, con el consumo de recursos que conlleva.

NPRCOB25 - Apartado 9

Tablas Cobol

- Se recomienda utilizar los campos índices creados en la definición de la tabla (claúsula INDEXED BY), en lugar de campos de subscripción (es decir, variables numéricas para acceder a las tablas), pues es lo más eficiente para trabajar con tablas.

- Si se utilizan campos de subscripción, para incrementar la eficiencia deben ser definidos como campos binarios (COMP) de menos de 5 dígitos.

- Es conveniente emplear una constante para probar el final de una tabla. Ejemplo:

```
01 CC-FIN-TABLA       PIC(X)    VALUE 'N'.
   88    FIN-TABLA                VALUE 'S'.
.....//.....
05 CN-MAX-TABLA PIC 9(2) comp value 30.
.....//.....

01  TB-TABLA.
        05  TB-TABLA-ITEM OCCURS 30 TIMES
                INDEXED BY IN-TABLA
                S99 COMP.
        ...//...
```

CASO 1

```
        PERFORM P221-TRATAR-TABLA
                VARYING IN-TABLA FROM 1 BY 1
                 UNTIL IN-TABLA  GREATER  CN-MAX-TABLA.
```

CASO 2

```
....//....
01   CN-SI              PIC X   VALUE 'S'.

....//....
PERFORM P221-TRATAR-TABLA
            UNTIL FIN-TABLA.

....//....
P221-TRATAR-TABLA.
        MOVE  { instrucciones de tratamiento del elemento}
         ADD 1 TO IN-TABLA
         IF IN-TABLA GREATER CTN-MAX-TABLA
         MOVE CN-SI TO CC-FIN-TABLA.
```

De esta forma, cuando cambia el tamaño de la tabla, sólo debemos cambiar la claúsula OCCURS y el valor de CN-MAX-TABLA, sin cambiar en todos los lugares del programa en donde se comprueba que el índice no sobrepase el valor máximo.

- En búsquedas secuenciales, es óptimo preguntar por el "último elemento" cargado para no recorrer toda la tabla si no está totalmente llena. Ejemplo:

```
        PERFORM Pxxx-CARGAR-TABLA
```

```
                    UNTIL FIN-CARGA
...//...
Pxxx-CARGAR-TABLA.
        {tratamiento de carga del elemento}
        ADD 1 TO IN-TABLA
        IF IN-TABLA GRATER CTN-MAX-TABLA
           MOVE CC-SI TO CC-FIN-TABLA
            SUBTRACT 1 FROM IN-TABLA GIVING VN-ULTIMO-ELEMENTO.
....//....
PERFORM Pyyy-BUSCAR-ELEMENTO
            UNTIL  CC-ENCONTRADO OR
             IN-TABLA GREATER VN-ULTIMO-ELEMENTO.
....//....
Pyyy-BUSCAR-ELEMENTO.
    IF {campo busqueda} = TB-TABLA-ITEM (IN-TABLA)
        MOVE CC-SI TO CC-ENCONTRADO.
     ELSE
        ADD 1 TO IN-TABLA
        IF IN-TABLA GREATER CTN-MAX-TABLA
            MOVE CN-SI TO CC-FIN-TABLA.
        END-IF
    END-IF

...//...
```

º Cuando los datos de la tabla son muy numerosos y están ordenados, es más eficiente utilizar la búsqueda indexada que la secuencial. Ejemplo:

```
01  CC-DESBORD-CURSOS              PIC X(2)      VALUE 'NO'.
    88 DESBORD-CURSOS                           VALUE 'SI'.
01  CC-CURSO-ENCONTRADO            PIC X(2)      VALUE 'NO'.
   88 CURSO-ENCONTRADO                          VALUE 'SI'.

01  CN-MAX-CURSOS                  PIC 9(3)      VALUE 100.
....//....
01  TB-CURSOS.
    05 TB-CURSOS-REG    OCCURS        100 TIMES
        ASCENDING KEY IS TB-CURSOS-CURSO
        INDEXED BY TB-CURSOS-IND.
        10    TB-CURSOS-CURSO                          PIC X(06).
              10    TB-CURSOS-MARCA                          PIC X(01).
....//....
INITIALIZE TB-CURSOS
*---    DESCARGAR EL FICHERO DE CURSOS EN LA TABLA
PERFORM  P11-TABLA-CURSOS
    THRU P11-TABLA-CURSOS-EXIT
....//....
 P11-TABLA-CURSOS.
    MOVE CA-NO TO NO-FIN-FICURSOS
    MOVE CA-NO TO CC-DESBORD-CURSOS
    MOVE ZEROS TO IN-CURSOS
*----   LEE REGISTRO FICHERO CURSOS
    PERFORM A11-LEER-CURSOS
        THRU A11-LEER-CURSOS-EXIT

*----   CARGA LA TABLA
     PERFORM P111-CARGA-CURSO
         THRU P111-CARGA-CURSO-EXIT
       UNTIL FIN-FICURSOS.
P11-TABLA-CURSOS-EXIT.
```

```
       EXIT.
*****************************************************************
P111-CARGA-CURSO.
*****************************************************************
          ADD CN-1                 TO IN-CURSOS
          MOVE REG-FICURSOS   TO TB-CURSOS-CURSO(IN-CURSOS)
*----     LEE REGISTRO FICHERO CURSOS
          PERFORM A11_LEER-CURSOS
               THRU A11-LEER-CURSOS-EXIT
*----     COMPRUEBA EL DESBORDAMIENTO
          IF IN-CURSOS  = CN-MAX-CURSOS
             IF NOT FIN-FICURSOS
                INITIALIZE TB-CURSOS
                SET DESBORD-CURSOS TO TRUE
                SET FIN-FICURSOS        TO TRUE
             END-IF
          END-IF.

..../////....
*
       SEARCH-CURSO.
*
             SET TB-CURSOS-IND   TO 1.
*
             SEARCH TB-CURSOS
             WHEN TB-CURSOS-CURSO (TB-CURSOS-IND)  = REG-ENCURSOS (IN-
CURSOS)

              SET CURSO-ENCONTRADO  TO  TRUE
              MOVE CA-S                    TO  TB-CURSOS-MARCA (TB-CURSOS-IND)
              END - SEARCH.
```

NPRCOB25 - Apartado 10

Integridad referencial, cardinalidad

El diseño de los datos en el desarrollo de una aplicación informática pasa por tres estados:

- El modelo Conceptual de datos.
- El modelo Lógico de datos.
- El modelo Físico de datos.

Conforme vamos pasando por cada uno de ellos, los datos del sistema se van refinando, hasta alcanzar en el último modelo las características con las que los datos se van a implantar.

Para la construcción de los modelos de datos se han de identificar una serie de conceptos, tales como:

- Entidades.
- Relaciones
- Atributos

En las entidades habrá que :

- Identificar **Entidades Padre**. Para cada una de las entidades identificadas habrá que preguntarse si depende de ella otra entidad. Para una posible Entidad Padre se verificará:
  - que los padres son conocidos cuando el hijo se crea
  - que permanecen constantes a lo largo de la vida de las entidades dependientes o hijas.
- Identificar **Entidades Hijos o Dependientes**. Para cada una de las entidades identificadas, habrá que preguntarse si depende de alguna entidad.

En las relaciones se deberá identificar el grado de asociación, indicando la cardinalidad de las mismas. **La Cardinalidad** nos indica cuantas ocurrencias de una entidad se deben relacionar con una ocurrencia de otra entidad.
Esta definición nos muestra que una entidad *dependiente o hija* está asociada con una y sólo una ocurrencia de una *entidad padre* y que cada *entidad* padre puede tener cero, uno o varios hijos.

Otro concepto es la **Clave**, se define como el atributo o conjunto de atributos concatenados pertenecientes al mismo tipo de entidad que hacen único el acceso a una entidad u ocurrenciade la tabla, es decir, que determinan de forma única una entidad.
Una vez seleccionada la clave, podremos ver si está compuesta por un único atributo: **Clave Simple**, o por un conjunto de atributos, **Clave Múltiple** o **concatenada**.

Otro concepto importante es el de **Clave Ajena**, esto es, “Atributo de una tabla que es clave en otra”.

Será muy importante su localización para evitar inconsistencia de la información contenida en las estructuras de datos: **Integridad Referencial**.
“Si existe CLAVE AJENA, el valor ha de ser igual al atributo clave o bien ser nulo”.
Si se da de baja, habrá que buscarlo en cualquier otro sitio que pueda aparecer y si no se da de baja al registro completo, al menos se pondrá el valor nuloa ese atributo.

Para el mantenimiento de la integridad de los datos, las bases de datos necesitan ciertas reglas para la realización de inserciones y borrados, como:

- No se pueden insertar hijos sin padres

- No se puede borrar un padre sin borrar todos sus hijos
- Borrar todas las ocurrencias de un hijo no requiere la automática eliminación del padre (dependerá de la relación establecida).

Para más información sobre este tema hay que acudir a **NORMA**S apartado "*Normas Generales DB2*" en "*Consideraciones de Diseño de Bases de Datos*" y "*Recomendaciones para Programas de Aplicación*".

Como norma, deberemos asegurarnos al codificar un programa con varias tablas relacionadas, que tenemos clara la cardinalidad y relación entre ellas o deberemos consultar cualquier duda de este tipo con el analista responsable.

NORMAS DE PROGRAMACIÓN EN COBOL

CODIGO	TEMA
NPRCOB30	Recomendaciones para programacion

Se consideran los siguientes apartados:

Apartado 1 - Normas para programación ON LINE

Apartado 2 - Normas para programación BATCH

NPRCOB30 - Apartado 1

Normas para la programación ON LINE

Los programas On Line deben incorporar mediante includes las siguientes **copys**

XXCCPTP	copy con los datos que el Módulo Director da al programa; estos son: datos de la sesión en la que se encuentra, código de mensaje final, terminación con errores...
XXABEND	datos de abend almacenados por distintos programas.
XXLOGICN	estructura de programa con nombres de párrafos definidos

Reposicionamiento estándar de cursores sobre la tabla DB2. La forma de reposicionarse en una tabla DB2 tras efectuar un commit o por nueva ejecución de programa en la instalación, se realiza a través del **reposicionamiento con cursores**.

Para programas on line de lista bajo Arquitectura On line, sae debe usar la utilidad **TSO LISTADOR**. Asegúrese de utilizar siempre esta opción para nuevos programas ya que reutilizar copias de otros puede provocar que no tome npvedades de Arquitectura.

El **tratamiento con cursores** más eficiente con respecto al acceso a los datos es declarar tantoscursores como columnas del índice (por el cual se está reposicionando) no sean condicionadas por el operador "=".

Ejemplo: TABLA1 tiene la clave EMPRESA/CENTRO

a) Para obtener todos los centros de una empresa, el cursor será:

```
DECLARE CSR-CEMPRESA-01 ...
        WHERE
                EMPRESA = EMPRESA
                CENTRO   = CENTRO
```

Para la lista se inicializa el centro al menor de sus valores (normalmente a blancos). Para reposicionamiento, se guarda el último centro.

b) Para obtener todos los centros de todas las empresas, los cursores serán:

```
DECLARE CSR-CEMPRESA-01 ...
        WHERE
                EMPRESA > EMPRESA
DECLARE CSR-CEMPRESA-02 ...
        WHERE
                EMPRESA = EMPRESA
                CENTRO   > CENTRO
```

Para la lista CSR-CEMPRESA-01 con empresa inicializada al menor de sus valores. Para reposicionamiento, se guarda la última empresa en CSR-CEMPRESA-01 y el último centro en CSR-CEMPRESA-02.

Con este sistema el programa puede hacerse más complejo, pero es el método más eficiente ya que usa el máximo de columnas de índice en todos los cursores.

Se abrirán los cursores desde el más restrictivo hasta el más amplio, excepto en la primera ejecución de programa donde lógicamente deberá abrirse el cursor menos restrictivo ya que no se conocen todos los datos.

Esto es posible en programas on line, ya que la Arquitectura permite distinguir la primera ejecución de un programa de las siguientes (PF8).

De esta forma se consigue el acceso a los datos por un mayor número de columnas de índice, sin embargo si se declarase un sólo cursor utilizando AND u OR sólo usaría la primera columna de índice.

Este tratamiento se aplica para scroll en programas online.

- Ver norma NPRAOL02 “Normas Generales de Programación” dentro de NORMAS DE PROGRAMACION CON ARQUITECTURA ON-LINE”.

NPRCOB30 - Apartado 2

Normas para la programación BATCH

Rutinas

Llamar a las rutinas de forma dinámica, para evitar la linkeditación de los programas por modificación de las rutinas.

Llamar a las rutinas sin comillas. Ver apartado de RUTINAS GENERALES.

Reposicionamiento

En grandes procesos batch con gran número de actualizaciones surgen dos situaciones problemáticas:

- Bloqueo innecesario de tablas modificadas que impide la concurrencia de otros procesos batch u online.

- Recuperación muy costosa en caso de ABEND o de rearranque de todo el sistema.

Para evitarlas, la aplicación debe estar preparada para realizar commits cada cierto número de actualizaciones.

Surge el problema del tratamiento de objetos DB2, tales como ficheros de salida. Para resolverlo se ha modificado una rutina que mediante la alocación dinámica de ficheros sincroniza la actualización de ficheros secuenciales de salida con la de objetos DB2.

El manual específico Arranques y Reposicionamientos (TB01) contiene información detallada sobre este aspecto.

Tratamiento de errores

Llamar a la rutina general XXCANCEL, según se indica en el apartado RUTINAS GENERALES.