

Contenido

COBOL	3
Estructura del programa fuente.....	3
Componentes de un programa	3
Disposición de las áreas de un programa fuente.	6
Líneas de propósito especial.....	8
Tipos de opciones de control de compilador.	9
Elementos de lenguaje.....	10
Conjunto de caracteres.	10
Separadores.....	10
Cadena de caracteres.	11
Tipos de palabras.	12
Literales.....	13
Conceptos de datos.....	15
Organización de datos con números de nivel.	17
Conceptos de la DATA DIVISION.....	21
Conceptos de la PROCEDURE DIVISION.....	25
LIBRARIES.....	32
Creando una LIBRARY.	32
Reglas para parámetros.....	33
Salir de una Library.	33
Asegurando una Library.....	33
Invocar a una Library.....	34
Declaración CALL para Libraries.....	34
Instrucción CANCEL para Libraries.....	37
CANDE.....	41
Transmisión de comandos a CANDE.....	41
Identificación de usuario e inicio de sesión.....	41
Comandos CANDE.	44
Comandos de control CANDE.....	46
WFL	47
Capacidades del WFL.....	47

Administración de Task.....	47
Administración del flujo de trabajo.....	48
Administración de archivos.....	48
Compilando programas.....	48
Reiniciando Tasks.....	49
Capacidades de soporte.....	49
Limitaciones del WFL.....	49
Creando un WFL job en CANDE.....	49
Iniciando WFL Jobs en CANDE.....	50
Opciones de la instrucción START.....	50
Estructura de un Job.....	52
Declaraciones.....	54
Ejecutando Tasks.....	55
Prevención de reinicios accidentales de tareas.....	57
Ejecutar varias tareas al mismo tiempo.....	57
Administración del flujo de control.....	58
Flujo de Control.....	60
Comunicación con el operador.....	68
DMSII.....	69
Descripción general de DASDL.....	69
Archivos TAILORED de la base de datos.....	70
Descripción general de los componentes de Enterprise Database Server.....	71
Introducción de los componentes del lenguaje DASDL.....	73
Ejemplos de definición de una base de datos.....	75
Instrucciones de control del compilador.....	78
Manejo de la base de datos con INQUIRY.....	78
Manejo de la base de datos con un programa COBOL74.....	81

COBOL

Estructura del programa fuente

Componentes de un programa

Divisiones

Se compone de 4 divisiones:

```
IDENTIFICATION DIVISION.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
PROCEDURE DIVISION.
```

IDENTIFICATION DIVISION.

Primera división y la más sencilla, solo es requerida para identificar datos del programa, no tiene efecto alguno en la ejecución del programa. Los párrafos se codifican en la columna A y los valores en columna 35.

PROGRAM-ID.	→ nombre del programa
AUTHOR.	→ autor del programa
INSTALLATION.	→ instalación donde se ejecuta
DATE-WRITTEN.	→ fecha en que fue escrito
DATE-COMPILED.	→ fecha de compilación
SECURITY.	→ confidencialidad

Solo el párrafo PROGRAM-ID es requerido, los demás son opcionales.

Se debe recalcar la puntuación al final de cada párrafo y al final de cada valor del párrafo, todo va en mayúsculas.

ENVIRONMENT DIVISION.

Segunda división de cobol, básicamente nos pide información del hardware y archivos a utilizar, tiene dos secciones:

CONFIGURATION SECTION.

```
-SOURCE-COMPUTER.  
-OBJECT-COMPUTER.  
-SPECIAL-NAMES.
```

INPUT-OUTPUT SECTION

```
-FILE-CONTROL  
  SELECT <NAME-FILE> ASSIGN TO DISK/PRINTER
```

DATA DIVISION.

La DATA DIVISION esta subdividida en siete secciones:

1. FILE SECTION.

Definimos cuales son las características que tendrán todos los archivos que tendrá nuestro programa.

2. DATA-BASE SECTION.

Describe las bases de datos que serán usadas en el programa.

3. WORKING-STORAGE SECTION.

Describe los registros y variables que después se usan en la lógica del programa.

4. LINKAGE SECTION.

Usada en las libraries, aparecen en el programa llamado y describe las variables que serán referenciadas por el programa llamador.

Definición de campos similar a la que se hace en la WORKING-STORAGE y no se permite usar la cláusula VALUE.

5. COMMUNICATION SECTION.

Describe las variables que servirán como interfaz entre la library DCI (*data communications interface*) y el programa.

6. REPORT SECTION.

Describe los contenidos y formatos de los archivos generados.

7. LOCAL-STORAGE SECTION.

Extensión de COBOL74, describe los parámetros que se recibirán desde otras *task* o desde *procedures* vinculados desde otro programa.

PROCEDURE DIVISION.

Le indica a la computadora los pasos que debe realizar para resolver el requerimiento abordado por el programa fuente. Esta división usa los datos descritos en la DATA DIVISION.

- Consta de párrafos (rutinas), instrucciones (verbos) y secciones.
- Las rutinas y secciones las nombra el programador y se codifican en el margen A (columna 8)
- Las instrucciones se codifican en el margen B (columna 12).

Tipos de instrucciones:

- Imperativas: ADD, DIVIDE, COMPUTE, ACCEPT, MOVE, OPEN, PERFORM
- Condicionales: IF, CASE.

Secciones.

La ENVIRONMENT DATA y PROCEDURE DIVISION pueden estar subdivididos en secciones. Una sección identifica aún más el propósito de una división. En la ENVIRONMENT y DATA DIVISION, COBOL74 especifica los nombres de las secciones, pero en la PROCEDURE DIVISION nosotros especificamos los nombres.

DIVISION	SECTION	PROPÓSITO DE LA SECCIÓN
IDENTIFICATION	N/A	N/A
ENVIRONMENT	CONFIGURATION	Especifica equipo de cómputo.
	INPUT-OUTPUT	Asocia archivos con dispositivos específicos.
DATA	FILE	Describe la estructura de los registros de los archivos.
	DATA-BASE	Describe una o más bases de datos que pueden ser usadas por el programa.
	WORKING-STORAGE	Describe elementos de datos intermedios.
	LOCAL-STORAGE	Describe los datos que se van a pasar o recibir como parámetros de un procedimiento externo.
	LINKAGE	Describe elementos de datos a los que se hará referencia por el programa llamador y el programa llamado.
	COMMUNICATION	Describe elementos de datos en el programa fuente que sirve como la interfaz entre la library de interfaz de comunicaciones de datos (DCI) y el programa.
	REPORT	Describe el contenido y formato del reporte generado por el Report Writer
PROCEDURE	Definido por el usuario.	Agrupar los párrafos en secciones que define el usuario.

Párrafos.

En la IDENTIFICATION y ENVIRONMENT DIVISION, un párrafo inicia con un encabezado que lo identifica. Un encabezado de párrafo consiste en una palabra reservada seguida por un punto.

En la PROCEDURE DIVISION, un párrafo inicia con una palabra definida por el usuario llamado nombre del párrafo. Este es seguido por un punto y opcionalmente una o más entradas.

Un párrafo termina inmediatamente antes del próximo encabezado de párrafo o nombre de sección, y al final de la división, o en las palabras llaves END DECLARATIVES en las DECLARATIVES SECTION de la PROCEDURE DIVISION.

Sentencias.

Una sentencia consiste en una o más instrucciones. Debes finalizar una sentencia con un punto. Una sentencia puede ser uno de los siguientes tres tipos:

- Una sentencia que le indica al compilador que tome acciones específicas durante el proceso de compilación.

- Una sentencia condicional que evalúe un valor verdadero y especifique una acción dependiendo del resultado de la evaluación.
- Una sentencia imperativa que especifica una acción incondicional a ser tomada.

Instrucciones (Statements).

Una instrucción es una combinación sintácticamente válida de palabras y símbolos que inicia con un verbo de COBOL74. Una instrucción termina cuando el compilador detecta un nuevo verbo o un punto.

Disposición de las áreas de un programa fuente.

El compilador espera que los componentes de tu programa fuente aparezcan en áreas específicas a lo largo de la línea o del código. Cada línea tiene 80 posiciones, las cuales son agrupadas en 5 áreas que constituyen el formato de línea.

COLUMNAS	NOMBRE	ENTRADAS ACEPTABLES
1-6	Área de secuencia.	Numero de secuencia
7	Indicador de área.	*, / , D , - , \$
8-11	Área A	<ul style="list-style-type: none"> • Encabezado de las divisiones • Encabezado de las secciones • Encabezados de párrafos • Descripción de archivos (FD). • Descripción de sort (SD). • Numero de niveles. • Palabra clave DECLARATIVE. • Palabra clave END DECLARATIVE.
12-72	Área B	<ul style="list-style-type: none"> • Sentences • Números de niveles que no sean 01 o 77
73-80	Identificación de área.	Comentarios, Mark ID.

Área de secuencia.

Columna de la 1 – 6.

Destinada para la secuencia numérica y sirve para identificar una línea del programa fuente.

Si agrega información no numérica en el campo de secuencia, es posible que no pueda usar ciertas funciones de Unisys dentro del compilador y que limite su capacidad para usar los archivos fuente con ciertos programas de utilidades de Unisys.

Área de indicadores

Columna 7.

El indicador de área puede ir vacío o contener uno de los cinco caracteres que indican un propósito especial para la línea.

- Un asterisco (*) indica que la línea es un comentario.

- Una diagonal (/) indica que es una línea de comentario y que al imprimirse haga un salto de página y el comentario aparezca en la parte superior de la página nueva.
- La letra D indica que la línea es una línea de debugging (depuración).
- Un guion medio (-) indica que la línea es una continuación de línea.
- El símbolo de pesos (\$) indica que la línea es un registro de control del compilador.
- Un espacio () indica que la línea es una línea normal de Cobol74.

Área A.

Columna de la 8 – 11.

Divisiones, secciones y encabezados de párrafos, indicadores de nivel, los niveles 01 y 77, y palabras clave para las DECLARATIVES deben iniciar en el área A. Puedes iniciar tu entrada en cualquier lugar dentro del área A. Es aceptable que tu entrada se extienda dentro del área B.

Las reglas para formar los encabezados de una división y sección son las siguientes:

- Ingresa el nombre de la división o sección en el área A.
- Seguido del nombre de la división o sección debe haber por lo menos un espacio.
- Coloca DIVISION o SECTION, la que sea apropiada.
- Termina la entrada con un punto.

Las reglas para formar un encabezado de párrafo son las siguientes:

- Ingresa el nombre del párrafo.
- Seguido del nombre coloca un punto.

Las reglas para formar las declarativas son las siguientes:

- Ingresa la palabra DECLARATIVES. en una sola línea al comienzo de la PROCEDURE DIVISION.
- Ingresa END DECLARATIVES. En una línea sola para finalizar las declarativas.

Área B.

Columna de la 12 – 72.

La mayor parte de tu código, incluyendo sentencias, instrucciones, cláusulas, y varios números de niveles, comienzan en el área B.

En la DATA DIVISION, todos los números de niveles excepto 01 y 77, pueden comenzar en el área A o B.

La primera sentencia o entrada de un párrafo comienza en la misma línea que el nombre de párrafo o en el área B de la siguiente línea que no esté en blanco y que no sea una línea de comentario.

Sentencias adicionales comienzan después de la sentencia anterior en el área B de la misma línea o en el área B de la siguiente línea en blanco que no es una línea de comentario.

Área de identificación.

Esta área es un área opcional. Puedes usarlo para propósitos de documentación.

Líneas de propósito especial.

Las líneas de propósito especial son aquellas que tienen un carácter o espacio en blanco en el área indicadora (columna 7).

- Líneas de comentarios.

Una línea de comentario es cualquier línea con un asterisco (*) o una barra (/) en el área del indicador (columna 7) de la línea. Una línea de comentario puede aparecer en cualquier lugar de un programa fuente y puede utilizar cualquier combinación de caracteres del conjunto de caracteres de la computadora, no solo los caracteres de COBOL74.

Si utiliza el carácter de barra diagonal para indicar un comentario, la impresora expulsa una página antes imprimiendo el comentario en la lista de salida del compilador.

- Líneas de continuación.

Una línea de continuación es una línea que continúa desde una línea anterior. Se indica una línea de continuación con un guion (-) en el área del indicador (columna 7). El guion significa que el primer carácter que no esté en blanco en el área B (columnas 12 a 72) de la línea de continuación sigue al último carácter que no está en blanco de la línea anterior, sin espacios insertados. Puedes utilizar líneas de continuación sucesivas. Área A (columnas 8 a 11) de una línea de continuación debe estar en blanco.

Seq1....2....3....4....5....6....7..
000100	IDENTIFICATION DIVISION.
000200	ENVIRONMENT DIVISION.
000300	DATA DIVISION.
000400	WORKING-STORAGE SECTION.
000800	PROCEDURE DIVISION.
000900	RUTINA-PRINCIPAL.
001000	DISPLAY "SUMA DE 2 NUMEROS DFGBDFGBIGHUOF UHDFUIHDFYUG UHDFIUF
001020	- "HGYGI JGIUVT ".
001040	
001100	DIS
001200	- PLAY "DIGITA PRIMER NUMERO:".
001800	STOP RUN.

```
#RUNNING 8643
#8643 DISPLAY:SUMA DE 2 NUMEROS DFGBDFGBIGHUOF UHDFUIHDFYUG UHDFIUF HGYGI
      JGIUVT  .
#8643 DISPLAY:DIGITA PRIMER NUMERO:..
#ET=0.0 PT=0.0 IO=0.0
```


- **Líneas de DEBUGGING.**

Una línea de DEBUGGING es cualquier línea con una D en el área del indicador (columna 7) de la línea. Una línea de DEBUGGING con espacios en las columnas 8 a 72 se considera lo mismo que una línea en blanco. Puede ingresar una línea de DEBUGGING en cualquier lugar después del párrafo OBJECT-COMPUTER.

El módulo de DEBUGGING es activado cuando especificas la cláusula WITH DEBUGGING MODE en el párrafo SOURCE-COMPUTER. Si no activas el módulo de depuración, el compilador trata una línea de DEBUGGING como una línea de comentario, por lo tanto, debes asegurarte de que tu programa sea sintácticamente correcto cuando las líneas de depuración son consideradas para ser líneas de comentarios.

Se requiere usar la opción de control de compilación \$SET FREE = FALSE, para poder usar las líneas de DEBUGGING.

- **Opciones de control de compilador.**

Una opción de control del compilador es cualquier línea con un signo de pesos (\$) en el área del indicador. (columna 7) de la línea. Esta línea especifica las opciones de control del compilador que se utilizarán durante el proceso de compilación.

Tipos de opciones de control de compilador.

Una opción de control del compilador puede ser de tipo booleano, de valor o inmediato. La siguiente lista muestra las opciones pertenecientes a cada categoría.

- **Booleano.**

Una opción booleana está habilitada (establecida en TRUE) o deshabilitada (establecida en FALSE). Cuando está habilitada, la opción indica al compilador que aplique una función asociada a todos los procesamientos posteriores hasta que la opción esté deshabilitada. Las opciones booleanas también pueden tener parámetros asociados relacionados con la función afectada por la opción booleana.

- **Valor**

Una opción de valor indica al compilador que almacene un valor asociado con una función dada. Las opciones de valor son las siguientes:

- **Inmediato**

Una opción inmediata indica al compilador que aplique una función que es independiente del procesamiento subsecuente. Las opciones inmediatas también pueden tener asociadas parámetros. Las opciones inmediatas son CLEAR, PAGE, y VOID.

Elementos de lenguaje.

Esta sección describe los siguientes elementos usados para formar los componentes de un programa fuente:

- Conjunto de caracteres
- Separadores
- Cadena de caracteres

Conjunto de caracteres.

La unidad más básica e indivisible del lenguaje COBOL74 es el carácter. El conjunto de caracteres que usas para escribir programas en COBOL74 incluyen las letras de alfabeto, dígitos y caracteres especiales. El conjunto de caracteres consta de 52 caracteres, incluyendo los 5 caracteres especificados en el estándar ANSI-74 mas el signo @.

Character	Meaning	Character	Meaning
0 through 9	Digit	A through Z	Letter
	Space or blank	+	Plus sign
-	Minus sign or hyphen	*	Asterisk
/	Virgule, or slash	=	Equal sign
\$	Dollar sign	,	Comma or decimal point
;	Semicolon	.	Period or decimal point
"	Quotation mark	(Left parenthesis
)	Right parenthesis	>	Greater than sign
<	Less than sign	@	Commercial at sign

Separadores.

Un separador es una cadena de uno o mas caracteres de puntuación. Puede unir un separador con otro separador o con una cadena de caracteres. Una cadena de caracteres debe estar vinculada a un separador

Separador	Explicación
(espacio)	<p>Los espacios pueden preceder o seguir a todos los demás separadores. Reglas especiales para los espacios son los siguientes:</p> <ul style="list-style-type: none">• Se requiere un espacio antes del delimitador de pseudotexto de apertura.• Un espacio que precede a la comilla final de un literal no numérica se considera parte de la literal.• Un espacio que sigue a las comillas iniciales de un literal no numérico es considerado parte de la literal.

.	Los puntos marcan el final de una entrada COBOL74. Un punto siempre se trata como un separador cuando va seguido de un espacio.
,;	Comas y punto y coma delimitan cláusulas. Una coma o punto y coma, siempre se trata como un separador cuando va seguido de un espacio.
()	Los paréntesis izquierdo y derecho delimitan subíndices, índices, expresiones aritméticas o condiciones.
“”	Las comillas delimitan los literales no numéricos. Si el literal continúa en otra línea, debe ingresar otra comilla como primer carácter que no esté en blanco en el área B y las comillas de apertura deben ir precedidas inmediatamente por un espacio, a la izquierda paréntesis (coma o punto y coma). Se deben colocar comillas de cierre seguido inmediatamente de un espacio, coma, punto y coma, punto o paréntesis derecho.
==	Los delimitadores de pseudotexto activan el pseudotexto. Deben aparecer en pares. Un delimitador de pseudotexto de apertura debe ir precedido de un espacio. Un delimitador de pseudotexto de cierre debe ir seguido de un espacio, una coma, punto y coma o punto. Nota: Usados en la instrucción COPY.
@	Las @ delimitan las literales undigit. Un carácter @ de apertura debe ir precedido inmediatamente por un espacio, una coma, un punto y coma o un paréntesis izquierdo; un carácter @ de cierre debe ir seguido inmediatamente de un espacio, coma, punto y coma, punto o paréntesis derecho. Ejemplo: 01 W-FS-HEX PIC 9(02) COM VALUE @1C@. 01 FS REDEFINES W-FS-HEX PIC X(01).

Cadena de caracteres.

Una cadena de caracteres es un conjunto de uno o más caracteres delimitados por separadores que forman una de las siguientes:

- Palabra
- Literal
- Cadena de caracteres
- Comentarios.

El tamaño máximo de una cadena de caracteres literales no numéricos sintácticamente válida en COBOL74 tiene 160 caracteres.

Una cadena de caracteres que contiene entre 161 y 261 caracteres hace que el compilador emita errores de sintaxis. Exceder el límite de 261 caracteres podría provocar que el proceso de compilación finalice de forma anormal sin errores de sintaxis, lo que provocaría resultados impredecibles.

Tipos de palabras.

Una palabra de COBOL es una cadena de carácter que forma una de las siguientes:

- Palabras reservadas.
- Palabra clave sensible al contexto.
- Palabra clave específica de la aplicación
- Nombre del sistema
- Palabra definida por el usuario

No puedes usar una palabra reservada como palabra definida por el usuario.

Palabras reservadas.

Una palabra reservada es una palabra COBOL74 que tiene un significado específico para el compilador. Por ejemplo, MOVE es una palabra reservada que indica al compilador que realice una operación de movimiento.

Se utilizan palabras reservadas en las siguientes seis maneras:

1. Como conectores que califican datos (OF o IN), vinculan dos o más operandos en una serie, o vinculan operadores lógicos para formar condiciones (AND y OR).
2. Como constantes figurativas que asocian nombres a valores que se utilizan habitualmente en un programa fuente.
3. Como palabras clave que son verbos u otras partes requeridas de una sintaxis.
4. Como palabras opcionales que aumentan la legibilidad de su programa.
5. Como registros especiales generados por el compilador, áreas de almacenamiento de solo lectura que dan acceso a características específicas de COBOL74.
6. Como palabras de caracteres especiales que indican operaciones aritméticas o relacionales.

Palabras definidas por el usuario.

Una palabra definida por el usuario es una palabra que usted define para completar el formato de una cláusula o declaración. Las reglas para formar una palabra definida por el usuario son las siguientes:

- Crear la palabra definida por el usuario hasta de 30 caracteres.
- Seleccione cada carácter del conjunto de caracteres de la A a la Z, del 0 al 9 y el guión (-).
- No utilice el guión como primer o último carácter de una palabra.
- No utilizar una palabra reservada.
- Asegúrese de que todas las palabras definidas por el usuario, excepto los números de nivel y los números de segmento, sean únicas. Puede utilizar la calificación para garantizar que una palabra sea única. No es necesario que los números de nivel y los números de segmento sean únicos. Un número de nivel o número de segmento determinado puede ser idéntico a un nombre de párrafo o a un nombre de sección.

- Incluya al menos un carácter alfabético en todas las palabras definidas por el usuario excepto nombres de párrafo, nombres de sección, números de nivel, nombres de texto, nombres de biblioteca, nombres de familias y números de segmento.

Literales.

Una literal es una cadena de caracteres cuyo valor es el conjunto ordenado de caracteres que lo compone o una palabra reservada que hace referencia a una constante figurativa.

Cada literal es uno de los siguientes cinco tipos:

- No numérico.
- Numérico.
- Punto flotante.
- Undigit
- Kanji

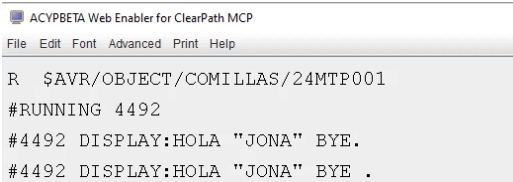
No numérico.

Un literal no numérico es una cadena de caracteres delimitada en ambos lados por comillas. Puede utilizar cualquier carácter permitido en el conjunto de caracteres de la computadora para formar un literal no numérico. Los literales no numéricos están en la categoría alfanumérica.

Las reglas para la formación de literales no numéricos son las siguientes:

- Los literales no numéricos pueden tener entre 1 y 160 caracteres de longitud.
- Una comilla simple está delimitada por dos comillas contiguas dentro de un literal no numérico. Cada par de comillas contiguas incrustadas representa un carácter de comilla simple.

```
PROCEDURE DIVISION.
MAIN-PROCEDURE.
    DISPLAY "HOLA ", QUOTES "JONA" QUOTES, " BYE".
    DISPLAY "HOLA ""JONA"" BYE ".
STOP RUN.
```



```
R $AVR/OBJECT/COMILLAS/24MTP001
#RUNNING 4492
#4492 DISPLAY:HOLA "JONA" BYE.
#4492 DISPLAY:HOLA ""JONA"" BYE .
```

- Las comillas delimitadoras se excluyen del valor del literal no numérico.
- A excepción de las comillas delimitadoras, todos los demás caracteres de puntuación dentro del literal se consideran parte del literal no numérico.
- Los literales utilizados para cálculos aritméticos no deben estar entre comillas como literales no numéricos. El literal "7.7" es un literal no numérico y se almacena de forma diferente al literal numérico 7.7 (no está entre comillas).

Numérico.

Un literal numérico es una cadena de caracteres usando los dígitos del 0 al 9, el signo más (+), el signo menos (-) y el punto decimal.

Las reglas para formar literales numéricos son las siguientes:

- Un literal puede tener entre 1 y 23 dígitos de longitud.
- Un literal debe contener al menos un dígito.
- Un literal no debe contener más de un carácter de signo. Si se utiliza un signo, debe aparecer como el carácter más a la izquierda del literal. Si el literal no tiene signo, es positivo.
- Un literal no debe contener más de un punto decimal. Un punto se asumirá como el punto decimal y podrá aparecer en cualquier lugar dentro del literal excepto como el carácter más a la derecha. Si el literal no contiene ningún punto decimal, el literal es un número entero. Un literal que cumple con las reglas para la formación de literales numéricos, pero que también está entre comillas, es un literal no numérico y es tratado como tal por el compilador.
- El valor de un literal numérico es la cantidad algebraica representada por los caracteres del literal numérico. Cada literal numérico pertenece a la categoría numérica. El tamaño de un literal numérico en caracteres es igual a la cantidad de dígitos que se especifiquen.

Punto flotante.

Un literal de punto flotante es una cadena de caracteres que utiliza dos números para representar un número original. El primer número se llama mantisa. Tiene un valor entre 0 (cero) y nueve. El segundo número se llama exponente. Representa la potencia de diez por la que se multiplica el primer número para obtener el número original.

La ventaja de la notación de punto flotante es que se pueden manejar números muy pequeños y muy grandes con facilidad. Puede utilizar literales de punto flotante como alternativa a la codificación de literales numéricos. Debe considerar el uso de literales de punto flotante con elementos de datos REAL y DOUBLE.

Las reglas para la formación de literales de punto flotante son las siguientes:

- La mantisa puede estar signada y debe tener un punto decimal.
- El exponente puede estar signado y debe ser un número entero.

Ejemplo:

8,765,432.1 es 8.7654321E6

Undigit.

Un literal undigit es una cadena de caracteres que representa el equivalente hexadecimal de un carácter EBCDIC.

Reglas para la formación de un literal undigit:

- Delimite ambos extremos del literal con el carácter arroba comercial (@).
- Seleccione los caracteres de los dígitos hexadecimales del 0 (cero) al 9 y los caracteres de la A a la F.

Reglas para el uso de un literal undigit:

Puede utilizar un literal undigit solo en contextos numéricos y alfanuméricos. En ambos contextos, se recomienda que los tamaños del literal undigit y del elemento de datos asociado coincidan. Los resultados de utilizar tamaños que no coinciden pueden ser impredecibles.

El compilador interpreta el literal undigit como un literal numérico de 4 bits (COMP) o como un literal alfanumérico de 8 bits. La interpretación del literal undigit se determina especificando el tipo de elemento de datos al que está asociado el literal sin dígitos.

Contextos alfanuméricos.

Un literal undigit es alfanumérico cuando la categoría del elemento de datos asociado es alfanumérico.

Contextos numéricos

Un literal undigit es numérico si aparece en la cláusula VALUE asociada con un elemento COMPUTATIONAL, independientemente de si el literal undigit aparece con ALL o no.

Los siguientes son ejemplos de literales undigit y sus equivalentes EBCDIC. Un programa podría incluir un literal undigit para enviar mensajes de secuencia de control a un terminal remoto.

Undigit literal	EBCDIC Equivalent
@0D@	CR (carriage return)
@25@	LF (line feed)

Conceptos de datos.

Para comprender la DATA DIVISION, debes comprender algunos de los conceptos que pertenecen a los datos de tu programa. El concepto de registro abarca la estructura, los niveles y las variables dentro del registro. El concepto de datos incluye la categorización de los datos, la alineación de los datos y la referencia a los datos. El concepto de tabla para manejar conjuntos de datos incluye manejo de índices y la indexación. El concepto de edición para formatear datos incluye el uso de símbolos para formatear datos, por ejemplo la supresión de ceros y reemplazos.

Registros.

El elemento de datos más inclusivo es el registro lógico. El registro se identifica mediante una entrada de nivel 01. En el registro se definen uno o más elementos de datos relacionados.

Una entrada de descripción de datos es una entrada en la DATA DIVISION que describe un elemento de datos. Cada entrada de descripción de datos consta de un número de nivel

seguido de un nombre para el dato, si es necesario, seguido de una serie de cláusulas independientes, según sea necesario.

Una descripción de registro es un conjunto de entradas de descripción de datos que describen las características de un registro en particular. Puedes especificar lo siguiente en una descripción de registro:

- Los elementos que se incluyen en el registro.
- El orden en que aparecen los elementos en el registro.
- La forma en que los elementos se relacionan entre sí en el registro.

El tamaño de una descripción de registro es la suma de los tamaños máximos de todos los elementos subordinados a un elemento de nivel 01. El tamaño máximo de un registro está restringido por el uso explícito o implícito del elemento de nivel 01 de acuerdo con la siguiente tabla:

Uso	Tamaño máximo
BINARY o DISPLAY	65,535 caracteres
COMP	65,535 dígitos
TASK, EVENT, INDEX, LOCK, o REAL	65,535 palabras

Niveles.

Los registros lógicos tienen subdivisiones para la referencia de datos. Los elementos en estas subdivisiones están organizados con un sistema de niveles. Una vez que se ha especificado una subdivisión, se puede subdividir aún más para permitir una referencia de datos más detallada.

Comprensión de variables elementales y agrupadas.

El elemento más pequeño de una descripción de datos se denomina variable elemental. Tenga en cuenta que las variables elementales no pueden tener niveles subordinados. Un registro consta de una secuencia de variables elementales o de una sola variable elemental.

Para hacer referencia a variables elementales como un conjunto, puede combinar las variables elementales en grupos. Cada grupo consta de una secuencia nombrada de una o más variables elementales. Puedes combinar grupos, a su vez, en grupos de dos o más grupos. Por lo tanto, una variable elemental puede pertenecer a más de un grupo. Puedes combinar los grupos con las cláusulas opcionales VALUE, USAGE o REDEFINES, seguida de un punto.

Un grupo incluye todas las variables del grupo y variables que lo siguen hasta que aparece un número de nivel menor o igual al número de nivel de ese grupo. Debes describir todas las variables inmediatamente subordinadas a un elemento de grupo determinado utilizando números de nivel idénticos mayores que el número de nivel utilizado para describir esa variable de grupo.

COBOL74 define que todas las variables agrupadas son alfanuméricas.

Alineación en el límite de bytes.

Si un elemento de grupo o un elemento EBCDIC elemental no va a comenzar en un límite de bytes, el compilador agrega un relleno de 4 bits para alinear el elemento en un límite de bytes. Cuando se lleva a cabo dicha alineación, el compilador incluye un mensaje “FILLER ADDED” en el listado de salida.

Limites.

Ninguna variable de datos elemental puede comenzar a más de 65,535 bytes (131,070 dígitos COMPUTACIONALES) desde el comienzo del registro de nivel 01 en el que está contenido el elemento de datos. Ninguna variable de datos elemental cuyo USO SEA COMPUTACIONAL puede comenzar mayor a 32,767 bytes (más precisamente, 65,535 dígitos COMPUTACIONALES) desde el comienzo del registro de nivel 01 en el que está contenido el elemento de datos.

Organización de datos con números de nivel.

Un sistema de números de nivel muestra la organización de variables elementales y variables agrupadas. Debido a que los registros son los elementos de datos más inclusivos, los números de nivel de los registros comienzan en 01. Debe asignar a las variables de datos menos inclusivos números de nivel superiores (pero no necesariamente sucesivos) que no sean mayores en valor que 49.

Construyendo un registro.

Numero de nivel	Entrada relacionada
01	La primera entrada en cada descripción de registro. Varias entradas de nivel 01 subordinadas a un indicador de nivel distinto de RD representan redefiniciones implícitas de la misma área.
02 hasta 49	La jerarquía de datos en un registro lógico.
66	Entradas de la cláusula RENAMEs para reagrupar elementos de datos.
77	Variables de la WORKING-STORAGE o LINKAGE SECTION que no están organizados en una jerarquía.
88	Nombres de condición que especifican valores de variables condicionales.

Datos.

COBOL74 admite las siguientes siete categorías de elementos de datos:

- Alfabético
- Numérico
- Editado numéricamente
- Alfanumérico
- Editado alfanumérico
- Kanji
- Editado kanji

Estas siete categorías de elementos de datos se agrupan en las tres clases siguientes:

- Alfabético
- Numérico
- Alfanumérico

Tablas.

Una tabla es un conjunto de elementos de datos lógicamente consecutivos que se especifican en la DATA DIVISION con una cláusula OCCURS. Al usar una tabla, puede hacer lo siguiente con sus datos:

- Definir elementos de datos contiguos.
- Acceder a un elemento por su posición en la tabla.
- Especificar la cantidad de repeticiones de un elemento.
- Identificar cada elemento con un subíndice o un índice.
- Acceder a elementos en tablas multidimensionales de longitud variable.
- Especificar claves ascendentes o descendentes.
- Busque en una dimensión de una tabla un elemento que satisfaga una condición específica.

La cláusula OCCURS especifica que el elemento debe repetirse la cantidad de veces que usted designe.

Para definir una tabla unidimensional, debe utilizar una cláusula OCCURS como parte de la descripción de los datos del elemento de la tabla. La cláusula OCCURS no debe aparecer en la descripción de los elementos del grupo que contienen el elemento de la tabla.

```
01 TABLE-1.  
    03 TABLE-ELEMENT OCCURS 20 TIMES.  
        05 NAME PIC X(30).  
        05 SSAN PIC 9(6).
```

Para crear una tabla bidimensional, debe definir una tabla unidimensional dentro de cada aparición de un elemento de otra tabla unidimensional. Para lograr esto, incluya una cláusula OCCURS en la descripción de datos del elemento de la tabla y en la descripción de solo un elemento de grupo que contenga ese elemento de tabla. Para definir una tabla tridimensional, debe incluir la cláusula OCCURS en la descripción de datos del elemento de la tabla y en la descripción de dos elementos de grupo que contengan ese elemento de tabla. En COBOL, puede definir tablas de hasta 48 dimensiones.

```
01 CENSUS-TABLE.  
    05 CONTINENT-TABLE OCCURS 8 TIMES.  
        10 CONTINENT-NAME PIC X(16).  
        10 COUNTRY-TABLE OCCURS 15 TIMES.  
            15 COUNTRY-NAME PIC X(18).  
            15 CITY-TABLE OCCURS 20 TIMES.  
                20 CITY-NAME PIC X(10).  
                20 CITY-POPULATION PIC X(12).
```

Acceder a las tablas.

Siempre que un programa hace referencia a un elemento de tabla, la referencia debe indicar la aparición prevista del elemento. Para acceder a una tabla unidimensional, el número de aparición del elemento deseado proporciona información completa. Para acceder a tablas de más de una dimensión, proporcione un número de aparición para cada dimensión de la tabla a la que se accede.

Subíndice.

Un subíndice es un número entero o una variable cuyo valor hace referencia a un elemento individual en una lista o tabla de elementos. Para utilizar un subíndice, primero debe proporcionar una cláusula OCCURS para definir múltiples ocurrencias de un elemento de datos. Los elementos de datos que se repetirán deben tener el mismo formato.

```
01 WKS-NOMBRES.  
    03 WKS-NOMBRE PIC X(30) OCCURS 5 TIMES.  
.  
.  
.  
ACCEPT WKS-NOMBRE (1).  
ACCEPT WKS-NOMBRE (I).
```

Índice.

Un índice, al igual que un subíndice, identifica los elementos individuales en una tabla de elementos similares.

El nombre dado en la frase INDEXED BY se usará para hacer referencia al índice asignado. El valor de un índice corresponde al número de ocurrencia de un elemento en la tabla asociada.

```
01 WKS-NOMBRES  
    03 WKS-NOMBRE PIC X(30) OCCURS 5 TIMES INDEXED BY I.
```

Para asignar un valor a un índice, el programa debe ejecutar una instrucción SET, SEARCH ALL, o un PERFORM VARYING.

Edición.

La edición es el proceso de utilizar símbolos en la cláusula PICTURE para especificar el formato de los datos para los listados/reportes. La operación de edición se produce cuando la variable se mueve del campo de envío al campo de recepción.

Algunas de las aplicaciones para las que podría querer utilizar la edición incluyen las siguientes:

- Separar campos con delimitadores como ceros o espacios
- Agregar asteriscos para proteger los importes en los cheques
- Agregar símbolos de crédito o débito para fines contables

- Formatear valores monetarios con comas (,) y signos de dólar (\$)
- Indicar valores positivos o negativos con signos más (+) y menos (–)
- Anteponer un punto decimal (.) a la parte fraccionaria de un importe.

Sending Area		Receiving Area	
PICTURE Clause	Data	Editing PICTURE Clause	Edited Data
9(5)	12345	\$ZZ,ZZ9.99	\$12,345.00
V9(5)	12345	\$\$\$,\$\$9.99	\$0.12
V9(5)	12345	\$ZZ,ZZ9.99	\$ 0.12
9(5)	00000	\$\$\$,\$\$9.99	\$0.00
9(3)V99	12345	\$ZZ,ZZ9.99	\$ 123.45
9(5)	00000	\$\$\$,\$\$\$.\$\$	
9(5)	01234	\$\$\$,\$\$\$.\$\$	\$1,234.00
9(5)	00000	\$**,***.**	*****.**
9(5)	00123	\$**,***.**	\$***123.00
9(3)V99	00012	\$ZZ,ZZ9.99	\$ 0.12
9(3)V99	12345	\$\$\$,\$\$9.99	\$123.45
9(3)V99	00001	\$ZZ,ZZ9.99	\$.01
9(5)	12345	\$\$\$,\$\$9.99	\$12,345.00
9(5)	00000	\$ZZ,ZZ9.99	
9(3)V99	00001	\$\$\$,\$\$\$.\$\$	\$0.01
S9(5)	(+) 12345	ZZZZ9.99+	12345.00+
S9(5)	(–) 00123	–99999.99	–00123.00
9(3)V99	12345	999.00	123.00
S9(5)	(–) 12345	ZZZZ9.99–	12345.00–
S9(5)	(+) 12345	ZZZZ9.99–	12345.00
9(5)	12345	BBB99.99	45.00
S9(5)V	(–) 12345	–ZZZZ9.99	–12345.00

S9(5)	(-) 12345	\$\$\$\$\$.99CR	\$12345.00CR
S9(5)V9(3)	(-) 12345	-.99	-12.34
S9(5)	(+) 12345	\$\$\$\$\$.99CR	\$12345.00
9(3)V99	12345	999.BB	123.
9(5)	12345	00999.00	00345.00

La tabla anterior muestra el orden de precedencia cuando se utilizan caracteres como símbolos en una cadena de caracteres.

Conceptos de la DATA DIVISION.

La tercera división de un programa fuente, la DATA DIVISION, describe los datos que el programa objeto acepta como entrada, manipula, crea o produce como salida. Los datos que se procesarán se dividen en tres categorías:

- Datos que están contenidos en archivos y que entran o salen de la memoria interna de la computadora desde un área o áreas específicas.
- Datos desarrollados internamente y colocados en un almacenamiento intermedio o de trabajo o en un formato específico para los reportes de salida.
- Constantes que usted define.

FILE SECTION.

La FILE SECTION define la estructura de los archivos de datos utilizados en el programa. Estos archivos han sido nombrados previamente y asignados a un dispositivo en la cláusula SELECT. Normalmente, cada archivo se define mediante una entrada de descripción de archivo (FD) y una o más descripciones de registro. Las descripciones de registro se escriben inmediatamente después de la entrada FD.

Entrada FD.

La entrada de descripción de archivo (FD) identifica un archivo declarado previamente en la cláusula SELECT y proporciona información sobre su estructura física. Las descripciones de registros del archivo siguen inmediatamente a su descripción de archivo.

Hay dos formatos para describir archivos. Estos formatos se utilizan de la siguiente manera:

1. Este formato describe los archivos de entrada y salida.
2. Este formato describe los archivos SORT y MERGED.

Cláusula BLOCK CONTAINS.

La cláusula BLOCK CONTAINS especifica el tamaño de un registro físico.

Cláusula RECORD CONTAINS.

Esta cláusula es necesaria cuando el registro físico contiene más de un registro lógico. Si no se especifica esta cláusula, se supone que el registro físico contiene un registro lógico tan grande como el registro más grande especificado para el archivo.

Esta cláusula no es necesaria en una entrada de archivo SD y no tiene efecto en la entrada de archivo SD si se especifica.

Cláusula VALUE OF.

La cláusula VALUE OF define los valores iniciales de los atributos de un archivo. Las cláusulas y frases descriptivas de la INPUT-OUTPUT SECTION y las descripciones de registros de archivos (excepto la cláusula VALUE OF) determinan implícitamente los valores iniciales de los atributos apropiados de un archivo. Sin embargo, estos valores de atributo pueden ser anulados o se pueden especificar otros atributos mediante la cláusula VALUE OF.

Esta cláusula no es necesaria en una descripción de combinación de clasificación (SD) y no tiene ningún efecto en la descripción de combinación de clasificación si se especifica.

Cláusula DATA RECORDS.

La cláusula DATA RECORDS es una cláusula opcional que documenta los nombres de los registros de datos asociados con un archivo.

Cláusula OCCURS.

La cláusula OCCURS define tablas. Si se utiliza la cláusula OCCURS, el nombre de datos que es el sujeto de esta entrada debe estar subindexada o indexada siempre que se haga referencia a él en una instrucción distinta a SEARCH o USE FOR DEBUGGING. Además, si el sujeto de esta instrucción es el nombre de una variable de grupo, entonces todos los nombres de datos que pertenecen al grupo deben subindexarse o indexarse siempre que se utilicen como operandos, excepto cuando los nombres de datos sean los objetos de una cláusula REDEFINES.

La cláusula OCCURS elimina la necesidad de entradas separadas para elementos de datos repetidos y proporciona la información necesaria para aplicar subíndices o índices.

La cláusula OCCURS tiene los dos formatos siguientes:

Formato	Explicación
1	Especifica que un elemento ocurre un número exacto de veces.
2	Especifica que un elemento ocurre un número variable de veces, dependiendo del elemento de datos al que hace referencia data-name-1.

Formato 1

$$\left\{ \begin{array}{c} \text{OCCURS} \\ \hline \text{OC} \\ \hline \end{array} \right\} \text{integer-2 TIMES}$$

$$\left[\begin{array}{c} \left\{ \begin{array}{c} \text{ASCENDING} \\ \hline \text{DESCENDING} \end{array} \right\} \text{KEY IS data-name-2 [, data-name-3]} \dots \\ \hline \end{array} \right]$$

INDEXED BY index-name-1 [, index-name-2]...

OCCURS or OC

OC es una abreviación de OCCURS.

Integer-2

Integer-2 no puede exceder el tamaño máximo de registro.

KEY IS

La frase KEY IS indica que el dato repetido es un arreglo en orden ascendente o descendente de acuerdo a el valor contenido en data-name-2 y data-name-3.

Los data-names se enumeran en orden descendente de importancia.

INDEXED BY

Se requiere una frase INDEXED BY si el sujeto de la entrada o una entrada subordinada a esta entrada se referencia mediante indexación. En nombre de índice identificado por esta cláusula no está definido en otra parte porque su asignación y formato dependen del hardware, y el nombre de índice no se puede asociar con ninguna jerarquía de datos porque no es un elemento de datos.

Formato 2

$$\left\{ \begin{array}{c} \text{OCCURS} \\ \hline \text{OC} \\ \hline \end{array} \right\} \text{integer-1 TO integer-2 TIMES } \underline{\text{DEPENDING ON data-name-1}}$$

$$\left[\begin{array}{c} \left\{ \begin{array}{c} \text{ASCENDING} \\ \hline \text{DESCENDING} \end{array} \right\} \text{KEY IS data-name-2 [, data-name-3]} \dots \\ \hline \end{array} \right]$$

INDEXED BY index-name-1 [, index-name-2]...

El formato 2 especifica que el sujeto de esta entrada tiene una cantidad variable de ocurrencias. El valor de integer-2 representa la cantidad máxima de ocurrencias y el valor de integer-1 representa la cantidad mínima de ocurrencias. La longitud del sujeto de la entrada no es variable, pero la cantidad de ocurrencias sí lo es.

Una entrada de descripción de datos que contiene el formato 2 de la cláusula OCCURS solo puede ir seguida en esa descripción de registro por entradas de descripción de datos que estén subordinadas a ella.

integer-1 TO integer-2

El valor de integer-1 debe ser menor que el valor de integer-2. Integer-1 e integer-2 no pueden superar el tamaño máximo de registro. Si se supera el límite, se produce un error de sintaxis.

DEPENDING ON

El valor de data-name-1 se utiliza para determinar el último elemento de la tabla al que se puede hacer referencia. Cuando el valor de data-name-1 es menor que el integer-2, los elementos de datos con números de ocurrencia que excedan el valor de data-name-1 son inaccesibles.

KEY IS

Si data-name-2 no es el sujeto de esta entrada, se aplican las tres condiciones siguientes:

- Todos los elementos identificados por los data-names en la frase KEY IS deben estar en el elemento del grupo que es el sujeto de esta entrada.
- Los elementos definidos por los data-names en la frase KEY IS no deben contener una cláusula OCCURS
- Ninguna entrada puede contener una cláusula OCCURS entre los elementos identificados por los nombres de datos en la frase KEY IS y el sujeto de esta entrada.

Clausula PICTURE.

La cláusula PICTURE describe el tipo de elemento de datos, el tamaño de una variable de datos y los requisitos de edición de un variable de datos elemental.

Existen los dos métodos siguientes para realizar la edición con la cláusula PICTURE:

- Edición de inserción
- Edición de reemplazo y supresión de ceros

El tamaño de una variable elemental es la cantidad de posiciones de caracteres que ocupa en el formato de datos estándar. El tamaño de un elemento elemental se indica mediante la cantidad de símbolos permitidos que representan posiciones de caracteres. Por ejemplo, 9999 indica un campo con cuatro dígitos.

Cláusula REDEFINES.

La cláusula REDEFINES permite que la misma área de almacenamiento de la computadora sea descrita por diferentes entradas de descripción de datos. Esta cláusula redefine el área de almacenamiento, no las variables de datos que ocupan el área.

Cláusula COMPUTATIONAL.

CMP y COMP son abreviaturas de COMPUTATIONAL. Las variables de datos computacionales elementales se representan internamente como dígitos contiguos de 4 bits. Un elemento COMPUTATIONAL puede representar un valor que se utilizará en los cálculos y debe ser numérico. Un literal numérico es una cadena de caracteres con caracteres seleccionados entre los dígitos "0" a "9", el signo más (+), el signo menos (-) y el punto decimal. Los dígitos "A" a "F" NO son numéricos. Los campos COMPUTATIONAL son elementos numéricos decimales empaquetados, no cadenas hexadecimales.

Si una variable agrupada se describe como COMPUTACIONAL, las variables elementales del grupo son COMPUTACIONALES. El grupo en sí no es COMPUTACIONAL y no se puede utilizar en cálculos.

Conceptos de la PROCEDURE DIVISION.

La última división de un programa, la PROCEDURE DIVISION, debe incluirse en cada programa COBOL.

Encabezado de la PROCEDURE DIVISION.

La PROCEDURE DIVISION se identifica y debe iniciar con el siguiente formato:

$$\begin{array}{c} \text{PROCEDURE DIVISION} \end{array} \left[\begin{array}{c} \text{USING} \left\{ \begin{array}{l} \text{data-name} \\ \text{file-name} \\ \text{task-name} \\ \text{event-name} \\ \text{lock-name} \end{array} \right\} \dots \end{array} \right] .$$

La cláusula opcional USING nombra los identificadores recibidos como parámetros en una tarea, procedimiento enlazado, librarys o el ambiente de comunicación entre programas (IPC).

Un data-name, file-name, task-name, event-name, o lock name en la cláusula USING del encabezado PROCEDURE DIVISION debe definirse en la LINKAGE SECTION del programa en el

que se encuentra este encabezado. El data-name, el task-name, event-name, y el lock name deben declararse en el nivel 77 o 01 y no deben ser elementos redefinidos.

Cuerpo de la PROCEDURE DIVISION.

El cuerpo de la PROCEDURE DIVISION debe conformarse por uno de los dos formatos a continuación:

Format 1

```
[  
  DECLARATIVES.  
  {  
    section-name SECTION [segment-number]. declarative sentence.  
    [paragraph-name. [sentence]...]...  
  }...  
  END DECLARATIVES.  
]
```

Format 2

```
{paragraph-name. [sentence]...}...
```

Las DECLARATIVES SECTIONS deben agruparse al inicio de la PROCEDURE DIVISION y deben estar precedidas por la palabra clave DECLARATIVES y seguidas por las palabras clave END DECLARATIVES. Una sección consta de un encabezado de sección seguido de uno o más párrafos sucesivos. Una sección termina de la siguiente manera:

- Inmediatamente antes de la siguiente sección.
- Al final de la PROCEDURE DIVISION.
- En las palabras clave END DECLARATIVES en la DECLARATIVES SECTIONS.

Un párrafo consta de un nombre de párrafo seguido de un punto y de una o más sentencias sucesivas. Un párrafo termina de la siguiente manera:

- Inmediatamente antes del siguiente nombre de párrafo o nombre de sección.
- Al final de la PROCEDURE DIVISION.
- En las palabras clave END DECLARATIVES en la DECLARATIVES SECTIONS.

Una sentencia consta de una o más instrucciones y termina en un punto.

Una instrucción es una combinación válida de palabras y símbolos que comienzan con un verbo.

Categorías de instrucciones y sentencias.

Las instrucciones y sentencias pueden ser de los tres tipos siguientes:

- Condicionales.
- Imperativos que dirigen al compilador.
- Imperativos que dirigen al programa.

Condicionales.

Una declaración condicional contiene una condición que puede ser VERDADERA o FALSA y especifica las acciones que se deben tomar en función del valor verdadero de la condición. El programa realiza una prueba para determinar el valor de la condición y realiza una acción específica cuando la condición es VERDADERA y otra acción específica cuando la condición es FALSA.

Imperativos que dirigen al compilador.

Una declaración imperativa que dirige al compilador hace que éste realice una acción específica durante el proceso de compilación.

Imperativos que dirigen al programa.

Una declaración imperativa que dirige un programa hace que éste realice una acción incondicional específica. Una declaración imperativa puede consistir en una secuencia de instrucciones imperativas.

Agrupaciones funcionales de verbos.

Las siguientes listas clasifican los verbos COBOL74 por función y resumen sus propósitos. Algunos verbos aparecen en más de una categoría funcional.

Conceptos en la PROCEDURE DIVISION.

CALL	Pasa el control de un programa que llama a un programa llamado.
CANCEL	Libera las áreas de memoria ocupadas por el programa llamado.
EXIT PROGRAM	Marca el final lógico de un programa llamado.
STOP RUN	Finaliza permanentemente el programa llamado y todos los demás programas en la unidad de ejecución.

Verbos aritméticos.

ADD	Suma dos o más operandos numéricos y almacena el resultado.
COMPUTE	Asigna el valor de una expresión aritmética a uno o más elementos de datos.
DIVIDE	Divide un operando numérico en uno o más operandos y almacena el cociente y el resto.
MULTIPLY	Multiplica operandos numéricos y almacena el resultado.
SUBTRACT	Resta uno o la suma de dos o más números operandos de uno o más elementos y almacena el resultado.

Verbos de dirección al compilador.

COPY	Incorpora texto de un programa de biblioteca a un programa fuente COBOL.
USE	Especifica procedimientos para manejar errores de entrada-salida además de los procedimientos estándar proporcionados por el sistema de control de entrada-salida.

Verbos de movimiento de datos.

ACCEPT	Hace que los datos de bajo volumen estén disponibles para un elemento de datos específico. Datos de DATE, DAY, TIME, TIMER, TODAYS-DATE o el registro TODAYS-NAME se mueve al elemento especificado.
MOVE	Transfiere datos, según las reglas de edición, a uno o más áreas de datos.
STRING	Concatena el contenido parcial o completo de uno o más elementos de datos en una única variable.
UNSTRING	Hace que los elementos de datos contiguos en un campo de envío se separen y se coloquen en múltiples campos de recepción.

Verbos imperativos.

ACCEPT	Hace que los datos de bajo volumen estén disponibles para un elemento de datos específico desde la terminal del operador.
ADD	Suma dos o más operandos numéricos y almacena el resultado.
ALLOW	Ejecuta un procedimiento de interrupción que se ha adjuntado a un elemento de EVENTO.
ALTER	Modifica el destino de una declaración GO TO etiquetada
ATTACH	Asocia un procedimiento de interrupción con un elemento de EVENTO
CALL	Transfiere el control a una tarea o procedimiento separado.
CLOSE	Finaliza el procesamiento de un archivo, un carrete o una unidad de un archivo. También especifica la disposición del archivo y del dispositivo a que está asignado el archivo.
COMPUTE	Asigna el valor de una expresión aritmética a uno o más elementos de datos
DELETE	Elimina un registro de forma lógica de un archivo relativo o indexado.
DISALLOW	Impide la ejecución de un procedimiento de interrupción que tiene se ha adjuntado a un evento.
DISPLAY	Hace que se transfieran datos de bajo volumen a una terminal de operador.

DIVIDE	Divide un operando numérico en uno o más operandos y almacena el cociente y el resto.
GO TO	Transfiere el control incondicionalmente de un procedimiento a otro. El control no se devuelve implícitamente a la declaración que sigue a la declaración GO.
IF	Evalúa una condición. Acción posterior del objeto. El programa depende de si el valor de la condición es verdadero o falso.
LOCK	Permite que un proceso niegue a los procesos relacionados el acceso a un área de almacenamiento común o pruebe un área de almacenamiento común para detectar una condición de bloqueo.
MULTIPLY	Multiplica operandos numéricos y almacena el resultado.
OPEN	Hace que un archivo esté disponible para su procesamiento.
PERFORM	Transfiere el control incondicionalmente a un procedimiento o a un grupo de procedimientos consecutivos y devuelve el control a la instrucción que sigue a la instrucción PERFORM.
READ	Para acceso secuencial, pone a disposición la siguiente lógica registro de un archivo secuencial. Para acceso aleatorio, hace disponible un registro específico de un archivo de almacenamiento masivo.
RELEASE	Transfiere registros a la fase inicial de una operación de clasificación, y escribe registros en un archivo de clasificación.
RETURN	Obtiene registros ordenados de una operación de clasificación o registros combinados de una operación de combinación.
REWRITE	Reemplaza un registro de forma lógica en un archivo de almacenamiento masivo.
RUN	Inicia otro programa como independiente, asíncrona tarea.
SEARCH	Busca en una tabla un elemento de tabla que satisfaga una condición especificada y ajusta el nombre de índice asociado para indicar ese elemento de la tabla.
SET	Establece puntos de referencia para las operaciones de manejo de tablas estableciendo índices asociados con los elementos de la tabla. También puede alterar el valor de interruptores externos y variables condicionales.
SORT	Secuencia los registros de un archivo en un conjunto de claves especificadas y hace que los registros ordenados estén disponibles para procedimientos de salida o archivos de salida.
START	Posiciona registros lógicamente en un archivo relativo o indexado cuando el archivo debe leerse secuencialmente.
STOP	Suspende un programa de forma permanente o temporal

STRING	Concatena el contenido parcial o completo de uno o más elementos de datos en un único elemento de datos.
SUBTRACT	Resta uno o la suma de dos o más números operandos de uno o más elementos y almacena los resultados.
UNLOCK	Desbloquea un área de almacenamiento común para que los procesos relacionados puedan acceder a ella.
UNSTRING	Hace que los elementos de datos contiguos en un campo de envío se separen y se coloquen en múltiples campos de recepción
USE	Especifica procedimientos para el manejo de excepciones de E/S.
WAIT	Suspende la ejecución del programa por un período específico de tiempo.
WRITE	Libera un registro lógico para una salida o una entrada-salida archivo.

Verbos de entrada y salida.

ACCEPT	Transfiere datos de bajo volumen desde la terminal del operador hacia una variable específica.
CLOSE	Finaliza el procesamiento de un archivo y especifica la disposición del archivo y del dispositivo al que está asignado el archivo.
DELETE	Elimina un registro lógico de un archivo relativo o indexado
DISPLAY	Hace que se transfieran datos de bajo volumen a una terminal de operador.
OPEN	Hace que un archivo esté disponible para su procesamiento.
READ (AT END, INVALID KEY)	Para acceso secuencial, pone a disposición la siguiente lógica registro de un archivo secuencial. Para acceso aleatorio, hace disponible un registro específico de un archivo de almacenamiento masivo.
REWRITE	Reemplaza un registro de forma lógica en un archivo de almacenamiento masivo.
STOP (literal)	Suspende la ejecución de un programa. La literal es comunicada al operador, y la ejecución continua con la siguiente instrucción ejecutable en el programa.
START	Proporciona una posición lógica para un archivo relativo o indexado cuando el archivo debe leerse secuencialmente.
WRITE	Libera un registro lógico para una salida o una entrada-salida archivo.

Verbos de comunicación entre programas.

- | | |
|--------------|---|
| CALL | Transfiere el control de un programa a otro. |
| CANCEL | Garantiza que la próxima vez que se llame a un programa al que se hace referencia en una declaración CALL, el programa estará en su estado inicial. |
| EXIT PROGRAM | Indica el final lógico de un programa llamado. |

Verbos de manejo de cadenas.

- | | |
|----------------------------------|--|
| INSPECT
(REPLACING, TALLYING) | Busca y cuenta o reemplaza las apariciones de caracteres especificados en una variable. |
| STRING | Concatena el contenido completo o parcial de una o más variables. |
| UNSTRING | Provoca que variables en un campo de envío sean separados y colocados dentro de múltiples campos de recepción. |

Verbos de manejo de tablas.

- | | |
|--------|---|
| SEARCH | Busca en una tabla por un elemento que satisface una condición específica, y ajusta el índice asociado para indicar ese elemento de la tabla. |
| SET | Establece puntos de referencia para las operaciones del manejo de tablas, estableciendo índices asociados con los elementos de la tabla. También altera el valor de interruptores externos y variables condicionales. |

LIBRARIES

Una library es un programa que proporciona un procedimiento o un conjunto de procedimientos que pueden ser llamados por otros programas. El procedimiento o conjunto de procedimientos están en código objeto. Por lo tanto, se puede considerar un programa de library como una colección de objetos de library a los que se puede acceder a través de un entry point. Cada objeto de la library es accesible para otros programas, incluidos otros programas de la library.

El programa que llama al objeto de la library se llama programa usuario o llamador. Las libraries no pueden ser llamadas recursivamente; es decir, un programa llamado no puede llamar a otro programa que, a su vez, llama al programa original.

Creando una LIBRARY.

El compilador crea automáticamente un programa de library COBOL74, siempre que se cumplan las siguientes restricciones para el programa de library:

- Sólo los parámetros permitidos para las libraries aparecen en la frase USING de la instrucción PROCEDURE DIVISION.
- No ocurren sentencias que sean incompatibles con su uso como library. Por ejemplo, una instrucción VALUE o una cláusula RECEIVED BY CONTENT dentro de una variable listada en la cláusula USING del encabezado PROCEDURE DIVISION o una directiva USE AS EXTERNAL PROCEDURE en la DECLARATIVES SECTION de la PROCEDURE DIVISION.

Una library tiene un único entry point. El nombre del entry point es PROCEDUREDIVISION, a menos que la library contenga un nombre de programa en la cláusula PROGRAM-ID en la IDENTIFICATION DIVISION y es compilada con la opción de compilador FEDLEVEL = 5. Si fuera el caso, el nombre descrito en la cláusula PROGRAM-ID sería usado como el nombre del entry point.

PROCEDURE DIVISION encabezado en una library.

Los parámetros son opcionales en las libraries, sin embargo, si los parámetros son usados, el programa library debe identificarlos con la cláusula USING en la PROCEDURE DIVISION.

En una library de COBOL74, la cláusula USING del encabezado PROCEDURE DIVISION puede especificar los siguientes tres tipos de parámetros:

- Variables por referencia
- Variables pasadas por referencia con tipo de conversión
- Archivos pasados por referencia

El formato del encabezado es como a continuación:


```

PROCEDURE DIVISION [ USING { data-name
                           file-name } ... ] .

```

El data-name debe definirse en la LINKAGE SECTION del programa en el que aparece el encabezado PROCEDURE DIVISION debe tener un número de nivel 01 o 77 y no debe ser un elemento redefinido.

El file-name debe definirse en la FILE SECTION del programa.

Reglas para parámetros.

Los parámetros para los entry point de una library pueden ser cualquiera de los siguientes tipos:

COMP, 01, 77	INDEX
BINARY, 01, 77	INTEGER (COMP)
DISPLAY, 01	REAL
DOUBLE	STRING (DISPLAY)
FILE	

Cada uno de los elementos de datos en la frase USING debe definirse como nivel 01 o 77, y no debe redefinir otro elemento de datos

Salir de una Library.

Un programa de library COBOL debe tener una declaración EXIT PROGRAM como medio para salir y regresar al programa de usuario. Una instrucción STOP RUN en un programa de library finaliza tanto la library como el programa que llama.

La instrucción EXIT PROGRAM hace que el control del programa se devuelva al programa de usuario en la instrucción que sigue a la instrucción que llamó a la library.

Asegurando una Library.

El bloqueo de código se provee por el compilador en la library para garantizar la integridad de los datos.

Debido a que los datos son globales tanto para el programa usuario como para el entry point, y debido a que los parámetros de la library deben permanecer globales, el compilador restringe el uso de una library COBOL a un usuario a la vez. Debe esperar hasta que el usuario anterior haya terminado antes de poder ingresar a la library.

Si un programa de library es llamado por programas de usuario escritos en otros lenguajes y la library se declara con la opción de control del compilador SHARING igual a PRIVATE, entonces la opción LIBRARYLOCK debe establecerse en TRUE para garantizar la integridad de los datos.

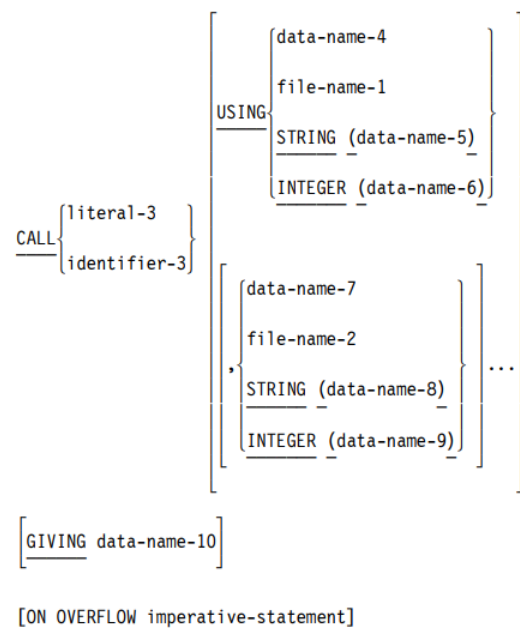
Invocar a una Library.

Un programa usuario puede hacer referencia a un programa de library utilizando la instrucción CALL o la instrucción CANCEL. La instrucción CALL hace que el control del programa pase desde el programa usuario al entry point especificado de la library. La declaración CANCEL solicita que el sistema operativo finalice la library.

Los programas de library escritos en COBOL74 tienen un único entry point, denominado PROCEDUREDIVISION de forma predeterminada. Si un programa que se llama tiene una cláusula PROGRAM-ID en la IDENTIFICATION DIVISION y tiene la opción de control del compilador FEDLEVEL establecida en 5, el nombre que aparece en la cláusula PROGRAM-ID es el nombre del entry point. En todos los demás casos, el entry point a las librarys COBOL74 es PROCEDUREDIVISION.

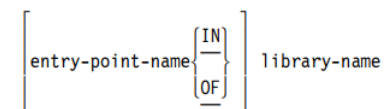
Declaración CALL para Libraries.

La library que se llama se puede describir de diferentes maneras dependiendo de si la library se llama con un literal o con un identificador, y dependiendo de si al atributo LIBACCESS se le asigna el valor BYTITLE (el valor predeterminado) o el valor BYFUNCTION.



Explicación del formato:

Literal-3: El contenido de literal-3 puede ser como a continuación



El nombre del entry point es el nombre del entry point en la library o la función que se llama.

El library-name es un título de library si el atributo LIBACCESS es BYTITLE, y es nombre de función si el atributo LIBACCESS es BYFUNCTION.

Un título de library es el nombre del objeto de la library que contiene el entry point. Un título de library tiene significado sólo si el atributo LIBACCESS del programa de library es BYTITLE (el valor predeterminado). Un título de library puede contener un código de usuario como primer nodo de directorio, pero no puede contener un nombre de familia (por ejemplo, ON DISK).

Un nombre de función es el nombre mediante el cual una library del sistema establecida se pone a disposición de los usuarios. Un nombre de función tiene significado sólo si el atributo LIBACCESS del programa de library es BYFUNCTION. El nombre de una función está limitado a un máximo de 17 caracteres alfabéticos, numéricos y en mayúsculas y no debe contener barras (/) ni otros caracteres especiales.

USING.

La cláusula USING identifica elementos de datos y archivos pasados como parámetros al procedimiento de la library y también habilita que puedas manipular algunos tipos de parámetros con los mecanismos de conversión de tipo STRING e INTEGER.

Las variables usadas como parámetros para el entry point de la library puede ser BINARY, COMP, DISPLAY, DOUBLE, INDEX o REAL.

La frase STRING (data-name-5) convierte el elemento DISPLAY especificado por data-name-5 en una representación de cadena para que el elemento pueda pasarse a un parámetro de cadena de un objeto de library. El tipo de datos de cadena no se puede representar directamente en COBOL o COBOL74, pero aparece en otros lenguajes como ALGOL.

La frase INTEGER (data-name-6) convierte el elemento COMP especificado por data-name-6 a una representación de enteros. El compilador genera el código objeto para pasar el valor numérico del campo empaquetado en una variable temporal como un entero por referencia y para almacenar el valor de esa variable temporal (en caso de que la library la haya modificado) nuevamente en el parámetro original. Data-name-6 debe ser el nombre de un elemento de datos elemental de nivel 01 o 77 declarado como USAGE COMPUTATIONAL.

GIVING.

La cláusula GIVING permite llamar a un procedimiento escrito como library y almacenar el valor del procedimiento en data-name10.

Data-name-10 debe especificar una variable numérica elemental. Las reglas con respecto a la relación entre el uso de data-name-10, el tipo de procedimiento, y el resultado obtenido por el procedimiento son descritos a continuación.

Uso de data-name-10**Tipo de procedimiento y resultado devuelto**

Nivel 77 USAGE REAL

El programa usuario espera que el procedimiento de la library sea un REAL PROCEDURE y el resultado sea de tipo REAL

Nivel 77 USAGE DOBLE

El programa usuario espera que el procedimiento de la library sea un DOUBLE PROCEDURE y el resultado sea de tipo DOUBLE

Todos los otros casos

El programa de usuario espera que el procedimiento de la library sea un INTEGER PROCEDURE. El resultado devuelto se convierte en el momento de la ejecución al formato adecuado para su almacenamiento en data-name-10 de acuerdo con las reglas de la declaración COMPUTE.

Efecto del estado de una library en una declaración CALL.

Los programas usuario pueden depender de que una library se encuentre en su estado inicial o no inicial en un llamado a la library. Una library está en su estado inicial la primera vez que cualquier usuario se liga a ella con éxito después de que la library comienza a ejecutarse.

El estado de una library depende del entorno en el que se utiliza y del valor de la opción de control del compilador SHARING.

A continuación, se muestra cómo el valor de la opción SHARING afecta el estado inicial de la library:

Valor de la opción SHARING	Efecto en el estado inicial de la Library
SHARED BY RUN UNIT	<p>La library está en su estado inicial en los siguientes momentos:</p> <p>La primera vez que cualquier programa en la unidad de ejecución llama a la library.</p> <p>La primera vez que cualquier programa en la unidad de ejecución llama a la library después de que se haya realizado con éxito una declaración CANCEL en la library desde dentro de la unidad de ejecución.</p>
PRIVATE	<p>La library está en su estado inicial en los siguientes momentos:</p> <p>Cada vez que el programa utiliza una declaración CALL con identifier-3 para llamar a la library. Este formato hace que la library comience y complete la ejecución cada vez que este formato es usado.</p> <p>La primera vez que el programa utiliza una declaración CALL con literal-3 para llamar a la library.</p> <p>La primera vez que el programa llama a la library después de haberla cancelado correctamente.</p>

SHARED BY ALL	<p>Una library temporal está en su estado inicial en los siguientes momentos:</p> <p>La primera vez que un programa llama a la library.</p> <p>La primera vez que un programa llama a la library después de que el sistema operativo haya determinado que no existen más usuarios y, por lo tanto, haya finalizado la library.</p> <p>Una library permanente está en su estado inicial la primera vez que un programa llama a la library.</p>
---------------	---

Instrucción CANCEL para Libraries.

La instrucción CANCEL, hace que el sistema operativo desligue la library especificada del programa que la llama e intente finalizar la library.

La instrucción CANCEL informa al sistema operativo que el vínculo entre el programa que realiza el llamado y la library será cortado.

El contenido de la literal o el identificador usado en la declaración CANCEL debe ser un nombre de library válido y puesto entre comillas.

Efecto del estado inicial de una library en una declaración CANCEL.

El funcionamiento y efecto de la instrucción CANCEL en el estado inicial de la library dependen del valor de la opción de control de compilador SHARING de la library.

Opción SHARING y efecto del estado inicial de la library en la declaración CANCEL:

Valor de SHARING	Efecto en el estado inicial de la Library
SHARED BY RUN UNIT o PRIVATE	<p>La library se envía al end-of-task (EOT). La ejecución repetida de las instrucciones CALL y CANCEL en la misma library es costosa debido a la sobrecarga envuelta en la terminación y reinicio de la tarea.</p> <p>El descuido en el uso de la sentencia CANCEL cuando la opción SHARING es igual a PRIVATE o SHARED BY RUN UNIT puede llevar a sistemas de aplicación extremadamente ineficientes.</p>
SHARED BY ALL	<p>La library no se envía al EOT. En cambio, el sistema operativo emite una advertencia en tiempo de ejecución indicando que la library fue desvinculada y no finalizada. Cuando se ejecuta la siguiente instrucción CALL para la library, la library se encuentra en el estado en el que se encontraba cuando ejecutó por última vez una instrucción EXIT PROGRAM.</p>

Atributos de la Library.

Los siguientes párrafos describen los tipos de atributos que son válidos para las libraries y la manera en la cual los atributos de una library pueden ser cambiados.

Tipos de atributos de la library.

Las libraries, como los archivos, tienen atributos que se pueden configurar. Los siguientes cinco atributos son asociados con las libraries.

- FUNCTIONNAME
- INTNAME
- LIBACCESS
- LIBPARAMETER
- TITLE

Todos los atributos excepto el atributo LIBACCESS son atributos EBCDIC de tipo STRING.

El atributo LIBACCESS es un atributo tipo mnemónico que tiene dos posibles valores: BYTITLE (por default) y BYFUNCTION. El efecto que los valores del atributo LIBACCESS tiene sobre los atributos TITLE y FUNCTIONNAME son:

Valor del atributo LIBACCESS	Efecto
BYFUNCTION	Configurar el atributo LIBACCESS en el valor BYFUNCTION tiene los siguientes efectos: El atributo TITLE no tiene efecto. El atributo FUNCTIONNAME es usado para localizar el archivo código apropiado en la tabla de funciones de la library del sistema operativo. El archivo código asociado con el atributo FUNCTIONNAME es usado.
BYTITLE	El atributo TITLE se utiliza para acceder al archivo de código.

El valor predeterminado del atributo INTNAME es el último nodo del título del archivo que se está llamando. Por ejemplo, dada una declaración con el formato CALL "X OF A/B/C/D", el INTNAME de la library a la que hace referencia esta declaración CALL es D.

Instrucción CHANGE ATTRIBUTE para libraries.

Los atributos pueden ser cambiados dinámicamente por el programa usuario antes del primer intento para vincularse a la library; estos no pueden ser cambiados después de que se estableció la vinculación.

La instrucción CHANGE ATTRIBUTE es usada para cambiar atributos de la library. El formato de esta instrucción es como a continuación:

```
CHANGE ATTRIBUTE library-attribute  $\left\{ \begin{array}{c} \text{OF} \\ \text{IN} \end{array} \right\}$  "library-id"
```



```
 $\left\{ \begin{array}{c} \text{identifer} \\ \text{literal} \\ \text{mnemonic-attribute-value} \end{array} \right\}$ 
```

El library-id especifica la library que ha cambiado los atributos y tiene el mismo formato que el library-id de tiempo de ejecución en el Formato 3 de la declaración CALL. El identifer debe ser un elemento de datos alfanumérico con USAGE IS DISPLAY. El literal debe ser un literal no numérico. El mnemonic-attribute-value, utilizado solo con el atributo LIBACCESS, debe ser BYTITLE o BYFUNCTION.

Los atributos de la library no se pueden cambiar después de que se emite una declaración CALL para la library a menos que la library haya sido cancelada.

Opciones de control de compilador de la Library.

Las opciones de control del compilador LIBRARYLOCK, SHARING y TEMPORARY controlan la manera en la que la library es usada.

LIBRARYLOCK.

Si una library es llamada únicamente por programas COBOL74 y la library es declarada con la opción SHARING igual a PRIVATE, entonces la opción LIBRARYLOCK puede permanecer FALSE.

Si una library es llamada por programas escritos en otros lenguajes y la library se declara con la opción SHARING igual a PRIVATE, entonces la opción LIBRARYLOCK debe establecerse en TRUE para garantizar la integridad de los datos.

Si la opción SHARING no es PRIVATE, la configuración de la opción LIBRARYLOCK no tiene efecto en la generación del código de bloqueo de la library.

SHARING.

El creador de la library especifica los usos simultáneos permitidos de una library mediante la opción SHARING.

A continuación, se muestran los distintos valores de las opciones y su significado.

Valor	Significado
PRIVATE	Se inicia una instancia independiente de la library para cada usuario.
SHARED BY ALL	Todos los usuarios simultáneos comparten la misma instancia de la library.
SHARED BY RUN UNIT	Una unidad de ejecución consta de un programa, todas las libraries SHARED BY RUN UNIT se inician directamente, y cualquier library SHARED BY RUN UNIT se inicia cuando el código de otra library congelada se ejecuta en la pila del programa.

TEMPORARY.

La opción de control del compilador TEMPORARY, junto con la opción SHARING, determina si una library creada por el compilador COBOL funciona como library temporal o permanente. La eficacia de la opción TEMPORARY depende del valor de la opción SHARING. Una library puede

convertirse en permanente solo si la opción SHARING se ha especificado como SHARED BY ALL o DONT CARE.

Se crea una library temporal si el valor de la opción TEMPORARY es TRUE y la opción SHARING tiene el valor SHARED BY ALL o DONT CARE. Una library temporal termina si ningún usuario hace referencia a ella. Si se vuelve a llamar a la library en un momento posterior, se inicia una nueva copia.

Se crea una library permanente si el valor de la opción TEMPORARY es FALSE (valor predeterminado) y la opción SHARING tiene el valor SHARED BY ALL o DONT CARE. Una library permanente permanece en la mezcla sin usuarios y sus procedimientos están disponibles para cualquier programa posterior. Una library permanente se termina solo mediante la acción directa del operador.

CANDE

Comman and Edit (CANDE) es un sistema de control de mensajes (MCS message control system) que permite:

- Crear, editar y administrar archivos.
- Controlar e iniciar tareas (tal como compilar o ejecutar programas).
- Cuestionar y controlar tareas.
- Iniciar y consultar tareas de comunicación datos en red.

Transmisión de comandos a CANDE.

En el emulador Web Enabler podemos transmitir la información de dos maneras desde nuestra terminal de CANDE hacia el servidor Unisys para su procesamiento, por línea y por página.

Es muy importante que estemos conscientes de que información deseamos transmitir al servidor, de esto depende el tipo de transmisión que utilizaremos.

- Por línea, se transmitirá únicamente los caracteres de la línea donde esta posicionado el cursor, desde el comienzo de la línea hasta donde este colocado el cursor. Normalmente se usa la combinación de teclas Ctrl + Enter, para transmitir por línea.
- Por página, se transmitirán TODOS los caracteres que hay desde el inicio de la página hasta donde este colocado el cursor. Normalmente se usa la tecla Enter, para transmitir por página.

Identificación de usuario e inicio de sesión.

Antes de poder utilizar la mayoría de los comandos en CANDE, tu estación debe estar activada y conectada al sistema. Para iniciar sesión, un usercode valido, y en la mayoría de los casos una password, deben presentarse al sistema.

Además de un usercode y una password, algunas instalaciones requieren otro accesscode y una accesscode password. Un accesscode restringe el acceso a determinados archivos y proporciona una capa adicional de seguridad.

Condición BREAK.

Aunque ?BRK no es un comando CANDE, el software de emulación de terminal estándar de Unisys emite una condición de interrupción cuando un usuario ingresa ?BRK. Cuando ocurre una condición de interrupción, CANDE notifica inmediatamente a cualquier archivo remoto abierto con capacidad de salida asignado a la estación del usuario y finaliza cualquier listado o actividad similar que pueda generar salida a una estación.

Por ejemplo, ?BRK finaliza los comandos FILE, FIND, MATCH y LIST.

Respuesta a comandos CANDE.

Excepto por la entrada de una sola línea y ciertos comandos de control, todos los comandos enviados a CANDE dan como resultado la visualización de un prompt (#), ya sea solo o acompañado de un mensaje.

El progreso en el procesamiento de un comando puede indicarse mediante un prompt (#) con una respuesta (por ejemplo, “#UPDATING”). Si el comando no se ha completado correctamente, el prompt va seguido de un mensaje de error. La finalización exitosa de un comando CANDE da como resultado la visualización del prompt (#) solo o con mensajes relacionados con los resultados del comando.

Manipulación de la cola de texto guardado.

Cada usuario tiene una cola de texto guardada que CANDE mantiene como referencia. El texto guardado más reciente (<0>) se puede editar y procesar como la siguiente línea de entrada. Los comandos proporcionados para enumerar y editar el texto guardado son ?SHOW, ?EDIT, ?REPEAT, and ?RETRIEVE.

Cada entrada en la cola sube un nivel. Si la cola de texto guardada contiene el número máximo de entradas, se descarta la entrada más antigua.

Manipulación de la cola de entrada.

CANDE puede ejecutar un número ilimitado de entradas en cola. La entrada en cola se encuentra en uno de tres estados: normal, pendiente o en espera.

- El estado normal ocurre cuando CANDE está ocupado ejecutando un comando para una estación y se ingresa otro comando antes de que se complete el primero. Se emite el mensaje “#QUEUED” al usuario, indicando que el nuevo comando se ha colocado en la cola de entrada. Si el comando que se está ejecutando actualmente se completa normalmente, entonces el primer comando en cola se convierte en el comando que se está ejecutando actualmente, y así sucesivamente. Si se produce un error en el comando que se está ejecutando actualmente y existe una entrada en cola, la entrada en cola estará pendiente o en espera.
- El estado pendiente ocurre si se proporciona un mensaje “#QUEUED INPUT PENDING”; Si se ingresa una línea normal de entrada después del mensaje, entonces la entrada en cola se descarta y la sesión continua con esa línea de entrada. Si la entrada en cola no debe descartarse, entonces se puede usar uno de los comandos de manipulación de la cola y la cola de entrada se puede establecer en el estado de espera.
- En el estado de espera, cualquier línea normal de entrada se pone en cola al final de la cola y se muestra el mensaje “#QUEUED”. Luego se muestra el mensaje “#QUEUED INPUT WAITING”.

Los comandos proporcionados para la manipulación de entradas en cola son ?GO, ?WAIT, ?PURGE, ?TAKE y ?ENTER.

Work Files.

CANDE puede acceder a cualquier archivo del usuario para diversos fines, pero sólo se pueden hacer cambios mediante el work file. Se puede crear un nuevo work file usando el comando MAKE, o se puede usar un archivo existente como work file usando el comando GET.

Iniciar el modo de página y moverse por el archivo de trabajo.

Los comandos que inician el modo de página y permiten el movimiento a través del work file se describen brevemente en la siguiente lista:

Comando	Definición
PAGE	Muestra una página que comienza con un número de secuencia especificado.
NEXT +/-	Muestra una página después de cambiar de la pagina mostrada actualmente hacia adelante (+) o hacia atrás (-) un número específico de páginas.
SAME	Actualiza la página mostrada actualmente.
+,-	Muestra una página después de cambiar de la página mostrada actualmente hacia adelante (+) o hacia atrás (-) un número específico de registros.
SEQ	Muestra una página con una columna de números de secuencia, determinada por la base y el incremento especificados, en las ubicaciones más a la izquierda de la pantalla.

El indicador de columna.

Una vez que se ha iniciado el modo de página, aparece un indicador de columna en la línea superior de la pantalla que especifica las columnas de texto reales para un registro en el archivo.

Asignación de números de secuencia a registros.

Los números de secuencia se pueden asignar a los registros manualmente si la secuencia está dentro del rango de los registros numerados. Los ceros iniciales de los números de secuencia se pueden reemplazar por espacios en blanco si se desea. Por ejemplo, si b significa un espacio en blanco, 00000100 se puede escribir como bbbbbb100. Las líneas en blanco sin numerar transmitidas al final de la página se ignoran. Sin embargo, si se desean líneas en blanco en la parte inferior de la página, se debe escribir un número de secuencia o un comando en la última de las líneas en blanco.

El comando SEQ, es utilizado después de crear el work file para secuenciar nuestro archivo.

Comandos CANDE.

Esta sección contiene los comandos CANDE para:

- Iniciación de tareas
- Creación de archivos de trabajo.
- Edición de texto
- Tareas varias

Los comandos CANDE se pueden abreviar para mayor comodidad. CANDE comprueba sólo los primeros cuatro caracteres de un comando; sin embargo, algunos comandos utilizados con frecuencia tienen abreviaturas más cortas.

Por ejemplo, el comando MAKE se puede abreviar como M y el comando SAVE se puede abreviar como SA.

A continuación, se muestran los comandos mas utilizados, para más información referirse al manual de referencia de operaciones CANDE.

Comando	Descripción
ACCESS	Se utiliza para interrogar o cambiar el código de acceso de una sesión.
ADD	El comando ADD es similar al comando COPY. La diferencia con COPY, es que si ya existe el archivo en la ruta destino no hace la copia y te informa del archivo ya existente.
BACKUPPROCESS	A través de esta utilidad, los archivos de respaldo (listados o reportes) se pueden copiar al disco, incluirlos en una terminal, imprimirlos o eliminarlos.
COMPILE	El comando COMPILE invoca a un compilador para compilar un programa y, si no hay errores de sintaxis en el programa, el resultado es un programa objeto.
COPY	La instrucción COPY copia archivos de disco desde discos, cintas o CD-ROM a discos, cintas o CD-ROM.
DELETE	El comando DELETE hace que las líneas especificadas <lista de rango de secuencia> se eliminen del work file.
EXECUTE	El comando EXECUTE provoca la ejecución de un programa objeto. RUN es sinónimo de EXECUTE.
FAMILY	El comando FAMILY permite interrogar y alterar las especificaciones de familia que se van a aplicar a la sesión actual.
FILE	El comando FILE enumera los nombres y tipos de archivos de su directorio en disco.
FIND	El comando FIND busca en un archivo (de forma predeterminada, el work file) las apariencias del texto de destino especificado.
GET	El comando GET designa un archivo guardado existente, o partes del mismo, como fuente de un nuevo work file.
HELP	Le permite acceder a cualquier libro de ayuda generado por la Utilidad de Ayuda.

LFILES	El comando LFILES ejecuta la utilidad SYSTEM/FILEDATA, que enumera los nombres y atributos de los archivos. Si no se especifica un nombre de archivo o de directorio, se enumeran los nombres y atributos de todos los archivos.
LIST	El comando LIST muestra el contenido de un work file u otro archivo especificado en la estación.
LOG	El comando LOG ejecuta la utilidad LOGANALYZER. Esta utilidad extrae información específica, según lo especificado por la opción LOGANALYZER, del archivo de registro del sistema y muestra la información en la terminal del usuario.
MAKE	Crea un nuevo work file.
MATCH	El comando MATCH compara dos archivos y determina las diferencias entre ellos. La salida que indica el resultado de la comparación se puede dirigir al terminal o a un nuevo archivo. El <archivo antiguo>, o partes de este, se compara con el <archivo nuevo>.
MOVE	El comando MOVE mueve líneas de un lugar a otro dentro del work file. Las líneas se mueven como un bloque contiguo y se les asignan nuevos números de secuencia.
NEXT	El comando NEXT inicia el modo de página y permite el movimiento a través del work file mientras está en modo de página
PAGE	El comando PAGE inicia el modo de página y le permite moverse por el work file. Puede especificar un rango de secuencia y un rango de columnas para ver partes de un work file.
REMOVE	El comando REMOVE elimina el archivo, archivos o los directorios de trabajo del disco.
REPLACE	El comando REPLACE busca en todo o parte del work file apariciones de texto de destino específico y reemplaza el texto de destino con texto nuevo. El sistema responde con un prompt (#) para indicar que se ha completado.
RUN	Los comandos RUN y EXECUTE son sinónimos.
SAME	El comando SAME inicia el modo de página o actualiza la página que se muestra actualmente en el work file.
SEQ	El comando SEQ hace que el sistema proporcione automáticamente un número de secuencia para cada línea que se ingresará en el archivo de trabajo.
START	El comando START indica al compilador WFL que procese el título del archivo especificado
TITLE	Los comandos TITLE y CHANGE son sinónimos. La única diferencia entre los comandos TITLE y CHANGE es que la palabra clave TO es opcional para el comando TITLE, pero necesaria para el comando CHANGE.
WHAT	El comando WHAT indica el estado del work file.

Comandos de control CANDE.

Los comandos de control te permiten:

- Proporcionan la capacidad de controlar e interrogar el entorno operativo.
- Debe estar precedido por el carácter de control de la estación (?).
- No se pueden poner en cola y se realizan o rechazan inmediatamente.

A continuación, se muestran algunos comandos de control, para mayor información, referirse al manual “CANDE Operations Reference Manual”.

Comando	Descripción
?AX	El comando ?AX pasa texto a un programa en ejecución en respuesta de una instrucción ACCEPT.
?BRK	Interrumpe el despliegue de los mensajes que están encolados para mostrarse en tu terminal.
?DS	Interrumpe la tarea a la que se hace referencia.
?LIBS	Muestra una lista de las bibliotecas congeladas bajo el código de usuario de la estación.
?J	Muestra todas las mezclas que se encuentran en ejecución para el USER.
?W	Muestra los programas que están es espera (por ejemplo un ACCEPT, un archivo).
?FA	Permite cambiar los atributos de un archivo que está siendo referenciada por una task que está en waiting, por ejemplo, para indicar la ruta correcta de un archivo ?FA TITLE = nombre_ruta.
?ON	Permite cambiar entre diferentes ventas de unisys, por ejemplo CANDE, LINEA, PROCESOS, LOGMIX, ODT...

WFL

WFL es un lenguaje de programación para escribir Jobs de batch. WFL incluye características para ejecutar programas, modificar su comportamiento y monitorear su progreso.

WFL también incluye características para el mantenimiento de archivos. La más importante de estas es la instrucción COPY, la cual puedes usar en un WFL job o ingresar como un comando en CANDE, MARC u ODT.

Para más información sobre WFL podemos consultar los siguientes manuales:

- WFL Made Simple.
- Work Flow Language (WFL) Programming Reference Manual.

Capacidades del WFL.

Los programas usualmente caen en una de dos categorías:

Programas de línea.

Aceptan transacciones en tiempo real desde un usuario y los resultados se muestran inmediatamente.

Programas Batch.

Se ejecutan en segundo plano y tiene una pequeña u nula interacción con el usuario. Los programas batch toman un largo tiempo de ejecución o consumen un mayor tiempo del procesador. Consecuentemente, debes tener cuidado con los programas batch para prevenir que entren en conflicto con otros o ralenticen programas de línea.

WFL se diseñó principalmente para administrar los programas batch.

Administración de Task.

Programas escritos en WFL son conocidos como WFL Jobs. Estos WFL Jobs existen principalmente para ejecutar programas escritos en otros lenguajes. A los programas iniciados desde un WFL job se les llama tasks.

Las características de administración de WFL task son:

- Ecuaciones de Task.
Opciones que modifican el comportamiento de las tasks.
- Ecuaciones de archivo.
Opciones que hacen que una task use diferentes archivos o use archivos de forma diferente a como normalmente lo haría.
- Parámetros.
Variables que sirven como entradas para la task.
- Estatus de consulta de Task.
Expresiones que puedes usar para revisar si una task se completó normalmente, antes de iniciar la siguiente task.

- Ejecución paralela.
La habilidad para ejecutar muchas tasks al mismo tiempo.

Administración del flujo de trabajo.

Las características de administración del flujo de trabajo de un WFL son:

- Prioridades.
Asignaciones que influyen la manera en que el sistema divide sus recursos entre Jobs y tasks.
- Colas de Job.
Estructuras que limitan cuantos Jobs de cierto tipo puede ejecutarse al mismo tiempo.
- Limites.
Asignaciones que limitan el uso de varios recursos para un job o sus tasks.
- Horas de inicio.
Asignaciones que especifican la fecha y hora de cuando un job será iniciado o que ejecutará el job de forma periódica.

Administración de archivos.

Las características de administración de archivos son:

- CHANGE y REMOVE.
Instrucciones que re titulan o eliminan archivos en disco.
- COPY.
Una instrucción que copia archivos de disco o respalda archivos de disco a cinta .
- PRINT.
Una instrucción que envía archivos para imprimir. Puedes configurar un numero de opciones para controlar el destino y apariencia de salida.

Compilando programas.

La instrucción WFL COMPILE te habilita específicamente cada una de las siguientes:

- Nombre del compilador.
Especifica el compilador a usar.
- Disposición del programa.
Guarda o ejecuta el programa compilado.
- Especificación de datos.
Suministra datos de entrada al compilador.
- Patches.
Especifica archivos patch para ser fusionados con el programa fuente.
- Ecuaciones de archivos o task.
Especifica ecuaciones de archivos o task para el compilador, por el programa compilado, o por ambos.

Reiniciando Tasks.

Los WFL Jobs son reiniciados automáticamente si son interrumpidos por un halt/load. El job típicamente se salta cualquier tasks que se completo antes del halt/load, y reinicia cualquier tasks que estuviera en progreso al tiempo de halt/load.

Capacidades de soporte.

WFL proporciona un numero de características que desempeñan un rol de asistencia para las características mayores previamente discutidas

- Flujo de control.
Son Instrucciones que ejecutan otras instrucciones condicionales, repetitivas, o de orden inusual.
- Variables y expresiones.
Constructos que almacenan o devuelven varios tipos de datos, incluyendo datos de tipo BOOLEAN, INTEGER, REAL y STRING.
- Comunicación con el operador.
Características que muestran información para operadores, o aceptan entradas desde operadores.
- Seguridad.
Características que controlan las restricciones de seguridad de los archivos y los privilegios de los Jobs.
- Expresiones de ambiente del sistema.
Características que recuperan la fecha y hora actual, o información acerca del sistema.

Limitaciones del WFL.

En particular WFL no puede:

- Leer o escribir en archivos.
- Consultar o actualizar bases de datos.
- Invocar procedimientos de libraries.

Aun con estas limitantes, dentro de su propósito especializado, WFL es un lenguaje excepcionalmente poderoso y flexible.

Creando un WFL job en CANDE.

Inicia una sesión en CANDE y ejecuta los siguientes pasos:

1. Usa el comando MAKE para crear un nuevo archivo de tipo JOB.
Por ejemplo:
`MAKE AVR/WFL/EJEMPLO JOB`
2. Edita el archivo en una de las siguientes formas:
 - Usando Workbench.
 - Usando CANDE

3. Guarda tu archivo.

Iniciando WFL Jobs en CANDE.

Inicia el WFL Job con el nombre específico del archivo.

Ejemplo:

```
START AVR/WFL/EJEMPLO ON PAGOS
```

En CANDE, puedes tener un archivo abierto para editar. Este archivo es conocido como workfile. Para iniciar el workfile, simplemente ingresa START sin especificar el nombre del archivo.

Seguimiento del progreso de un WFL en CANDE.

Ejemplos de mensajes del sistema mostrados por un job desde CANDE y su explicación.

- `START DBDATA/JOB`
- `#RUNNING 9609`
El compilador de WFL está compilando el job.
- `#JOB 9610 IN QUEUE 2`
El job esta siendo compilado, y ha sido asignado el número de mix 9610, y asigmo en job a cola 2.
- `#9610 BOJ DBDATA/JOB`
BOJ significa “Beginning of Job”. El WFL job ha sido seleccionado desde la cola del job y es ejecutado.
- `#9610\9611 BOT (ICS)OBJECT/DBDATA ON DEV`
BOT significa “Beginning of Task”. El WFL job ha iniciado el task (ICS)OBJECT/DBDATA ON DEV. La task es ejecutada con el numero de mix 9611.
- `#9610\9611 EOT (ICS) (ICS)OBJECT/DBDATA ON DEV`
EOT significa “End of Task”, e indica la terminación normal de la task (ICS)OBJECT/DBDATA. Nota que el usercode (ICS) aparece dos veces. El primer usercode mostrado es el atributo del usercode de la task. El segundo usercode mostrado es el usercode del archivo código objeto.
- `#9610\9610 EOJ (ICS) JOB DBDATA/JOB`
EOJ significa “End of Job”, e indica la terminación normal del WFL job.

Opciones de la instrucción START.

Iniciar un Job inmediatamente.

Inicia el Job llamado AVR/WFL/EJEMPLO

```
START AVR/WFL/EJEMPLO
```

Iniciar un job con parámetros.

```
START AVR/WFL/EJEMPLO (TRUE, 6, 5.3, "DIAG")
```

Pasa diferentes tipos de parámetros a un Job. Un parámetro es una pieza de información que el job puede leer dentro de una variable. El valor de un parámetro puede cambiar la forma de ejecución del job.

Puedes incluir uno o mas espacios entre cada parámetro. Estos espacios no son requeridos.

Iniciar un job con parámetros opcionales.

- Usando comas como marcadores de posición.
`START YORK/ULIB (, , , "DIAG")`
Pasa únicamente el cuarto parámetro al job que acepta cuatro parámetros opcionales. En este ejemplo, el cuarto parámetro es string. Las tres comas indican que los primeros tres parámetros son omitidos.
- Usando nombres de parámetros.
`START YORK/ULIB (RMODE := "DIAG")`
Pasa únicamente el parámetro nombrado RMODE a un job que acepta parámetros opcionales. La frase RMODE := indica cual de los parámetros opcionales será recibido.

Iniciar un job únicamente para revisar sintaxis.

```
START ADREP/UCON SYNTAX
```

Compila el Job y muestra mensajes indicando si hay cualquier error de sintaxis. Sin embargo, la opción SYNTAX previene que el Job actual se ejecute, incluso si no hay errores.

Iniciar un job en un horario específico.

```
START ADFAB/FILECON; STARTTIME = 17:00
```

Compila y coloca al job en la cola de jobs. El sistema no selecciona el job de la cola de Jobs para ejecutarlo hasta que sean las 5:00 PM.

Iniciar un job desde otro WFL job.

```
BEGIN JOB RUNNIT;  
RUN OBJECT/RECS;  
START (JAS)CLEANUP/JOB ON MUPACK;  
END JOB
```

Ejecuta un programa y después usa la instrucción START para iniciar otro WFL job. La instrucción START en un WFL job también puede permitir las varias opciones que han sido descritas previamente (SYNTAX, STARTTIME).

La instrucción START AND WAIT inicia un segundo job como una task dependiente dentro de un WFL job y espera a que la task dependiente se complete.

Iniciar otra copia de el mismo WFL job.

Es posible tener WFL Jobs que necesiten ejecutarse con regularidad. Para dichos trabajos, puede incluir una instrucción START que reinicie el job en una fecha u hora posterior.

Estructura de un Job.

La estructura básica de un WFL Job es simple y flexible. Los elementos básicos de un Wfl Jobs, incluyen:

- Estructura general
- Títulos del Job
- Formato del Job
- Comentarios
- Instrucciones incluidas de otro archivo

Estructura general.

```
BEGIN JOB;  
    <statements>;  
END OF JOB
```

Un WFL job inicia con la frase BEGIN JOB y termina con la frase END JOB. En medio, puedes incluir tantas instrucciones como quieras.

Especificar un nombre de Job.

```
BEGIN JOB <job name>;  
<statements>;  
END JOB
```

Es una buena idea incluir un nombre al job después de la frase BEGIN JOB. El nombre del job que elijas aparecerá en los mensajes del sistema y en las entradas de registro relacionadas con el job.

Algunos Jobs simples.

Ejecutando una Task.

```
BEGIN JOB RUNHEADERS;  
    RUN OBJECT/UTIL/NOTEHEADERS;  
END JOB
```

Este Job ejecuta un programa llamado OBJECT/UTIL/NOTEHEADERS.

Compilando un programa.

```
BEGIN JOB COMPILEPROG;  
    COMPILE (JONES)OBJECT/GEN/REPORT WITH ALGOL LIBRARY;  
    COMPILER FILE CARD(TITLE=(JONES)GEN/REPORT ON MYPACK) ;  
END JOB
```

Este Job compila un programa ALGOL desde el archivo fuente (JONES)GEN/REPORT ON MYPACK. El compilador crea un archivo de código objeto llamado (JONES)OBJECT/GEN/REPORT.

La línea que comienza con COMPILER FILE CARD se considera parte de la declaración COMPILE.

Copiando archivos.

```
BEGIN JOB COPYFEB;  
    COPY DATA/FEB AS ARCHIVE/DATA/FEB FROM RFPACK(PACK) TO  
    RECON(PACK) ;  
END JOB
```

Este Job copia el archivo DATA/FEB de la familia de discos RFPACK a la familia de discos RECON. La copia se crea con el nombre ARCHIVO/DATOS/FEB.

Usando múltiples instrucciones.

```
BEGIN JOB RUNHEADERS;  
    RUN OBJECT/UTIL/NOTEHEADERS;  
    COMPILE (JONES) OBJECT/ALGOL/TEST WITH ALGOL LIBRARY;  
    COMPILER FILE CARD(TITLE=ALGOL/TEST ON MYPACK) ;  
    COPY DATA/FEB AS ARCHIVE/DATA/FEB FROM RFPACK(PACK) TO  
    RECON(PACK) ;  
END JOB
```

Formato del Job.

Reglas generales.

- WFL no distingue entre mayúsculas y minúsculas; Las palabras clave y las variables pueden estar en mayúsculas, minúsculas o mixtas.
- Cada instrucción debe terminar con un punto y coma (;). Un signo de interrogación (?) en la columna uno se trata como un punto y coma. Sin embargo, también pueden aparecer puntos y coma en medio de algunas instrucciones, por ejemplo aquellas que incluyen ecuaciones de task o archivo, la parte principal de la instrucción termina con punto y coma, y cada ecuación también termina en punto y coma. Por ejemplo la instrucción COMPILE usada para compilar un programa.
- La mayoría de las instrucciones pueden comenzar en cualquier columna.
- La cantidad de espacios en blanco entre palabras puede variar.
- Una sola instrucción puede continuar en varias líneas. Pueden aparecer varias instrucciones en una sola línea, siempre que estén separadas por punto y coma.
- La sangría puede hacer que la estructura del trabajo sea más obvia para cualquiera que esté leyendo el trabajo.

Ejemplo de formato.

```
BEGIN JOB DAILYTOTALS;  
    COPY DAILY/TOTALS FROM DBCOM(PACK) TO ARCH(PACK) ;  
    IF TIMEDATE(DAY) = "SUNDAY" THEN  
        BEGIN  
            RUN OBJECT/WEEKTOTALS;  
            RUN OBJECT/RELAY;  
        END;  
END JOB
```

En este Job, la instrucción COPY y la instrucción IF tienen la misma sangría, porque ambas instrucciones están en el mismo nivel y se ejecutan una después de la otra.

Sin embargo, BEGIN... END tiene sangría para enfatizar que es parte de la declaración IF. Las instrucciones RUN tienen sangría para enfatizar que están agrupadas dentro de BEGIN... END. Este uso de sangría hace que la estructura general de la declaración de trabajo sea más fácil de leer.

Comentarios.

El signo de porcentaje (%) indica el inicio de un comentario. Un comentario puede aparecer solo en una línea o puede aparecer después de una declaración WFL. Los comentarios pueden incluir letras mayúsculas y minúsculas, números, espacios y signos de puntuación.

Los comentarios no se ejecutan ni se muestran al usuario. Solo son visibles para alguien que esté leyendo el archivo fuente de su trabajo. Los comentarios ayudan a otros programadores a comprender cómo actualizar o mejorar su Job. Los comentarios también le recuerdan por qué diseñó el trabajo de cierta manera.

Declaraciones.

Las declaraciones se usan para definir constantes, variables, subrutinas y especificaciones de datos globales. Las etiquetas no se declaran.

Sintaxis de las declaraciones.

La siguiente sintaxis representa las declaraciones disponibles en un trabajo WFL:

- Declaración constante: hace referencia a valores constantes por nombre en lugar de especificar los valores reales en todo el Job.
- Declaración booleana: declara una variable de tipo BOOLEAN. El valor inicial predeterminado de una variable booleana es FALSE.
- Declaración Integer: declara una variable de tipo INTEGER. El valor inicial predeterminado de una variable integer es 0.
- Declaración real: declara una variable de tipo REAL. El valor inicial predeterminado de una variable real es 0.
- Declaración String: declara una variable de tipo STRING. El valor inicial predeterminado de una variable string es una cadena nula ("").
- Declaración de archivos: define un archivo lógico con los atributos de archivo especificados.
- Declaración de tasks: declara variables de task, que pueden asociarse con tasks particulares en una declaración de inicio de task.
- Declaración de subrutinas: identifica una serie de instrucciones que se ejecutan cuando la subrutina es invocada por una instrucción de invocación de subrutina.
- Especificación de datos globales: contiene datos que pueden ser utilizados como entrada por programas iniciados en el trabajo. La especificación de datos globales se lee como si fuera un archivo de entrada de un lector de tarjetas y puede cerrarse y reabrirse o leerse por más de una task.

Todas las declaraciones de un job deben aparecer después de la lista de atributos del job y deben anteceder a cualquier instrucción. Todas las declaraciones en una subrutina deben estar después de la instrucción BEGIN de la subrutina y deben preceder a cualquier instrucción ejecutable en la subrutina.

Alcance de las declaraciones.

Las declaraciones pueden ocurrir a nivel de Job o dentro de subrutinas. Las declaraciones que ocurren a nivel de Job pueden definir:

- Variables globales
- Subrutinas
- Especificaciones de datos globales

Se puede hacer referencia a los elementos declarados a nivel de job en cualquier parte del job, incluso en cualquiera de las subrutinas.

Las variables y subrutinas declaradas dentro de una subrutina son locales para esa subrutina. Estas sólo se pueden utilizar en la subrutina en la que están declaradas y en subrutinas anidadas dentro de esa subrutina. Las variables locales se reinician cada vez que se invoca la subrutina. No se puede hacer referencia a una variable o subrutina antes de su declaración.

Inicialización de la variable.

El valor inicial de una variable puede ser especificado en la declaración; si esto no sucede, el valor por default para el tipo de variable es aplicado. La instrucción INITIALIZE asigna un valor de NEVERUSED al atributo task STATUS de la variable task especificada, y restaura los valores por default de todas los demás atributos de task y ecuaciones de archivo asociados con esa variable task.

Ejecutando Tasks.

Cuando ejecuta un programa desde WFL, el sistema crea algo llamado taks. Una task es una copia única de un programa que se ejecuta en la memoria del sistema. Si ejecuta el mismo programa varias veces, creará una nueva task cada vez.

Usando la instrucción RUN.

```
BEGIN JOB;  
    RUN (WESLEY)OBJECT/ALTCOM ON SERV;  
END JOB  
Ejecuta el programa (WESLEY)OBJECT/ALTCOM ON SERV.
```

Puntos clave para recordar sobre la instrucción RUN:

- La instrucción RUN puede iniciar programas escritos en casi cualquier lenguaje de programación.
- Sin embargo, la instrucción RUN no puede iniciar otro WFL job; debe utilizar la instrucción START para ese propósito.

- La instrucción RUN debe especificar el nombre del archivo de código objeto de un programa. El archivo de código objeto es la versión compilada del programa. Por el contrario, el archivo de código fuente almacena el programa tal como fue escrito originalmente por el programador.

Monitoreo mediante variables Task.

Puedes usar variables task para leer o escribir atributos de las tasks. Una variable task se relaciona con programa o una task en un WFL.

```
BEGIN JOB TVARDEMO (BOOLEAN DIAG);
  TASK T(PRIORITY = 55, USERCODE = JAMES/PW,FAMILY DISK = DISK ONLY);
  IF DIAG THEN
    T(SW1 = TRUE);
  RUN (PARTS)OBJECT/PROG [T];
  IF (T ISNT COMPLETEDOK) THEN
    DISPLAY "ERROR IN SUMMARY RUN";
END JOB
```

Este ejemplo usa la variable task T en las siguientes maneras:

- TASK T..
Declara la variable task y asigna valores para los atributos de task PRIORITY, USERCODE, y FAMILY.
- T(SW1=TRUE)
Asigna un valor al atributo task SW1 de la variable task T.
- [T]
Asigna la variable task para esta ejecución del programa OBJECT/PROG. Como resultado OBJECT/PROGRAM se ejecuta con los atributos task que fueron previamente asignados a T.
- T ISNT COMPLETEDOK
Evalua si la task no finalizo correctamente entonces muestra el mensaje “ERROR IN SUMMARY RUN”.

Reuso de variables tasks.

Inesperados efectos secundarios pueden resultar de reutilizar una variable task. Para prevenir estos problemas debes reinicializar la variable task antes de cada uso.

```
BEGIN JOB
  TASK T;
  RUN SYSTEM/CARDLINE [T];
  FILE CARD(KIND=DISK,TITLE=INPUT1);
  FILE LINE(KIND = DISK);
  INITIALIZE (T);
  RUN SYSTEM/CARDLINE [T];
  FILE CARD(KIND=DISK,TITLE=INPUT1);
END JOB
```


Est Job usa la instrucción INITIALIZE para prevenir efectos secundarios. Esta instrucción restaura todos los atributos de una variable task a sus defaults.

Otra opción para restaurar los atributos a sus defaults es:

```
T (STATUS = NEVERUSED);
```

Prevención de reinicios accidentales de tareas.

```
BEGIN JOB;  
  ON RESTART,  
    ABORT "Aborting job because of automatic restart";  
  RUN OBJECT/PAYROLL;  
END JOB
```

Ejecuta el programa OBJECT/PAYROLL no más de una vez. Si el sistema reinicia el job después de un halt/load, la instrucción ON RESTART invoca una instrucción ABORT para finalizar el job.

Reglas:

La instrucción ON RESTART está relacionada con la función de reinicio automático de los WFL Jobs. El sistema reinicia automáticamente cualquier WFL Job que se interrumpa por un halt/load. En este punto, todo lo que necesita saber es que pueden ocurrir reinicios automáticos si no toma medidas para evitarlos. Por ejemplo, supongamos que su WFL Job ejecuta una task que imprime cheques de pago. Si se produce un halt/load mientras la task se está ejecutando, después de halt/load el trabajo se reinicia y ejecuta la task nuevamente. El resultado podría ser que algunos cheques de pago se impriman dos veces.

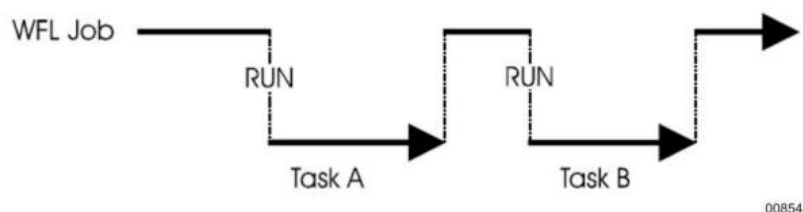
La instrucción ON RESTART especifica las acciones que se deben tomar si el job se reinicia después de una halt/load.

Ejecutar varias tareas al mismo tiempo.

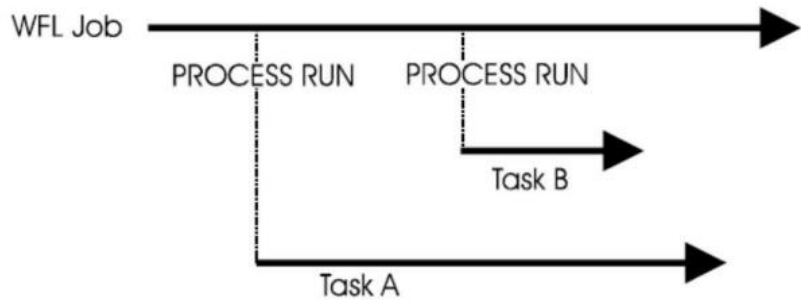
Si su WFL Job ejecuta varias tasks, puede elegir entre ejecutar las tasks una tras otra o ejecutarlas al mismo tiempo (en paralelo).

Flujo de control de tareas.

La siguiente figura muestra el flujo de control de un job que ejecuta task una tras otra. El Job utiliza una instrucción RUN para iniciar cada task. El job en sí está suspendido y no puede realizar ningún trabajo mientras se ejecuta cada task.



La siguiente figura muestra el flujo de control de un Job que ejecuta dos tasks al mismo tiempo. El Job utiliza una instrucción PROCESSRUN para iniciar cada task. El Job y las tasks se ejecutan en paralelo; es decir, todos pueden trabajar al mismo tiempo.



008549

La ejecución de tasks en paralelo puede permitir que el Job se complete más rápidamente.

Ejemplo:

```
BEGIN JOB FINANCIAL;
  PROCESS RUN OBJECT/INITIAL("WEEKEND");
  PROCESS COPY FINANCE/OUTPUT/= FROM ACCTS(PACK) TO HIST(PACK);
  REMOVE INIT/PAYABLE ON PACK;
END JOB
```

Pasar parámetros a una tarea.

```
RUN OBJECT/REORD(16.3, 4, TRUE, "MONTHLY");
```

Pasa cuatro parámetros al programa OBJECT/REORD. Los parámetros son valores que cambian la forma en que se ejecuta un programa. Puede pasar parámetros sólo a programas que estén diseñados para recibirlos. Los parámetros que pase deben coincidir con el número, tipo y orden de los parámetros en el programa.

La instrucción RUN puede incluir cuatro tipos de parámetros: Boolean, integer, real, y string.

Administración del flujo de control.

La gestión del flujo de trabajo consiste en controlar el orden, los tiempos y las prioridades de los distintos Jobs que se ejecutan en el sistema. El objetivo de la gestión del flujo de trabajo es garantizar que el sistema funcione de manera eficiente y dé preferencia a los elementos más urgentes de la carga de trabajo.

Start Time.

Puedes usar el atributo STARTTIME para especificar la fecha y la hora cuando un WFL Job será iniciado.

Hora de inicio en la lista de atributos del Job.

```
BEGIN JOB DAILY;
  STARTTIME = 18:00;
  RUN OBJECT/UPDATE;
END JOB
```

Provoca que el Job espere en una cola de job hasta al menos las 6:00 p.m.

Hora de inicio en la instrucción START.

```
START DAILY/JOB; STARTTIME = 14:00;
```

Provoca que el Job espere en una cola de job hasta al menos las 2:00 p.m.

La instrucción START sobre escribe cualquier start time especificado en la lista de atributos del Job.

Formatos del valor STARTTIME.

```
STARTTIME = 18:00;
```

```
STARTTIME = 18:00 ON 09/14/2001;
```

```
STARTTIME = +4:00;
```

```
STARTTIME = 15:00 ON +4;
```

Reglas para asignar valores a STARTTIME:

- Los tiempos son especificados usando un formato de 24 horas. Esto es, 18:00 significa 6:00 pm.
- Las fechas se especifican, primero con el mes, después el día, luego el año.
- Un signo mas (+) significa que las siguiente hora o fecha debe ser agregada a la hora y fecha actual. Por ejemplo, si la hora actual son las 13:00, entonces +4:00 dará la hora de 17:00.

Ejecutar un Job, de forma regular.

Es posible que tenga trabajos de WFL que deban ejecutarse con regularidad. Para dichos trabajos, puede incluir una declaración START que reinicie el trabajo en una fecha u hora posterior.

Iniciar Job todos los días.

```
BEGIN JOB AUDIT/JOB;  
  RUN OBJECT/DBAUDIT ON SERV;  
  START AUDIT/JOB; STARTTIME = 07:00 ON + 1;  
END JOB
```

Ejecuta el programa OBJECT/DBAUDIT diariamente. El Job primero ejecuta el programa. Luego, la declaración START inicia una nueva copia del Job con una HORA DE INICIO de las 7:00 a. m. del día siguiente.

Iniciar Job en días laborables.

```
BEGIN JOB AUDIT/JOB;  
  RUN OBJECT/DBAUDIT;  
  CASE TIMEDATE(DAY) OF  
  BEGIN  
    ("SUNDAY", "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY") :  
      START AUDIT/JOB; STARTTIME = 07:00 ON + 1;  
    ("FRIDAY") : START AUDIT/JOB; STARTTIME = 07:00 ON + 3;  
    ("SATURDAY") : START AUDIT/JOB; STARTTIME = 07:00 ON + 2;  
  END;  
END JOB
```

Ejecuta el programa todos los días de la semana (de lunes a viernes), pero no los fines de semana.

TIMEDATE(DAY)

Devuelve el nombre del día actual de la semana.

+ 1, + 2 o + 3

Indica el número de días que se deben omitir antes de que se inicie el trabajo.

Flujo de Control.

De forma predeterminada, el sistema ejecuta todas las declaraciones en un WFL Job una tras otra, en el orden exacto en que aparecen. Esta sección describe todas las formas en que puede cambiar el orden en que se ejecutan las instrucciones. En particular, esta sección describe los siguientes temas:

- **BEGIN...END:** Agrupa instrucciones.
- **IF:** Tomar una decisión.
- **CASE:** Elegir entre muchas alternativas.
- **GO TO:** Saltar a una instrucción a otra en el Job.
- **WHILE y DO:** Repetir declaraciones en bucles.
- **Subroutines:** Reutilización de instrucciones en varios puntos del Job.
- **WAIT:** Interrumpir el trabajo temporalmente.
- **ABORT y STOP:** Interrumpir el trabajo permanentemente.

BEGIN...END: Agrupa instrucciones.

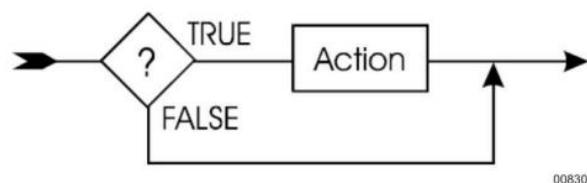
Agrupa dos o más instrucciones.

IF: Tomar una decisión.

Usando instrucciones IF, puedes:

- Haga una elección, basada de si un valor booleano particular es TRUE o FALSE.
- Proporcione una alternativa si el valor es FALSE.
- Pruebe varios valores booleanos con la instrucción IF anidándolos.

Instrucción IF simple.

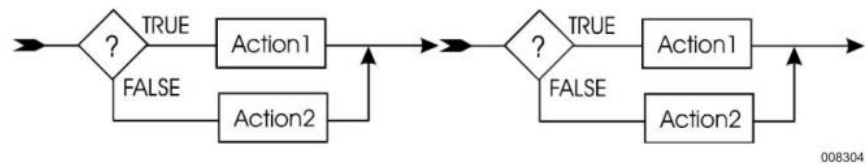


Ejecuta una acción si una condición de prueba es TRUE. De lo contrario, omite esa acción.

Reglas.

La instrucción IF ejecuta otra instrucción si una condición particular es verdadera. Usted especifica la condición que debe verificarse y la instrucción que debe ejecutarse si la condición es verdadera.

Instrucción IF ELSE.



Ejecuta una de dos acciones, dependiendo de si la condición de prueba es TRUE o FALSE.

Encadenamiento de la instrucción IF.

```
IF FILE TRANS/STATUS IS RESIDENT THEN
  RUN OBJECT/TRANS;
ELSE
  IF TRANS/BACKUP IS RESIDENT THEN
    BEGIN
      RUN OBJECT/TRANS;
      FILE SOURCE = TRANS/BACKUP;
      REMOVE TRANS/BACKUP;
    END;
  ELSE
    ABORT "NO FILE TRANS/STATUS OR TRANS/BACKUP";
```

Si el archivo TRANS/STATUS es residente, ejecuta una instrucción RUN. De lo contrario, comprueba si TRANS/BACKUP es residente y ejecuta una instrucción RUN y una instrucción REMOVE o aborta el Job.

Anidación de instrucciones IF.

Anidamiento incorrecto.

```
IF LENGTH (AXINPUT) GTR 0 THEN
  IF TAKE (AXINPUT,1) = "R" THEN
    RETRYVAL := TRUE;
ELSE
  BADINPUT := TRUE;
```

Una declaración IF ocurre dentro de otra declaración IF. A primera vista, podría parecer que la cláusula ELSE va con la declaración IF más externa. Sin embargo, para declaraciones IF anidadas, el compilador WFL hace coincidir la primera cláusula ELSE que encuentra con la declaración IF más interna. Como resultado, la asignación BADINPUT se ejecuta sólo si la longitud de AXINPUT es mayor que 0 y el primer carácter no es R.

Correcto anidamiento con BEGIN y END.

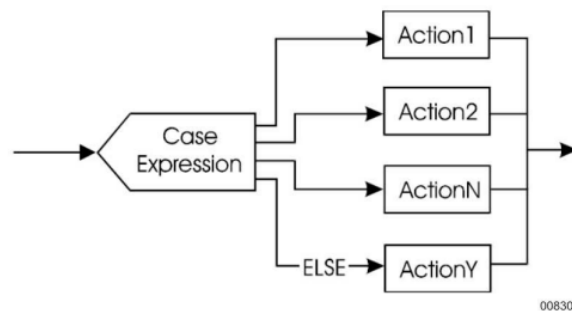
Utiliza una cláusula BEGIN... END para separar claramente la instrucción IF anidada de la instrucción IF externa.

Anidamiento correcto con una cláusula ELSE nula.

```
IF LENGTH (AXINPUT) <= 0 THEN
  IF TAKE (AXINPUT,1) = "R" THEN
    RETRYVAL := TRUE;
  ELSE
    ;
ELSE
  BADINPUT := TRUE;
```

Utiliza una cláusula ELSE nula para separar claramente la declaración IF anidada de la declaración IF externa. La asignación BADINPUT se ejecuta cada vez que AXINPUT tiene una longitud de 0.

CASE: Elegir entre muchas alternativas.



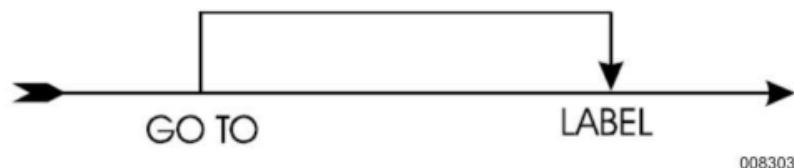
Evalúa una expresión case, y ejecuta una de las acciones específicas de la lista.

Ejemplo:

```
CASE STYPE OF
BEGIN
  ("DAILY"): RUN OBJECT/DAILY/UPDATE;
  ("WEEKLY"): RUN OBJECT/WEEKLY/UPDATE;
  ("SUMMARY"):
    BEGIN
      RUN OBJECT/REPORTSUM;
      PRINT REPORTSUM/DATA;
    END;
ELSE:
  ABORT "INVALID INPUT STRING; USE DAILY, WEEKLY, OR SUMMARY";
END;
```

Ejecuta varios programas. Por ejemplo, si STYPE tiene el valor DAILY, esta declaración ejecuta el programa OBJECT/DAILY/UPDATE.

GO TO: Saltar a una instrucción a otra en el Job.



Salta hacia adelante o hacia atrás en el Job hasta una instrucción con una etiqueta especificada.

Ejemplo:

```
RUN OBJECT/REPORT/PREP [T];  
IF T ISNT COMPLETEDOK THEN  
    GO TO LASTTASK;  
RUN OBJECT/REPORT/VALIDATE;  
LASTTASK:  
RUN OBJECT/REPORT/PRINT;
```

Omite la segunda de tres tareas si la primera no se completa normalmente.

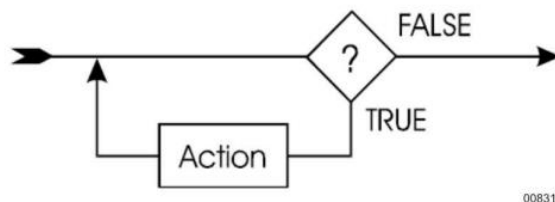
El uso de instrucciones GO TO es generalmente una señal de un mal estilo de programación. En general, cuantas más declaraciones GO TO utilice, más difícil será comprender o mantener el Job. Siempre que tenga la tentación de utilizar una instrucción GO TO, considere si sería más claro utilizar otra estructura de flujo de control. Por ejemplo, el ejemplo anterior podría expresarse más claramente como:

```
RUN OBJECT/REPORT/PREP [T];  
IF T IS COMPLETEDOK THEN  
    RUN OBJECT/REPORT/VALIDATE;  
RUN OBJECT/REPORT/PRINT;
```

WHILE y DO: Repetir declaraciones en bucles.

Si desea repetir una acción varias veces seguidas, puede utilizar una de las declaraciones de bucle: WHILE o DO.

Repetir una acción mientras se cumple una condición: Instrucción WHILE.



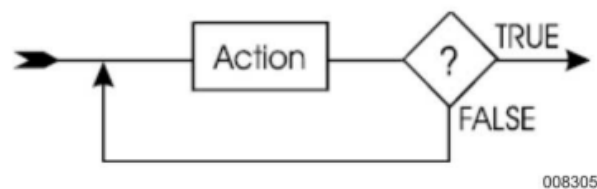
Si la condición de prueba es TRUE, ejecuta una acción repetidamente hasta que la condición de prueba sea FALSE. Si la condición de prueba es FALSE al inicio, la acción nunca se ejecuta.

Ejemplo:

```
WHILE LENGTH(INSTR) GEQ 4 DO
  BEGIN
    IF TAKE(INSTR, 4) = "ACCT" THEN
      BEGIN
        INSTR := DROP(INSTR, 4);
        FRONT := FRONT & "ARCH";
      END
    ELSE
      BEGIN
        FRONT := FRONT & TAKE(INSTR, 1);
        INSTR := DROP(INSTR, 1);
      END;
    END;
  END;
```

Busca a través de la cadena INSTR y reemplaza cualquier aparición de ACCT con ARCH. La instrucción WHILE prueba la longitud de INSTR al principio, porque no tiene sentido seguir comprobando si INSTR no es lo suficientemente largo para contener la frase ACCT.

Repetir una acción hasta que se cumpla una condición: Instrucción DO.



Ejecuta una acción repetidamente hasta que la condición de prueba sea TRUE. Si la condición de prueba es FALSE al inicio, la acción aún se ejecuta una vez.

Ejemplo:

```
DO
  BEGIN
    INITIALIZE(T);
    RUN OBJECT/VALIDATE [T];
  END
UNTIL T(TASKVALUE) GTR 0;
```

Ejecuta el programa OBJECT/VALIDATE hasta que devuelva un TASKVALUE mayor que 0.

La instrucción INITIALIZE se incluye porque la variable de tarea T se está reutilizando.

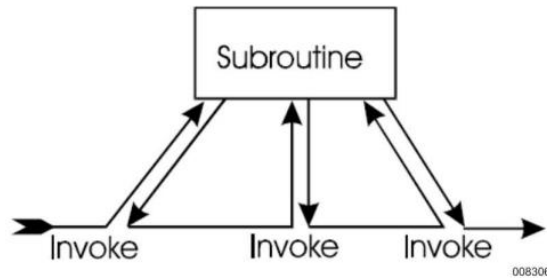
Repetir una acción un número fijo de veces.

```
I := 0;
DO
  BEGIN
    INITIALIZE(T);
    RUN OBJECT/VALIDATE [T];
    I := I + 1;
  END
UNTIL I GEQ 4;
```


Ejecuta un programa exactamente cuatro veces. La variable entera I se utiliza para realizar un seguimiento de cuántas veces se ha ejecutado el programa.

Este uso de la instrucción DO con una variable entera es el equivalente WFL más cercano a la instrucción FOR proporcionada por algunos otros lenguajes.

Subroutines: Reutilización de instrucciones en varios puntos del Job.



Cada vez que se invoca una subrutina, se ejecutan todas las instrucciones de la subrutina. Luego la ejecución continúa con la declaración que sigue a la declaración de invocación de subrutina. La subrutina se puede invocar desde muchos lugares diferentes dentro del trabajo WFL.

Ejemplo:

```
110 BEGIN JOB;
120
130 SUBROUTINE CLEANUP;
140 BEGIN
150 RUN OBJECT/UTIL/TRACK;
160 SECURITY TRACK/OUTPUT PUBLIC IO;
170 COPY TRACK/OUTPUT TO DEVPK(PACK);
180 END;
190
200 RUN OBJECT/VALIDATE;
210 CLEANUP;
220 RUN OBJECT/FORMAT;
230 CLEANUP;
240 RUN OBJECT/REPORT;
250 CLEANUP;
260
270 END JOB
```

Ejecuta tres programas. Después de ejecutar cada programa, ejecuta la subrutina CLEANUP.

Subrutina CLEANUP.

Comienza la declaración de la subrutina. Tenga en cuenta que un WFL Job consta de una serie de declaraciones, seguidas de una serie de instrucciones. Por lo tanto, la declaración de subrutina se encuentra antes en el Job que las declaraciones que invocan la subrutina.

Uso de parámetros.

Puede utilizar parámetros para pasar datos a una subrutina. Se puede hacer que el comportamiento de la subrutina varíe, dependiendo de los parámetros que le pase.

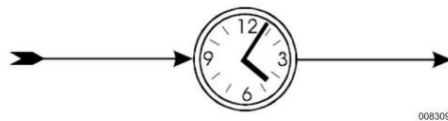
Salida prematura de una subrutina.

Con la instrucción RETURN provocamos la salida de una subrutina.

```
SUBROUTINE REPORTER;  
BEGIN  
    TASK T;  
    RUN OBJECT/PAYROLL/REPORT [T];  
    IF (T IS COMPLETEDOK) AND  
        (FILE PAYROLL/NEWMaster IS RESIDENT) THEN  
        RETURN;  
    RUN OBJECT/PAYROLL/RECOVER;  
END REPORTER;
```

En el ejemplo anterior se evalúa que OBJECT/PAYROLL/REPORT y que el archivo PAYROLL este residente, si se cumplen entonces se sale de la subrutina y ya no ejecuta OBJECT/PAYROLL/RECOVER.

WAIT: Interrumpir el trabajo temporalmente.



Detiene la ejecución del trabajo hasta que se cumpla una condición particular.

Esperando un archivo.

```
WAIT (FILE TEMP/INDEX/HTML ON SERV IS RESIDENT) ;
```

Espera hasta que el archivo TEMP/INDEX/HTML esté presente en la familia SERV.

Esperando un operador OK.

```
WAIT (OK) ;
```

Espera a que el operador introduzca un comando <mix number> OK.

Esperando por un período de tiempo.

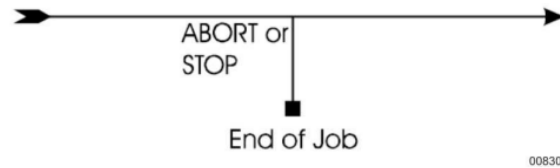
```
WAIT (120) ;
```

Espera 120 segundos (es decir, dos minutos).

Esperando TASKS.

La declaración WAIT tiene varias características que permiten que el trabajo espere tareas paralelas.

ABORT y STOP: Interrumpir el trabajo permanentemente.



Finaliza la ejecución del trabajo. Las declaraciones restantes en el trabajo nunca se ejecutan.

Las declaraciones ABORT y STOP difieren sólo en los mensajes de terminación que producen.

- Una instrucción ABORT hace que el sistema muestre un mensaje de terminación PDS, que es típico de una terminación anormal.
ABORT "Job abortado debido a la falla en OBJECT/PREPARE";
- Una declaración STOP hace que el sistema muestre un mensaje de terminación EOJ, que es típico de una terminación normal.
STOP " Job terminado debido a la falla en OBJECT/SORTDATA";

VARIABLES LOCALES Y GLOBALES

```
BEGIN JOB SCOPE;  
INTEGER NUM;
```

```
SUBROUTINE VALOR (INTEGER X);  
  BEGIN  
    INTEGER NUM;  
    X := 10;  
    NUM := 5;  
    DISPLAY "EL VALOR LOCAL " & STRING(NUM, *);  
  END;
```

```
VALOR(NUM);  
DISPLAY "EL VALOR GLOBAL " & STRING(NUM, *);  
END JOB
```

```
START AVR/WFL/SCOPE/24MTP002  
#RUNNING 5590  
#JOB 5591 IN QUEUE 0  
#5591 BOJ SCOPE  
#  
#5591 DISPLAY:EL VALOR LOCAL 5.  
#5591 DISPLAY:EL VALOR GLOBAL 10.  
#5591\5591 EOJ (S264) JOB SCOPE
```

Como se observa en el ejemplo, tenemos dos variables con el mismo nombre, una de manera local y la otra de manera global; aunque se llaman igual no son la misma, ya que en la subrutina se le dio un valor de 5 a la variable local y a la variable global no se le asignó un valor, es por ello por lo que al momento de hacer el display comprobamos el valor de cada una.

Comunicación con el operador.

DISPLAY:

Muestra una cadena de texto al operador como parte de los mensajes del sistema.

Ejemplos:

```
DISPLAY "Inicio de programa";  
DISPLAY "Se muestra valor de un integer ", & STRING(NUM,*);
```

Recibir información desde el operador.

ACCEPT: esperando un String.

```
STRING AXSTR;  
AXSTR := ACCEPT ("Enter AX BRIEF or AX LONG");  
IF AXSTR = "BRIEF" THEN ...
```

Suspende el Job y espera la entrada del operador. El Job aparece en la pantalla de entradas en espera con el mensaje

```
ACCEPT:Enter AX BRIEF or AX LONG.
```

Si el número de mix es 8364, el operador puede reiniciar el trabajo con un comando como el siguiente:

```
8364 AX BRIEF
```

Este comando asigna el valor "BRIEF" a la cadena AXSTR

DMSII

Un sistema de gestión de bases de datos (Database Management System) es un paquete de software especializado que se utiliza para describir una base de datos y mantener relaciones entre los distintos elementos de datos de la base de datos.

Enterprise Database Server centraliza las siguientes funciones de gestión de archivos:

- Descripción de los datos a gestionar.
- Rutinas especializadas que permiten que muchos usuarios accedan a la base de datos simultáneamente. Utilizando las rutinas, denominadas Accessroutines, se pueden agregar, modificar y eliminar registros de la base de datos. Las Accessroutines mantienen automáticamente índices de los datos que permiten una recuperación eficiente en respuesta a una amplia gama de consultas. Las Accessroutines también son responsables de la gestión del espacio en disco y packs.
- Lógica de recuperación que se invocará en caso de un mal funcionamiento del hardware o software que afecte a la base de datos.
- Aplicación de la seguridad para evitar el acceso no autorizado a la base de datos.
- Controles aplicados por el sistema para garantizar la integridad de los datos.
- Interfaces de programas de aplicación a través de las cuales los programas de usuario acceden a la base de datos.

El Enterprise Database Server contiene los siguientes componentes:

- DASDL
- Accessroutine
- Audit y Recovery
- Reorganización de la base de datos
- Archivo Control
- DMSUPPORT library

Descripción general de DASDL.

El lenguaje de definición de estructura y datos (Data and Structure Definition Language) se usa para describir las características físicas y lógicas de la base de datos, y los criterios que se utilizarán para garantizar la integridad y seguridad de los datos contenidos en ella.

Mediante la compilación del archivo DASDL se comprueba la sintaxis adecuada y se produce un archivo llamado DESCRIPTION/<nombre de la base>, el archivo DESCRIPTION es usado durante la compilación de todo el software de Enterprise Database Server y los programas de usuario que acceden a la base de datos

Ejemplo:

```
C AVR/DASDL/ABCAG04/24MTP002 AS $AVRDB
```

La instrucción del ejemplo compila el DASDL y nos genera el TAILORED y todos los archivos de la base de datos. AVR/DASDL/ABCAG04/24MTP002 es la ruta de nuestro DASDL fuente y AVRDB es el nombre que le asignamos a la base de datos.

Archivos TAILORED de la base de datos.

Durante la generación de bases de datos, el Enterprise Database Server crea varios archivos para uso exclusivo de la base de datos generada. Estos archivos se conocen como archivos tailored.

Los archivos TAILORED son:

- Archivo DESCRIPTION
DESCRIPTION/NAMEDB
- Archivo CONTROL
NAMEDB/CONTROL
- Library DMSUPPORT
DMSUPPORT/ NAMEDB
- Programa RECONSTRUCT
RECONSTRUCT/ NAMEDB

Accessroutines.

Accessroutines es una colección de rutinas especializadas que permiten que muchos usuarios accedan a la base de datos simultáneamente. Accessroutines es responsable de toda la gestión física y lógica de la base de datos.

Audit y Recovery.

Audit y recovery son usados para restaurar la base de datos seguido de un proceso o almacenamiento fallido.

Reorganización de la base de datos.

La reorganización de la base de datos permite cambios en el formato de una base de datos existente. La reorganización se puede utilizar para reordenar data sets, sets, y subsets; para generar nuevos sets o subsets automáticos para permitir un acceso más rápido a los data sets; y para cambiar formatos de registros agregando, eliminando o cambiando campos.

Archivo Control.

El archivo de control contiene marcas de fecha y hora (TIMESTAMP), que se utilizan para garantizar que se estén utilizando los archivos de base de datos actuales y que los programas de aplicación, las accessroutines y otro software de base de datos sean compatibles con los archivos de base de datos.

DMSUPPORT library.

La library DMSUPPORT contiene toda la información que conforma una base de datos. Hay una library DMSUPPORT por cada base de datos.

Descripción general de los componentes de Enterprise Database Server.

Una base de datos es una colección de datos ubicados en uno o más archivos. Una base de datos consta de los siguientes componentes principales:

- DATA SETS
- SETS
- SUBSETS
- Elementos de datos
- Datos globales

Data Sets

Un data set es una colección de registros relacionados almacenados en un archivo en un dispositivo de almacenamiento de acceso aleatorio. Contiene elementos de datos y tiene propiedades lógicas y físicas similares a los archivos. Sin embargo, a diferencia de los archivos convencionales, los data sets pueden contener otros data sets, sets y subsets.

Restart data set.

Es un tipo de data set que debes incluir si tu base de datos es auditada, este data set es requerido en los programas usuario al momento de entrar al estado de transacción con el BEGIN-TRANSACTION y el END-TRANSACTION.

Sets

Un set es una estructura que permite el acceso a todos los registros de un data set en alguna secuencia lógica. El set contiene una entrada para cada registro del data set. Cada entrada del set es un índice que localiza un registro del data set. Si se especifican elementos clave para el set, se accede a los registros del data set en función de estas claves; de lo contrario, se accede a los registros de forma secuencial. Se pueden declarar varios sets para un único data set, lo que permite acceder a los datos de un data set en varias secuencias diferentes.

Subsets

Un subset es similar a un set. A diferencia de un set, un subset sólo necesita hacer referencia a registros seleccionados en el data set.

Elementos de datos

Un elemento de datos es un campo en un registro de base de datos que se utiliza para contener una información individual.

Datos globales

Los elementos de datos que no forman parte de ningún conjunto de datos se denominan elementos de datos globales. Los elementos de datos globales generalmente consisten en información como totales de control, fechas de proceso y populations, que se aplican a toda la base de datos. Todos los elementos de datos globales se almacenan en un único registro.

Introducción de los componentes del lenguaje DASDL.

Conjunto de caracteres.

Caracteres EBCDIC usados en DASDL:

0 - 9	/
<espacio> o <vacio>	"
A - Z	=
,	\$
-	<
;	>
+	(
:)
*	^
.	

Aunque el lenguaje DASDL utiliza sólo los caracteres enumerados en la tabla de arriba, las cadenas pueden contener cualquier carácter EBCDIC.

Comentarios.

Un comentario es información descriptiva sobre la base de datos y puede tener hasta 255 caracteres. Puedes tener varias líneas de comentarios.

```
" —<string>— "
COMMENT — <any EBCDIC characters except semicolon (;)> ;
% —<string>
```

Si un comentario en uno de los 2 primeros formatos significa que es parte de la sintaxis de un componente DASDL, el comentario se incluye cuando un programa de usuario invoca la base de datos y se almacena en tablas de descripción con otros atributos de la base de datos. De lo contrario, el comentario es de referencia o definición y no se almacena en las tablas de descripción.

Expresión

Una expresión puede ser aritmética o booleana.

Expresión aritmética.

Una expresión aritmética produce uno o más valores numéricos. La expresión se forma combinando variables y constantes mediante operadores aritméticos.

Operador	Expresión
+	Suma
-	Resta
*	Multiplicación
/	División

Ejemplos de definición de una base de datos.

Ejemplo 1:

La base de datos en este ejemplo consiste en un único data set nombrado PERSONNEL. El data set contiene cuatro elementos de datos: NAME, EMPLOYEE-NO, DEPARTMENT, y PHONE

```
PERSONNEL DATA SET
(
NAME ALPHA(30);
EMPLOYEE-NO NUMBER(6);
DEPARTMENT ALPHA(20);
PHONE NUMBER(10);
);
```

Ejemplo 2:

La base de datos consiste en un único data set y dos sets. El data set PERSONNEL puede ser accedido por NAME usando el set FIND-BY-NAME, y se puede acceder por EMPLOYEE-NO usando el set FIND-BY-NUMBER.

```
PERSONNEL DATA SET
(
NAME ALPHA(30);
EMPLOYEE-NO NUMBER(6);
DEPARTMENT ALPHA(20);
PHONE NUMBER(10);
);
FIND-BY-NAME SET OF PERSONNEL KEY IS NAME;
FIND-BY-NUMBER SET OF PERSONNEL KEY IS EMPLOYEE-NO;
```

Ejemplo 3:

En este ejemplo, la base de datos utiliza la opción STATISTICS para medir el uso de la base de datos. El parámetro ALLOWEDCORE controla cuántos núcleos se puede utilizar para los búffers de la base de datos. El elemento global POP-OF-PERSONNEL contiene un recuento del número de registros en el data set PERSONNEL.

```
OPTIONS (STATISTICS);
PARAMETERS (ALLOWEDCORE = 18000);
POP-OF-PERSONNEL POPULATION(9999) OF PERSONNEL;
PERSONNEL DATA SET
(
NAME ALPHA(30);
EMPLOYEE-NO NUMBER(6);
DEPARTMENT ALPHA(20);
PHONE NUMBER(10);
);
```

Ejemplo 4:

En este ejemplo, la base de datos incluye un restart data set que contiene información de reinicio para programas de aplicaciones de usuario. La especificación física para el data set PERSONNEL asigna el data set a pack y le asigna dos buffers.

```
OPTIONS (AUDIT);
RESTARTDS RESTART DATA SET
(
  R-PROGRAMNAME ALPHA(25);
  R-USERINFO ALPHA(250);
);
PERSONNEL DATA SET
(
  NAME ALPHA(30);
  EMPLOYEE-NO NUMBER(6);
  DEPARTMENT ALPHA(20);
  PHONE NUMBER(10);
);
FIND-BY-NAME SET OF PERSONNEL KEY IS NAME;
FIND-BY-NUMBER SET OF PERSONNEL KEY IS EMPLOYEE-NO;
PERSONNEL (KIND = PACK, BUFFERS = 2);
```

Instrucción Initialize.

En el DASDL la instrucción Initialize hace que DMUTILITY inicialice todos los archivos de la base de datos. La instrucción Initialize debe usarse sólo al crear una nueva base de datos. No debería estar presente cuando se modifica la descripción de una base de datos existente, para actualizar utilizamos la instrucción Update.

Ejemplo:

```
INITIALIZE;
D DATA SET
(
  K NUMBER (4);
);
S SET OF D KEY K;
```

Sets y Subsets.

Sets y subsets ejecutan dos funciones:

- Permiten recuperar de manera ordenada y rápidamente los registros del data set.
- Representan relaciones entre los registros del data set.

Tres métodos son usados para recuperar registros. El método para usar depende de si la estructura es un set o un subset.

- Sin llave
- Llave aleatoria
- Llave secuencial

Una o mas llaves de acceso se deben especificar cuando un set o subset utiliza acceso aleatorio o secuencial ordenado por llaves. Las llaves deben ser variables que se encuentran dentro del set o subset que se está usando, las llaves se definen usando el DASDL.

Nombre del Data set, Set y Subset .

El nombre del Data set, Set y Subset es un identificador que nombra al Data set, Set y Subset, respectivamente. El nombre de estos no puede exceder 17 caracteres.

Clausula Key.

La cláusula key identifica la llave o llaves del set. Todas las llaves deben ser variables o grupos de variables en el data set al que hace referencia el set.

Cuando aparecen varios elementos llave, los elementos llave se enumeran de izquierda a derecha en orden de importancia.

Ejemplo de declaración Set.

```

EMPLOYEE DATA SET
(
    NAME ALPHA(20);
    EMP-NO NUMBER(6);
    DEPT FIELD(8);
    TITLE ALPHA(15);
);

BY-NAME SET OF EMPLOYEE
    KEY IS NAME,
    INDEX SEQUENTIAL;

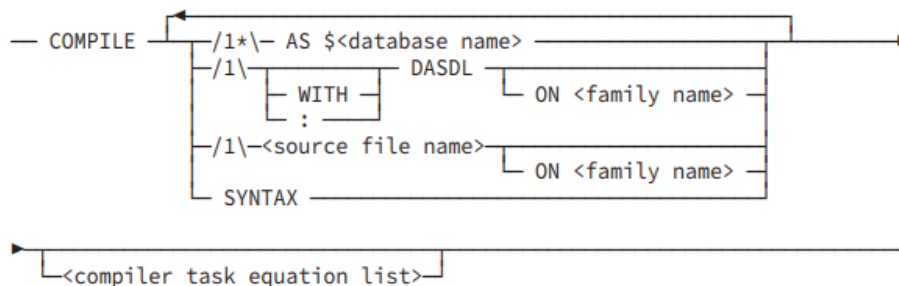
BY-EMP-NO SET OF EMPLOYEE
    KEY IS EMP-NO,
    INDEX RANDOM;

```

BY-NAME ordena los registros del data set EMPLOYEE usando el elemento llave NAME.

BY-EMP-NO ordena los registros en secuencia ascendente usando el elemento llave EMP-NO.

Iniciar compilación DASDL usando CANDE.



Ejemplo del comando para compilar DASDL es:

```
COMPILE AS $DBNAME
```

Esta forma de compilar funciona cuando tenemos abierto nuestro DASDL en workfile.

Una vez compilado nuestro DASDL, la base de datos habrá sido creada junto con los archivos necesarios para su administración.

Instrucciones de control del compilador.

Las instrucciones de control del compilador controlan cómo el compilador DASDL procesa la entrada fuente, maneja los errores y genera la salida. Las instrucciones de control del compilador se identifican con un signo de pesos (\$) en la columna 1 o un espacio en blanco en la columna 1 y un signo de pesos en la columna 2.

Manejo de la base de datos con INQUIRY.

El INQUIRY es un programa en línea interactivo que se puede usar para examinar o modificar información de una base de datos, el INQUIRY tiene un pequeño número de instrucciones que pueden combinarse para realizar operaciones complejas.

INQUIRY proporciona entre otras capacidades las siguientes:

- Examinar información de la base de datos. El INQUIRY es capaz de acceder a la información en cualquier parte de la base de datos, incluso si se tienen que realizar búsquedas secuenciales. Siempre que sea posible el INQUIRY usará los SETS para acceder a la información de la base de datos.
- Modificación de la base de datos. Podemos usar el INQUIRY para modificar, crear o eliminar registros de las bases de datos.
- Generación de reportes. El INQUIRY permite generar reportes simples con la información de la base de datos.

Crear INQUIRY desde CANDE.

El INQUIRY se genera mediante el siguiente procedimiento:

1. El usuario ejecuta el programa BUILDINQ ingresando la siguiente instrucción:

```
RUN $SYSTEM/BUILDINQ
```

2. BUILDINQ responde con el siguiente mensaje en pantalla:

```
WHAT DATABASE?
```

El usuario ingresa el nombre de la base de datos.

```
NAMEDB
```

Si BUILDINQ no puede localizar el archivo DESCRIPCIÓN/NAMEDB o si el nombre de la base de datos especificada es incorrecto, BUILDINQ muestra el siguiente mensaje y luego finaliza:

```
***** ERROR ***** NO SUCH DATABASE
```

3. Si el archivo DESCRIPTION/NAMEDB y el archivo DATABASE/PROPERTIES están presentes, BUILDINQ muestra el siguiente mensaje:

```
WHICH SOFTWARE?
```

```
1 DMINQUIRY
```

2 DMINTERPRETER

El usuario de ingresar la opción 1 .

4. BUILDINQ muestra el siguiente mensaje:

ALLOW INQUIRY ONLY (YES OR NO) ?

El usuario ingresa YES para permitir únicamente consulta e ingresa NO para poder utilizar los comandos UPDATE, CREATE, y DELETE. Si se ingresa NO, BUILDINQ muestra el siguiente mensaje:

ALLOW UPDATE (YES OR NO) ?

YES permite UPDATE; NO, no permite la UPDATE . BUILDINQ procede a preguntar:

ALLOW CREATE (YES OR NO) ?

YES permite CREATE; NO, no permite CREATE . BUILDINQ procede a preguntar:

ALLOW DELETE (YES OR NO) ?

YES permite DELETE; NO, no permite DELETE .

5. BUILDINQ procede a preguntar:

WHICH OPTION

1 TOTAL DATABASE

2 SELECTED DATASETS

El usuario ingresa 1 o 2 para especificar qué parte de la base de datos será accesible para la consulta.

6. Cuando se ha seleccionado la opción deseada, BUILDINQ pregunta:

INQUIRY PROGRAM NAME (NULL FOR DEFAULT) ?

Si se transmite una entrada nula (ctrl + enter), en respuesta al mensaje, se utiliza el siguiente nombre predeterminado para el programa de consulta:

OBJETO/CONSULTA/<nombre de la consulta>

7. Cuando el programa de INQUIRY ha sido nombrado, BUILDINQ pregunta:

WHAT QUEUE (NULL FOR DEFAULT) ?

Si se ingresa una entrada nula (ctrl + enter), la consulta se compila en la cola predeterminada del sistema; de lo contrario, la consulta se compila en la cola especificada por el usuario .

8. Cuando la cola ha sido asignada, BUILDINQ pregunta:

DEFINITION FILE NAME (NULL FOR DEFAULT) ?

Si se ingresa una entrada nula, el archivo de definiciones predeterminado tiene el siguiente formato: (<usercode>)DEFINITIONS/<inquiry name>

Comandos de INQUIRY.

Los comandos de consulta UPDATE, CREATE, y DELETE se pueden utilizar para modificar el contenido de una base de datos. A continuación se muestran algunos comandos:

UPDATE.

El comando UPDATE cambia los valores de los elementos en el registro seleccionado.

CREATE.

Crea un nuevo registro para ser agregado a un data set.

DELETE.

Elimina el registro seleccionado actualmente en el data set.

DISPLAY.

Muestra elementos de un registro seleccionado.

SELECT.

Localiza registros que satisfacen las condiciones de selección.

SHOW

Muestra partes específicas de una descripción de base de datos, contenidos del búfer actual, elementos definidos, elementos virtuales o especificaciones de subconjunto generado.

Ejecutar INQUIRY.

Una vez creado nuestro INQUIRY, pasamos a ejecutarlo con la instrucción:

R INQUIRY/NAMEDB

Ejemplo:

```
R INQUIRY/AVRPOPDB
#RUNNING 4185
#?
AVRPOPDB INQUIRY SSR 62.0 (62.080.0001)
#INITIALIZING INQUIRY
#AVRPOPDB READY
T F T
#
SELECT PERSONNEL
#
D ALL
  NAME=JONATAN                                EMPLOYEE-NO=123456
  DEPARTMENT=--  PHONE=--
#
S NAME = "AMERICA"
#
D ALL
  NAME=AMERICA                                EMPLOYEE-NO=123456
  DEPARTMENT=--  PHONE=--
```



```

SELECT PERSONNEL
#
D ALL
    NAME=JONATAN                      EMPLOYEE-NO=123456
    DEPARTMENT=CAPACITACION           PHONE=5512345678

```

```

#
UPDATE NAME = "JONATAN HERNANDEZ"
SELECTED RECORD OF PERSONNEL UPDATED
#
D ALL
    NAME=JONATAN HERNANDEZ            EMPLOYEE-NO=123456
    DEPARTMENT=CAPACITACION           PHONE=5512345678

```

```

CREATE NAME = "USER PRUEBA", EMPLOYEE-NO=999999, DEPARTMENT="NONE"
RECORD OF PERSONNEL CREATED
#
D ALL
    NAME=USER PRUEBA                  EMPLOYEE-NO=999999
    DEPARTMENT=NONE                   PHONE=--

```

Manejo de la base de datos con un programa COBOL74.

Otra manera de gestionar los registros de nuestra base de datos es a través de un programa COBOL, a continuación, se muestran ejemplos para los comandos:

CREATE.

Inicializa el área de trabajo del usuario para el registro del data set.

DELETE.

Borra un registro de la base de datos, remueve del data set, sets, subsets automáticos, pero no de los subsets manuales.

Si utilizamos una expresión de selección en el DELETE, tu programa primero buscara el registro, lo leerá, lo bloqueara (lock), para después eliminarlo.

FIND.

Lee un registro y lo transfiere al área de trabajo del usuario asociada.

STORE.

Guarda en el data set un registro nuevo o las modificaciones hechas a un registro ya existente.

Antes de hacer un STORE debes haber hecho un CREATE, RECREATE, o un LOCK /MODIFY.

LOCK.

Encuentra un registro de la misma forma que hace un FIND, luego lo bloquea para prevenir que otros usuarios lo puedan modificar.

Ningún otro usuario podrá bloquear el registro una vez que haya sido bloqueado, por lo tanto, debemos liberar el registro en cuanto ya no se requiera tenerlo bloqueado. Podemos desbloquear el registro de manera explícita usando la instrucción FREE o de manera implícita ejecutando una instrucción LOCK, FIND, DELETE, CREATE o RECREATE sobre el mismo data set.

BEGIN-TRANSACTION.

Coloca un programa en estado de transacción. Esta opción es utilizada únicamente con bases de datos auditadas. Cualquier intento para modificar una base de datos auditada cuando el programa no esta en estado de transacción resulta en un error.

Las siguientes instrucciones modifican una base de datos:

- ASSIGN
- DELETE
- GENERATE
- INSERT
- REMOVE
- STORE

END-TRANSACTION.

Termina el estado de transacción de un programa.

Ejemplo:

Teniendo en mente que tenemos una base de datos llamada EMPLEADOSDB, con un data set llamado REGISTROEMPLEADO con los campos de NOMINA, NOMBRE, APELLIDO-PATERO, APELLIDO-MATERO, TELEFONO, DIRECCION, DEPARTAMENTO y SUELDO y un set llamado EMPNOMINA que utilizara como llave el campo NOMINA Y un restart data set llamado RESTARTEMPLEADOS veamos el siguiente ejemplo:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
DATA-BASE SECTION.
DB EMPLEADOSDB ALL.
WORKING-STORAGE SECTION.
77 WS-MENU-OPCION  PIC X.
    88 OPC-VALIDA      VALUE "A" THRU "C", "M".
    88 ALTA             VALUE "A".
    88 BAJA            VALUE "B".
    88 CONSULTA        VALUE "C".
    88 MODIFICACION    VALUE "M".
    88 ALTA            VALUE "S".
77 WS-EXISTE-EMPLEADO  PIC 9.
77 WS-NOMINA           PIC 9(05).
77 WS-NOMBRE           PIC X(20).
77 WS-APELLIDO-PATERO  PIC X(15).
77 WS-APELLIDO-MATERO  PIC X(15).
77 WS-TELEFONO         PIC 9(10).
77 WS-DIRECCION        PIC 9(50).
77 WS-DEPARTAMENTO     PIC 9(15).
77 WS-SUELDO           PIC 9(10).
77 WS-RESP             PIC X(02).
PROCEDURE DIVISION.
RUTINA-PRINCIPAL.
    PERFORM 100-INICIO.
    PERFORM 200-PROCESO.
    PERFORM 300-FIN.
100-INICIO.
    OPEN UPDATE EMPLEADOSDB.
200-PROCESO.
    PERFORM MENU UNTIL SALIR.
300-FIN.
    CLOSE EMPLEADOSDB.
    STOP RUN.
```

```

MENU.
    DISPLAY "BASE DE DATOS DE EMPLEADOS TESI".
    DISPLAY "¿QUE DESEAS HACER?".
    DISPLAY "A = ALTA DE EMPLEADO".
    DISPLAY "B = BAJA DE EMPLEADO".
    DISPLAY "C = CONSULTA DE DATOS DEL EMPLEADO".
    DISPLAY "M = MODIFICACION DE DATOS DEL EMPLEADO".
    DISPLAY "S = SALIR".
    ACCEPT WS-MENU-OPCION.
    IF OPC-VALIDA
        DISPLAY "INGRESA NUMERO DE NOMIDA: "
        ACCEPT WS-NOMINA
        PERFORM BUSCA-NOMINA
        IF ALTA
            PERFORM PROCESO-ALTA
        ELSE
            IF BAJA
                PERFORM PROCESO-BAJA
            ELSE
                IF CONSULTA
                    PERFORM PROCESO-CONSULTA
                ELSE
                    IF MODIFICACION
                        PERFORM PROCESO-MODIFICACION
                    ELSE
                        NEXT SENTENCE
        ELSE
            IF NOT SALIR
                DISPLAY "OPCION INVALIDA".
BUSCA-NOMINA.
    MOVE 1 TO WS-EXISTE-EMPLEADO.
    FIND REGISTROEMPLEADO VIA EMPNOMINA AT NUM-NOMINA = WS-NUM-NOMINA
    ON EXCEPTION
        IF DMSTATUS (NOTFOUND)
            MOVE 0 TO WS-EXISTE-EMPLEADO
        ELSE
            PERFORM ERROR-DMSII.
PROCESO-ALTA.
    IF WS-EXISTE-EMPLEADO = 1
        DISPLAY "YA EXISTE EMPLEADO CON ESA NOMINA"
    ELSE
        PERFORM ALTA-EMPLEADO.
ALTA-EMPLEADO.
    DISPLAY "NOMBRE: "
    ACCEPT WS-NOMBRE
    DISPLAY "APELLIDO PATERNO: "
    ACCEPT WS-APELLIDO-PATERNO
    DISPLAY "APELLIDO MATERNO: "

```

```

ACCEPT WS-APELLIDO-MATERNO
CREATE EMPNOMINA
MOVE WS-NUM-NOMINA          TO NUM-NOMINA
MOVE WS-NOMBRE              TO NOMBRE
MOVE WS-APELLIDO-PATERNO TO APELLIDO-PATERNO
MOVE WS-APELLIDO-MATERNO TO APELLIDO-MATERNO
BEGIN-TRANSACTION AUDIT RESTARTEMPLADOS
    STORE REGISTROEMPLEADO
    ON EXCEPTION
        IF DMSTATUS(DUPPLICATES)
            DISPLAY "STORE DUPLICADO"
        ELSE
            PERFORM ERROR-DMSII.
END-TRANSACTION AUDIT RESTARTEMPLADOS
DISPLAY "ALTA EMPLEADO, OK".
PROCESO-BAJA.

IF WS-EXISTE-EMPLEADO = 1
    DISPLAY "NUMERO DE NOMINA: " NOMINA
    DISPLAY "NOMBRE: " NOMBRE
    DISPLAY "APELLIDO PATERNO: " APELLIDO-PATERNO
    DISPLAY "APELLIDO MATERNO: " APELLIDO-MATERNO
    DISPLAY "DESEA ELIMINAR EL REGISTRO DEL EMPLEADO? SI/NO"
    ACCEP WS-RESP
    IF RESP = "SI"
        PERFORM BAJA-EMPLEADO
    ELSE
        DISPLAY "BAJA CANCELADA".
BAJA-EMPLEADO.

BEGIN-TRANSACTION AUDIT RESTARTEMPLADOS
    DELETE REGISTROEMPLEADOS VIA EMPNOMINA AT NOMINA = WS-NOMINA
    ON EXCEPTION
        IF DMSTATUS(NOTFOUND)
            DISPLAY "REGISTRO NO EXISTE"
        ELSE
            PERFORM ERROR-DMSII.
END-TRANSACTION AUDIT RESTARTEMPLADOS
DISPLAY "BAJA EMPLEADO, OK".
PROCESO-CONSULTA.

IF WS-EXISTE-EMPLEADO = 1
    DISPLAY "NUMERO DE NOMINA: " NOMINA
    DISPLAY "NOMBRE: " NOMBRE
    DISPLAY "APELLIDO PATERNO: " APELLIDO-PATERNO
    DISPLAY "APELLIDO MATERNO: " APELLIDO-MATERNO
    ELSE
        DISPLAY "NO EXISTE REGISTRO DE EMPLEADO".

```

PROCESO-MODIFICACION.

```
IF WS-EXISTE-EMPLEADO = 1
    DISPLAY "NUMERO DE NOMINA: " NOMINA
    DISPLAY "NOMBRE: " NOMBRE
    DISPLAY "APELLIDO PATERNO: " APELLIDO-PATERNO
    DISPLAY "APELLIDO MATERNO: " APELLIDO-MATERNO
    DISPLAY "TELEFONO: " TELEFONO
    DISPLAY "DIRECCION: " DIRECCION
    DISPLAY "DESEA MODIFICAR INFORMACION DE EMPLEADO? SI/NO"
    ACCEPT WS-RESP
    IF RESP = "SI"
        PERFORM LOCK-REGISTROEMPLEADOS
        PERFORM MODIFICACION-EMPLEADO
        DISPLAY "MODIFICACION EN DATOS DE EMPLEADO, OK"
    ELSE
        DISPLAY "NO EXISTE REGISTRO DE EMPLEADO".
LOCK-REGISTROEMPLEADOS.
MOVE 1 TO WS-EXISTE-EMPLEADO.
LOCK REGISTRO EMPLEADOS VIA EMPNOMINA AT NOMINA= WS-NOMINA
ON EXCEPTION
    IF DMSTATUS (NOTFOUND)
        DISPLAY "YA NO SE ENCONTRO REGISTRO"
    ELSE
        PERFORM ERROR-DMSII.
MODIFICACION-EMPLEADO.

    DISPLAY "NOMBRE: " NOMBRE
    ACCEPT NOMBRE
    DISPLAY "APELLIDO PATERNO: " APELLIDO-PATERNO
    ACCEPT APELLIDO-PATERNO
    DISPLAY "APELLIDO MATERNO: " APELLIDO-MATERNO
    ACCEPT WS-APELLIDO-MATERNO
    DISPLAY "TELEFONO: " TELEFONO
    ACCEPT TELEFONO
    DISPLAY "DIRECCION: " DIRECCION
    ACCEPT DIRECCION
    BEGIN-TRANSACTION AUDIT RESTARTEMPLEADOS
        STORE REGISTROEMPLEADO
        ON EXCEPTION
            IF DMSTATUS (DUPLICATES)
                DISPLAY "STORE DUPLICADO"
            ELSE
                PERFORM ERROR-DMSII.
    END-TRANSACTION AUDIT RESTARTEMPLEADOS
ERROR-DMSII.
```

CALL SYSTEM DMTERMINATE.