UNIVERSIDAD AUTONOMA DEL ESTADO DE MÉXICO

INGENIERIA EN COMPUTACION

MONOGRAFIA DE LENGUAJE DE CONTROL DE TRABAJOS (JCL)

TESINA PARA OBTENER EL TITULO DE INGENIERIA EN COMPUTACION

P R E S E N T A:

DIANA NUÑEZ CARRILLO

DIRECTOR DE TESINA:

DR. JOEL AYALA DE LA VEGA

REVISORES:

DR. OZIEL LUGO ESPINOSA

DR. ALFONSO ZARCO HIDALGO

Texcoco Estado de México 2014

Texcoco, México, a 4 de Febrero del 2014

M. EN C. JUAN MANUEL MUÑOZ ARAUJO
SUBDIRECTOR ACADEMICO DEL
CENTRO UNIVERSITARIO UAEM
TEXCOCO.

PRESENTE:

AT'N M. EN P.P. ANTONIO INOUE CERVANTES

RESPONSABLE DEL DEPARTAMENTO DE TITULACIÓN

Con base a las revisiones efectuadas al trabajo escrito titulado "Monografía de Lenguaje de Control de Trabajos (JCL)" que para obtener el título de Licenciada en Ingeniería en Computación presenta la sustentante C. Diana Núñez Carrillo, con número de cuenta 0222654 respectivamente, se concluye que cumple con los requisitos teórico - metodológicos necesarios para su aprobación, pudiendo continuar con la etapa de digitalización del trabajo escrito.

ATENTAMENTE

Dr. en C. Oziel Lugo Espinosa

Dr en C:. Alfonso Zarco Hidalgo

Dr. en Ed. Joel Ayala de la Vega

c.p.p. Sustentante: Diana Núñez Carrillo
c.p.p. Director.- Dr. Joel Ayala de la Vega
c.p.p. Titulación.- M. en P.P. Antonio Inoue Cervantes

TITULACION
RECIBIDO
Por INOUE
Texcoco, Méx., a 04 de 02 del 2014

RESUMEN

La importancia de los sistemas operativos nace en los 50's, cuando se hizo evidente que el operar una computadora por medio de enchufes (primera generación) y luego por medio del trabajo en lote (segunda generación) se podía mejorar, se comenzó a ver que las tareas que realizaba el operador de la computadora podían plasmarse en un programa que a través del tiempo resultó ser un sistema operativo ya que tenía una enorme complejidad al administrar los recursos de una computadora. Gracias a esto se fueron desarrollando varios Sistemas Operativos hasta llegar al MVS que se usa actualmente.

El Sistema Operativo MVS se ayuda de JES que es el encargado de gestionar la entrada de trabajos Batch, una de las principales funciones de JES es preparar los trabajos que se le envían para su ejecución, interpretando las sentencias de control que componen cada una de las cadenas o JOB's. Estas cadenas son escritas en un lenguaje especial llamado Lenguaje de Control de Trabajos (JCL)

Desde sus inicios JCL ha sido implementado por muchas empresas para controlar procesos pesados que contienen tareas repetitivas y una gran carga de información. Se trata de un tipo especial de lenguaje de programación usado para dotar de instrucciones al Sistema Operativo permitiendo la coordinación y administración de los recursos del sistema. Permite identificar a los usuarios y sus trabajos, así como especificar los recursos requeridos para ejecutar cada trabajo.

En un JCL los enunciados mas importantes son JOB (marca el inicio de un trabajo), EXEC (define el inicio de un paso dentro de un JOB) y DD (Indíca al sistema cuáles son los programas que queremos que ejecute). Cada uno de estos enunciados tienen parámetros con los que se ayudan a controlar con precisión las instrucciones que se le quieren dar al sistema.

Además de enunciados un JCL cuenta con utilidades que son programas que proporciona el Sistema Operativo para facilitar ciertas tareas al usuario. Estos programas son usados para realizar funciones en común como mantener y manipular datos del sistema, reorganizar, cambiar, comparar o manipular datos y ficheros de usuario. Los programas de utilidad se controlan mediante dos tipos de sentencias, las sentencias de JCL y las sentencias de utilidad.

Cuando hablamos de un proceso Batch con JCL forzosamente debemos hablar de COBOL que es el lenguaje en que están escritos los programas que se van a gestionar. Cuando tenemos una aplicación Batch donde se manipulan grandes cantidades de datos y se usan tablas DB2 surgen problemas como bloqueo de tablas o recuperación de datos muy costosa, para intentar corregir esto se ha creado un reposicionamiento que consiste en codificar programas COBOL y JCL's de tal modo que en caso de error durante la ejecución, se pueda volver a arrancar el proceso desde el último dato tratado y confirmado.

Así pues el reposicionamiento tiene elementos que serán descritos en este trabajo para poder explicar lo más claro posible este proceso, con esto se pretende mostrar como un lenguaje tan viejo como JCL aun tiene cabida en nuestros días y no parece que este desaparezca en un futuro próximo.

Índice

Índice de Figuras

IMPORTANCIA DE LA TEMÁTICA

Actualmente los trabajos por lotes son ampliamente utilizados en supercomputadores, durante mucho tiempo se ha utilizado este tipo de procesamiento de datos por lo que es muy completo, lo cual significa que muchas empresas basan sus transacciones en este tipo de procesamiento para lo cual es indispensable contar con un buen programa de organización para que el ciclo de información sea eficiente y por lo tanto satisfactorio.

El lenguaje JCL es empleado en muchas empresas donde se tienen grandes cantidades de datos y donde las tareas son repetitivas por ejemplo en Bancos, organismos gubernamentales, empresas privadas etc. ya que se usa para una simple nómina hasta operaciones contables bancarias con un nivel muy alto de dificultad.

Por lo anterior es importante que el egresado de ingeniería en computación tenga una idea clara de los procesos tipo Batch y JCL ya que hay una gran demanda laboral para quien tenga estos conocimientos.

PLANTEAMIENTO DEL PROBLEMA

En nuestros días no es posible encontrar fácilmente literatura sobre trabajos por lotes y mucho menos manuales que hagan referencia al Lenguaje de Control de Trabajos (JCL), así pues es fundamental tener un escrito donde se plasme la importancia de esta herramienta y la forma en que se utiliza, para dar al egresado de la carrera de ingeniería en computación una base.

MÉTODOS Y TÉCNICAS DE INVESTIGACIÓN

Para realizar este escrito se va a desarrollar un método de búsqueda bibliográfica además será enriquecido con la experiencia laboral que he tenido en área bancaria, como analista y desarrollador de programas JCL y COBOL en la aplicación de Compensación Bancaria durante 6 años.

CAPÍTULO 1. MVS

INTRODUCCIÓN

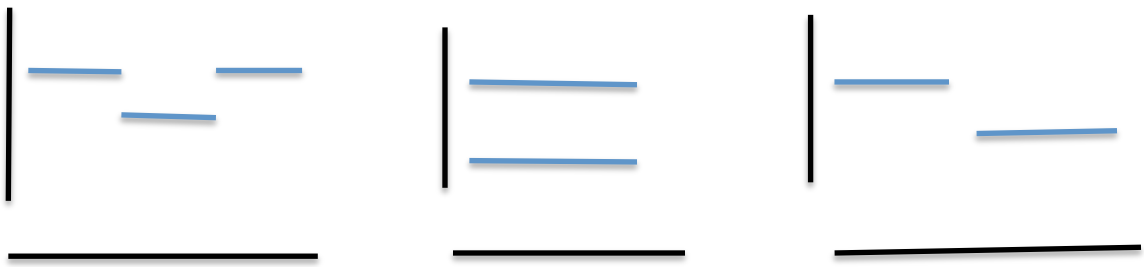
El MVS es el sistema operativo de IBM para sus grandes procesadores. Normalmente, los ordenadores que utilizan este sistema operativo suelen estar destinados a la gestión masiva de datos permitiendo funciones de multiprogramación, multiproceso y pipeline dando soporte a uno o varios sistemas de teleproceso.

El término de MULTIPROGRAMACIÓN se da a aquellos sistemas que permiten la ejecución simultánea de varios procesos en el ordenador, repartiendo el tiempo de ejecución entre las distintas secuencias de instrucciones de cada uno de los programas.

Mientras que el termino de MULTIPROCESO se aplica a aquellos sistemas que por tener varios procesadores, tienen la capacidad de ejecutar dos o más instrucciones en el mismo instante y en consecuencia, pueden ejecutar dos o mas procesos al mismo tiempo.

En cuanto PIPELINE consiste en múltiples procesos ordenados de tal forma que el flujo de salida de un proceso alimenta la entrada del siguiente proceso. Los elementos del pipeline son generalmente ejecutados en paralelo, en esos casos, debe haber un almacenamiento tipo buffer insertado entre elementos.

La Figura 1 muestra gráficamente como trabaja el paralelismo.



Multiprogramación Multiproceso Pipeline

Figura 1. Formas de trabajar el paralelismo.

HISTORIA

La importancia de los Sistemas Operativos nace en los 50's, cuando se hizo evidente que el operar una computadora por medio de enchufes (primera generación) y luego por medio del trabajo en lote (segunda generación). Esta etapa se podía mejorar, se comenzó a ver que las tareas que realizaba el operador de la computadora podían plasmarse en un programa que a través del tiempo resultó ser un Sistema Operativo ya que tenía una enorme complejidad al administrar los recursos de una computadora. Los primeros Sistemas Operativos fueron GM-NAA I/O y SHARE desarrollados por la General Motors' Research para el computador IBM 704. Posteriormente IBM tomó el proyecto, y con algunas mejoras, lo nombró IBSYS. El principal componente de IBSYS fue una serie de instrucciones conocidas como Job Control Language (JCL). Anterior a IBSYS, IBM produjo para su IBM 709, 7090 y 7094 un Sistema Operativo cuyo único propósito fue compilar programas FORTRAN, El Sistema Operativo fue nombrado Fortran Monitor System (FMS).

Ya en la tercera generación de computadoras nace OS/360 de IBM que posteriormente se llamaría MVS (Multiple Virtual Storage, Múltiple Almacén Virtual en inglés) Se manejaban tres variantes del OS/360: PCP (Programa de

control primario) de un solo flujo por lo tanto no soportaba la ejecución de tareas múltiples, MPF (Multiprogramación con un número fijo de tareas) inicialmente cuatro tareas posteriormente aumento a quince y MVT (Multiprogramación con un número variable de tareas) era una mejora que era capaz de la ejecución de múltiples tareas. Sobre esta base, el sistema SVS (Single Virtual Storage, Único Almacén Virtual) añadió el "almacén virtual", mejor conocido como memoria virtual; el espacio de direccionamiento de esta memoria era compartido por todas las aplicaciones.

Finalmente en base a SVS y MVT en 1974 MVS añadió la capacidad de que cada programa tuviera su propio espacio de direccionamiento de memoria, de allí su nombre. Fue el Sistema Operativo más usado en los modelos de mainframes System/370 y System/390 de IBM.

EVOLUCIÓN MVS

La versión 1 de MVS, con sus diferentes revisiones (releases), es lo que comúnmente se conoce como MVS/370, se introduce el concepto de espacio de direcciones.

La versión 2 de MVS más sus diferentes versiones componen el MVS/XA (eXtended Architecture). Esta realmente fue una revisión de la versión anterior para solucionar la gran cantidad de problemas presentados en aquella. El MVS/XA es una versión que afecta tanto al hardware como al software ya que la memoria virtual pasa de 16 Mb a 2 Gb, necesitando 4 bytes para direccionar cada posición de memoria. Además introduce un subsistema de canales que gestiona todas las operaciones de entrada y salida liberando de ellas a los procesadores.

La versión 3 se corresponde con lo que se conoce como MVS/ESA (Enterprise System Architecture). Esta versión se diferencia de la anterior, entre otras cosas,

en que puede direccionar hasta un máximo de 15 espacios de direcciones de 2 Gb. Posibilita la utilización de los espacios de direccionamiento para datos. Así mismo esta versión introduce el uso de micro código.

Luego de MVS/ESA se renombró como OS/390 con esta versión se incrementa la productividad de la instalación de cómputo mediante la asignación dinámica de recursos. Automatiza las operaciones de la computadora y la administración de trabajo, recurso y carga de trabajo. Con esto se incrementa la productividad de los recursos humanos.

Después surge z/OS y combina una serie de productos, antes separados, relacionados algunos de los cuales son todavía opcionales. z/OS ofrece los atributos de los Sistemas Operativos modernos, pero también conserva muchas de las funciones originarias de la década de 1960, y cada década posterior que todavía se encuentran en el uso diario. z/OS se introdujo por primera vez en octubre de 2000. La versión más reciente es z/OS Versión 2 Release 1.

COMPONENTES PRINCIPALES

VTAM Virtual Telecommunications Access Method (Método de Acceso Virtual de Telecomunicaciones). Este método fue creado por IBM a fin de aprovechar las nuevas capacidades del Hardware del S/370, de la memoria virtual y de los terminales y unidades de control inteligentes.

TSO Time Sharing Operation (Operaciones de tiempo compartido). Es un subsistema de IBM para poder acceder de forma interactiva a los servicios de MVS. Normalmente va acompañado de un producto a base de paneles, para hacer más sencillo su manejo. Este producto es ISPF/PDF.

CICS Customer Information Control System (Sistema de control de información de clientes) CICS es un gestor de transacciones. La ejecución de un programa es una transacción, y cada transacción genera una Tarea. Una Transacción es

cada una de las entradas que se realiza desde el Terminal. Una Tarea es la unidad de trabajo de CPU creada por una transacción. Cuando se invoca una transacción, un programa determinado se carga en memoria y se inicia la Tarea. Así aunque varios usuarios invoquen la misma transacción, cada uno tendrá una Tarea distinta.

CICS es multitarea (concurrencia). Además varias tareas diferentes pueden compartir el mismo programa si éste es reentrante (no cambia en ningún momento).

JES Job Entry Subsistem (Subsistema de entrada de trabajos) Subsistema de entrada de trabajo (Gestor de trabajo por lotes). De esta manera el JCL le dice al Sistema Operativo todos los requerimientos de entrada y salida que se necesitan para ejecutar un proceso o varios, en una secuencia determinada.

CAPÍTULO 2. JES: SUBSISTEMA DEDICADO A LA GESTIÓN DE TRABAJOS BATCH

INTRODUCCIÓN

El procesamiento Batch es usado para programas que puedes ser ejecutados con mínima interacción humana y en una hora calendarizada, después de que un trabajo en Batch se envía al sistema para su ejecución, normalmente no hay más interacción humana con el trabajo hasta que éste se complete.

El encargado de gestionar la entrada de trabajos Batch es el JES (Job Entry Subsystem), este surge de la necesidad de un programa que se encarga de controlar las lectoras de tarjetas, fichas perforadas y las impresoras. Así fue heredando la función de interpretar las sentencias que le envían como entrada para la ejecución de un trabajo. JES maneja colas de entrada y salida de datos, recibe trabajos dentro del Sistema Operativo y los calendariza para su procesamiento, además controla la salida resultado del procesamiento.

Este Subsistema se ejecuta en MVS y existen dos versiones el JE2 que es el resultado de la revisión del programa HASP (Houston Automatic Spooling Priority) y el JES3 que viene de la revisión de programa ASP (Attached Support Processor).

Para ejecutar trabajos Batch en MVS se puede utilizar un comando SUBMIT de TSO (Time Sharing Option) para enviar a una cola de entrada de trabajos del JES el JCL que sirve de parámetro a dicho comando o desde un programa en ejecución, el cual puede enviar otros trabajos a través del lector interno de JES.

PRINCIPALES FUNCIONES

Las funciones que lleva a cabo el JES son aceptar y formar trabajos enviados para ejecución, esto en base a sus clases; formar trabajos enviados para un iniciador (programa que solicita el siguiente trabajo en la lista); verifica la validéz de la sintaxis del JCL, reserva y asignación de recursos (Fichas DD del JCL); aceptar salida de un trabajo mientras se encuentra corriendo y forma la salida; opcionalmente, puede imprimir la salida o almacenarla en el spool para que el manejador de salida la acceda. La Figura 2 muestra las funciones del JES.

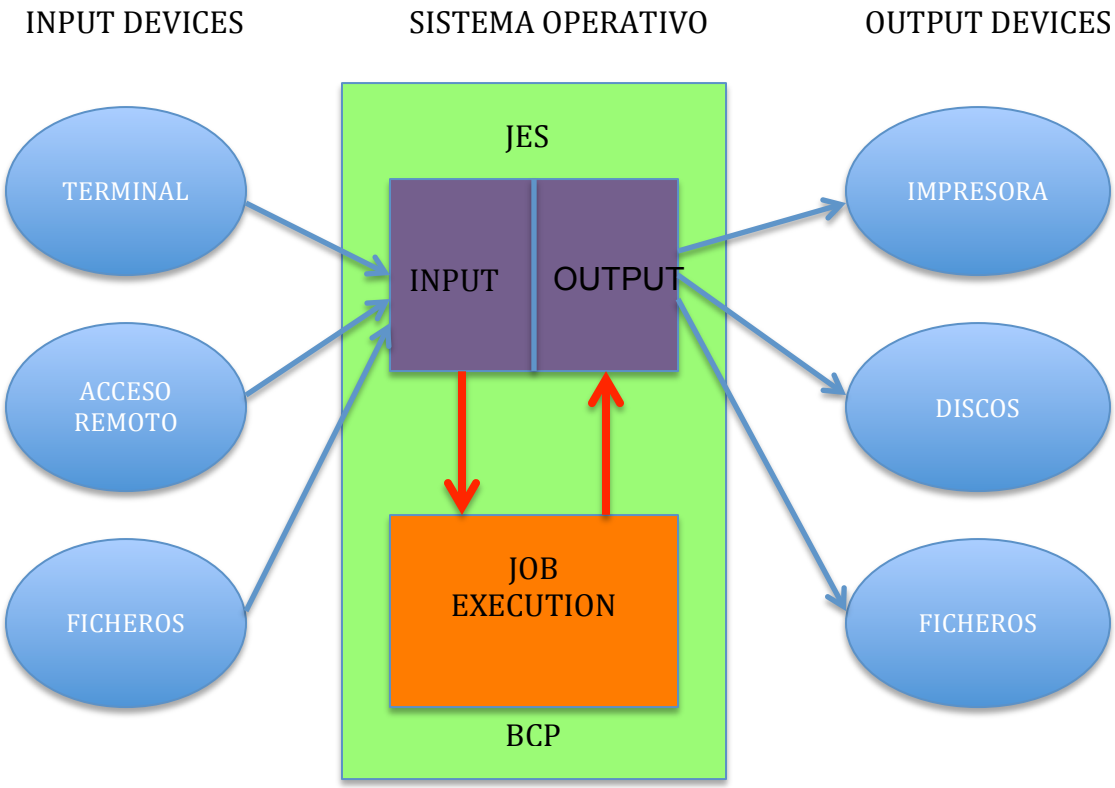


Figura 2. Funciones JES.

COMPONENTES PRINCIPALES

Los componentes principales del JES son los iniciadores, el spool y fichero de check-point. Los iniciadores son parte integral del Sistema Operativo que lee, interpreta y ejecuta el JCL, maneja la ejecución de trabajos tipo Batch, uno a la vez en el mismo espacio de direcciones si diez iniciadores se encuentran activos (diez espacios de direcciones), entonces diez trabajos del tipo Batch pueden correr al mismo tiempo. Normalmente corre en múltiples espacios de direcciones.

Los JOB's mediante el parámetro CLASS, tienen asignado la clase de entrada del JES en la que se debe ejecutar. Esto es, los iniciadores son colas que tienen asignadas unas clases de proceso, y sirven para seleccionar trabajos a fin de repartir la carga de trabajos Batch del sistema. La clase es un carácter entre A-Z o 0 -9, de este parámetro se hablara más a detalle en el capitulo 5.

Un ejemplo de cómo se codifica el parámetro CLASS en un JCL es el siguiente:

```
//NOMPROGR  JOB (000), CLASS = C
```

Si el sistema tiene definidas las siguientes relaciones:

INICIADOR	CLASES	TRABAJOS
1	ABD	8
2	ARK	36
3	AD	27
4	CE	10

El sistema podrá asignar dicho JOB solo al iniciador 4 pues es el único que puede ejecutar los trabajos de la cola C. Suponiendo que tiene 10 trabajos ejecutándose en dicho iniciador, el trabajo se encolara a la espera de su turno.

El SPOOL (Simultaneius Peripherical Operations On Line) es un método para formar y retener datos de entrada o salida, además de contener todos los trabajos que estén ejecutándose, en espera de ejecutar o en espera de salida. Para realizar la gestión, el SPOOL se organiza formando colas de entrada, ejecución y salida.

En las colas de entrada están todos los trabajos que están esperando un iniciador libre asignado a su clase para poder ejecutarse, en las colas de ejecución se encuentran los trabajos que se están ejecutando y por último, pero no menos importante, en la cola de salida están los trabajos que hayan terminado su ejecución y que están esperando una salida impresa.

El fichero check-point es en el que se basa JES para establecer las operaciones a realizar. La configuración de todo este sistema es labor del departamento de técnica de sistemas. Normalmente, en las instalaciones se definen varios ficheros de SPOOL, pero en cambio, sólo se define un fichero de check-point, y opcionalmente otro, que sirve como una copia del primero, para poder reactivar las funciones del JES, en caso de error en el primer fichero de check-point.

El fichero contiene una referencia de todas las colas de entrada y salida, además de los trabajos en ejecución, por lo que este es el fichero que consulta y mantiene el JES para, explorando las colas, saber cuál es el trabajo que debe atender en primer lugar; y una vez establecido éste, accede el SPOOL, donde se encuentra realmente el JCL, para enviarle a ejecución, al tiempo que actualiza el estado de los ficheros con las colas afectadas.

CAPÍTULO 3. TIPOS DE ARCHIVOS MANEJADOS EN PROCESOS BATCH

INTRODUCCIÓN

Los Sistemas Operativos se ayudan de los archivos para el almacenamiento de datos y programas en los dispositivos de memoria masiva, la forma en que se accede a ellos es por medio de un lenguaje de programación en este caso COBOL.

Una gran ventaja es que permiten superar las limitaciones impuestas por el tamaño de la memoria principal a la hora de manipular grandes volúmenes de datos. Dada la gran cantidad de datos que puede manejar una computadora en la actualidad, el uso de los dispositivos de almacenamiento como memoria secundaria permite que la cantidad de datos que trata un programa no esté limitada por el tamaño de la memoria principal.

Como tal, un archivo no es más que una agrupación de datos, cuya estructura interna es la que el usuario, el programador o el Sistema Operativo, le haya conferido implícitamente. Así mismo, un archivo es independiente de un programa ya que un mismo archivo creado por un programa puede ser utilizado por otros.

ARCHIVOS SECUENCIALES

Se puede decir que un archivo secuencial es un conjunto de información estructurada en unidades de acceso denominada registro.

La forma más común de estructura de archivo es el archivo secuencial. En este tipo de archivo se usa un formato fijo para los registros. Todos los registros tienen el mismo tamaño, constan del mismo número de campos de tamaño fijo en un orden particular. Un archivo secuencial es la forma más simple de almacenar y recuperar registros de un archivo, se almacenan los registros uno tras otro. El primer registro almacenado se coloca al principio del archivo, el segundo se almacena inmediatamente después y así consecutivamente.

Para leer un archivo secuencial, el sistema siempre comienza al principio del archivo y lee un registro a la vez hasta llegar al registro deseado. Para facilitar la lectura se debe conocer la longitud y la posición de cada campo.

Hay diferentes formatos de registros en un archivo secuencial los cuales son: fijo y variable. El formato fijo se caracteriza por tener todos los registros con la misma longitud, y todos los bloques o conjuntos físicos de registros lógicos de un mismo fichero, también tienen el mismo tamaño. El máximo tamaño del bloque es de 32760 Bytes. En el formato variable los registros lógicos que componen el fichero tienen longitudes distintas. El tamaño máximo del bloque es de 32760 mientras que el de registro es de 32756 bytes. La siguiente imagen muestra cómo se representa un archivo secuencial.

Registro 1
Registro 2
Registro 3
Registro 4
Registro 5
Registro 6

Normalmente el uso de los archivos secuenciales se da en procesos en lote, donde se ha hecho notar que son eficientes cuando se llevan a cabo diversas operaciones sobre una gran cantidad de registros o de todo el archivo, tales como respaldo de datos, generación de reportes, transmisión física de datos, archivos de nómina etc. los archivos secuenciales proveen la mejor utilización de espacio y son rápidos cuando el acceso es secuencial. En el capítulo 7 se verá como definir un archivo secuencial.

ARCHIVOS PARTICIONADOS: PDS

Un PDS (Partitioned Data Set) es un conjunto de registros llamados miembros que son escritos en forma secuencial ascendente y registrados con un nombre en el directorio del archivo, contienen cierto número de miembros (ficheros secuenciales) y un directorio. A estos archivos se les conoce comúnmente como LIBRERÍAS o BIBLIOTECAS. Es almacenado en un sólo volumen (DASD.- Direct Access Storage Device).

El directorio es un índice que es usado para localizar un miembro dentro de un PDS, está situado en el comienzo del fichero y contiene una entrada para cada miembro. Las entradas están ordenadas alfabéticamente por nombre de miembro.

ARCHIVOS VSAM

Este tipo de organización de fichero es especial contando además con un método de acceso propio. Su nombre proviene del acrónimo Virtual Storage Access Method. Debiendo ser procesados por sistemas que traten memoria virtual, tal y como indica su nombre, y además deben residir en disco.

Los registros de un VSAM pueden ser de una longitud fija o variable. Están organizados en bloques de tamaño fijo llamados Intervalos de Control (CI) y a su vez en divisiones más grandes llamadas Áreas de Control (AC). El tamaño de los Intervalos de Control se miden en bytes y las Áreas de Control en número de pistas o cilíndros de disco. Puesto que los Intervalos de Control son las unidades de transferencia entre el disco y la computadora, una lectura completa leerá un Intervalo de Control. Por su parte, las Áreas de Control son las unidades de reserva de espacio, de tal manera que cuando al definir un VSAM, se definen un número integro de Áreas de Control.

Existes cuatro tipos de archivos VSAM:

KSDS (Key Secuenced Data Set) En este tipo de archivos los registros son almacenados en un orden secuencial ascendente de acuerdo a un campo llamado llave (Registro Indexado). La llave debe ser única y estar en la misma posición en cada registro. En un archivo KSDS se pueden adicionar, actualizar, consultar o borrar registros. Los registros lógicos (LR) pueden variar de tamaño, además en cada área de control hay n Intervalos de control (CI) como se muestra en la Figura 3.

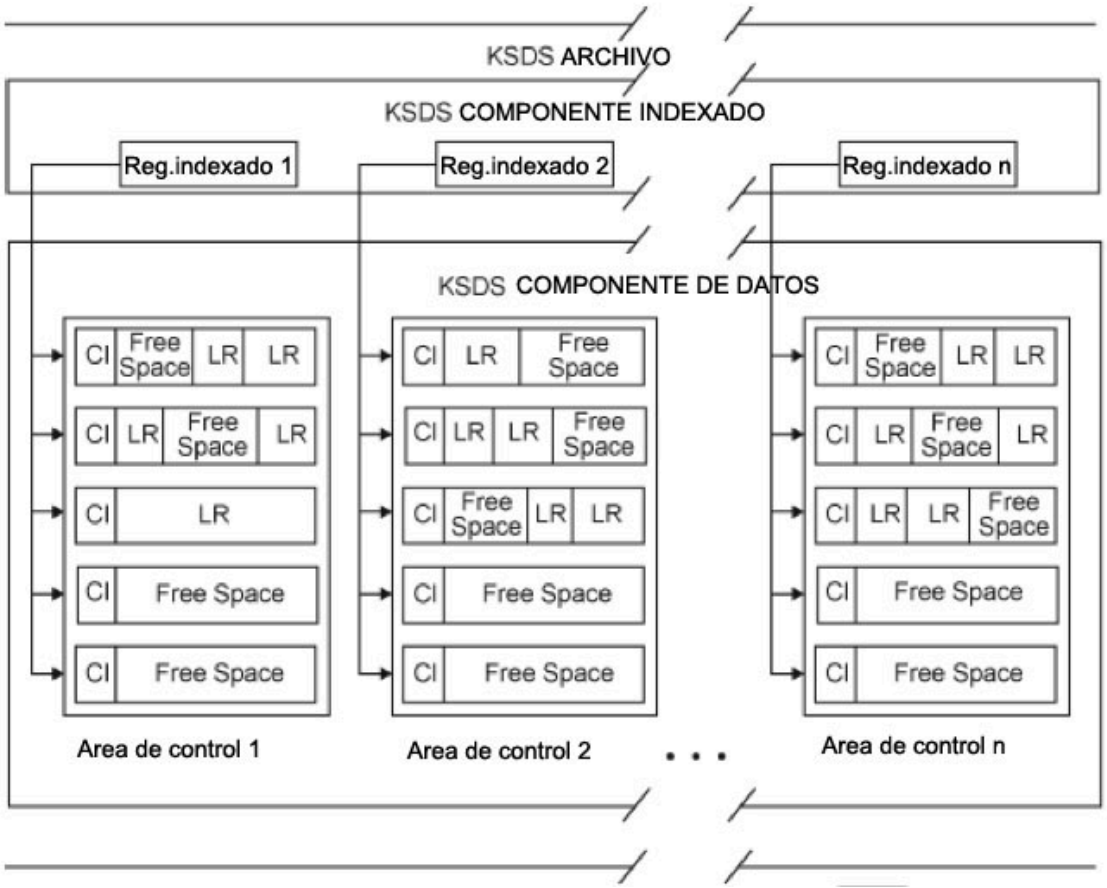


Figura 3. Ejemplo archivo KSDS.

RRDS (Relative Record Data Set) Contiene registros ordenados por un número de registro relativo, y sólo pueden tener acceso por medio de este número. Aquí cada intervalo de control (CI) tiene n registros lógicos (LR) que son del mismo tamaño. Un archivo RDS se representa en la Figura 4.

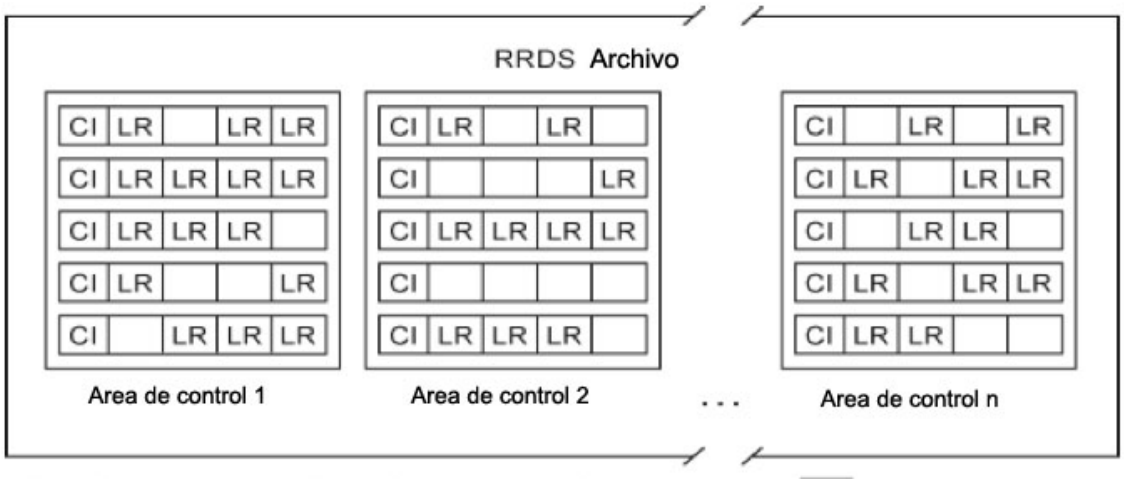


Figura 4. Archivo RRDS.

ESDS (Entry Sequence Dala Set) Su forma de uso es muy similar a un archivo secuencial. Los registros son insertados sólo al final del archivo, no pueden ser borrados, sólo pueden marcarse para que ese espacio sea reutilizado; pueden definirse sólo índices alternos. En el intervalo de control (CI) los registros lógicos (LR) pueden ser distintos unos de otros. Además que en área de control varían los intervalos de control como se muestra en la Figura 5.

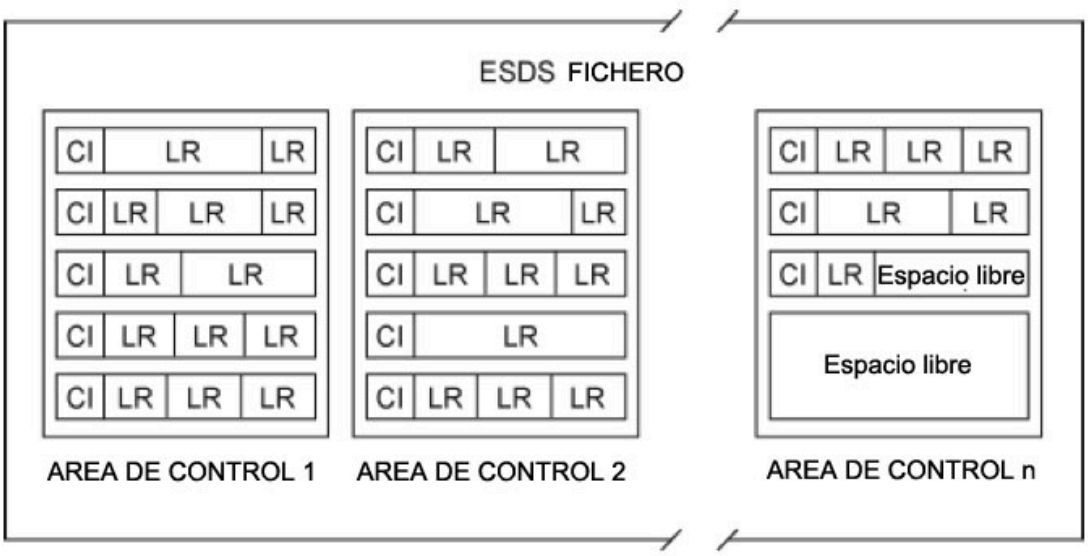


Figura 5. Archivo ESDS.

LDS (Linear Data Set) Los archivos VSAM lineales son raramente utilizados por

programas aplicativos. Son más efectivos en aplicaciones especializadas como DB2 que los utiliza para almacenar datos. Sus principales características son: No se cuenta con índices primarios o alternos; No almacena información de control por lo que no puede tener acceso como si almacenara registros individuales

Debido a que aún no se ha mencionado la sintaxis de JCL se creó el anexo I para consultar ejemplos de manipulación de archivos VSAM.

ARCHIVOS GDG

Un GDG (Generation Data Group) consiste en un conjunto de archivos con un mismo nombre que están cronológicamente relacionados, es decir, el nombre del fichero es el mismo, únicamente que se genera una nueva versión del fichero cada vez que se crea uno nuevo. Se utilizan para almacenar diferentes versiones de un sólo archivo, cada archivo en un GDG es conocido como una generación y se pueden tener hasta un máximo de 255 generaciones y pueden ser almacenadas en cualquier tipo de dispositivos (disco, cinta, cartucho, etc.).

El fichero actual siempre es aquel que se referencia como (0), siendo el anterior (-1) y uno posterior sería (+1).

Para entender mejor cómo funcionan lo haremos a base del siguiente ejemplo.

XXXXXXXX.YYYYYYYY.ZZZZZZZZ(0) --> Última versión / versión actual
XXXXXXXX.YYYYYYYY.ZZZZZZZZ(+1) --> Versión nueva
XXXXXXXX.YYYYYYYY.ZZZZZZZZ(-1) --> Versión anterior a la actual

Supongamos que tenemos un fichero de 3 versiones con los datos de los últimos 3 días de la semana.

Nuestro fichero tendría este aspecto:

XXXXXXXX.YYYYYYYY.ZZZZZZZZ --> Base (no contiene información)
XXXXXXXX.YYYYYYYY.ZZZZZZZZ.G0001V00 --> Datos Martes 1
XXXXXXXX.YYYYYYYY.ZZZZZZZZ.G0002V00 --> Datos Miércoles 2
XXXXXXXX.YYYYYYYY.ZZZZZZZZ.G0003V00 --> Datos Jueves 3

Para hacer referencia a la versión que queramos lo indicaremos de este modo:

XXXXXXXX.YYYYYYYY.ZZZZZZZZ(-2) --> Datos Martes 1
XXXXXXXX.YYYYYYYY.ZZZZZZZZ(-1) --> Datos Miércoles 2
XXXXXXXX.YYYYYYYY.ZZZZZZZZ(0) --> Datos Jueves 3

Imaginemos que queremos guardar los datos del Viernes 4. Para crear la nueva versión de datos le indicaremos:

XXXXXXXX.YYYYYYYY.ZZZZZZZZ(+1)

Una vez creada una nueva versión nuestro fichero, la versión (+1) que hemos usado pasará a ser la (0), y la que era (0) pasará a ser (-1), es decir, quedará del siguiente modo:

~~XXXXXXXX.YYYYYYYY.ZZZZZZZZ(-3) --> Datos Martes 1~~.

(La versión más antigua se pierde)
XXXXXXXX.YYYYYYYY.ZZZZZZZZ(-2) --> Datos Miércoles 2
XXXXXXXX.YYYYYYYY.ZZZZZZZZ(-1) --> Datos Jueves 3
XXXXXXXX.YYYYYYYY.ZZZZZZZZ(0) --> Datos Viernes 4

Si quisiéramos utilizar la versión del Viernes 4, ya no haríamos referencia a la versión (+1), sino a la (0).

Sin embargo, si dentro del mismo JCL, creamos una versión nueva y a continuación, en otro paso, la utilizamos (en un programa que lea el fichero por ejemplo), hay que referirse al fichero como (+1). Esto es porque hasta que no acaba la ejecución del JCL, no se actualiza la información de las versiones, por lo tanto si utilizáramos la versión (0) en el programa no estaríamos tomando la versión creada en el paso anterior, sino la versión anterior a esa.

Para poder ver la implementación de un GDG con sintaxis de JCL se creó el anexo 2.

CAPÍTULO 4. JCL: LENGUAJES DE CONTROL DE TRABAJOS

INTRODUCCIÓN

Como ya sabemos, JES es el quien controla los trabajos por lotes. Una de sus funciones es preparar los trabajos que se le envían para su ejecución, interpretando las sentencias de control que componen cada una de las cadenas o JOB's. Estas cadenas son escritas en un lenguaje especial llamado Lenguaje de Control de Trabajos (JCL).

JCL es un conjunto de especificaciones que constituyen un lenguaje de programación de tareas para el Sistema Operativo que gobierna un equipo informático, generalmente un Mainframe. Mediante declaraciones y sentencias de JCL se informa al Sistema Operativo de las tareas que debe realizar, la secuenciación de las mismas y los contenedores de datos de entrada y salida para cada uno de los trabajos a ejecutar.

Nos ayuda a ejecutar programas, definir ficheros, etc., dentro del SO de un Mainframe. Proporciona un alto grado de flexibilidad e independencia respecto de la localización física de los ficheros y de los programas. A través de JCL se puede especificar quién envía el trabajo y que recursos (programas, archivos, memoria) y servicios son necesarios para que el sistema procese el programa.

EJEMPLO DE UN PROCESO BATCH DE RECEPCIÓN Y DIAGNÓSTICO DE LOTES

El lote es entregado dentro de una cinta y debe ser copiado a un archivo secuencial para poder trabajar con él. El lote debe ser registrado en la entidad de control de lotes y se le hace un diagnóstico a su información, generando un archivo de notificación. Por último, se debe respaldar el lote recibido.

Por ejemplo, primero copia el lote de información recibida en cinta a un archivo secuencial en disco, luego registra el lote de información recibido en la entidad de control de lotes, después hace un diagnóstico de la información contenida en el lote y genera un archivo de notificación. Y por último respalda el lote de información recibida. En la Figura 6 se muestra un diagrama que muestra el proceso de recepción y diagnóstico de lotes.

Podemos ver que el proceso (Figura 6), se divide en los siguientes pasos:

1. Copiar el lote contenido en la cinta a un archivo secuencial nuevo.
2. Correr el programa PXP000, para registrar el lote en la entidad de control de lote.
3. Correr el programa PXP001 para diagnosticar la información contenida en el lote y generar el archivo secuencial nuevo con la información de la notificación.
4. Respalda el lote recibido en una cinta.

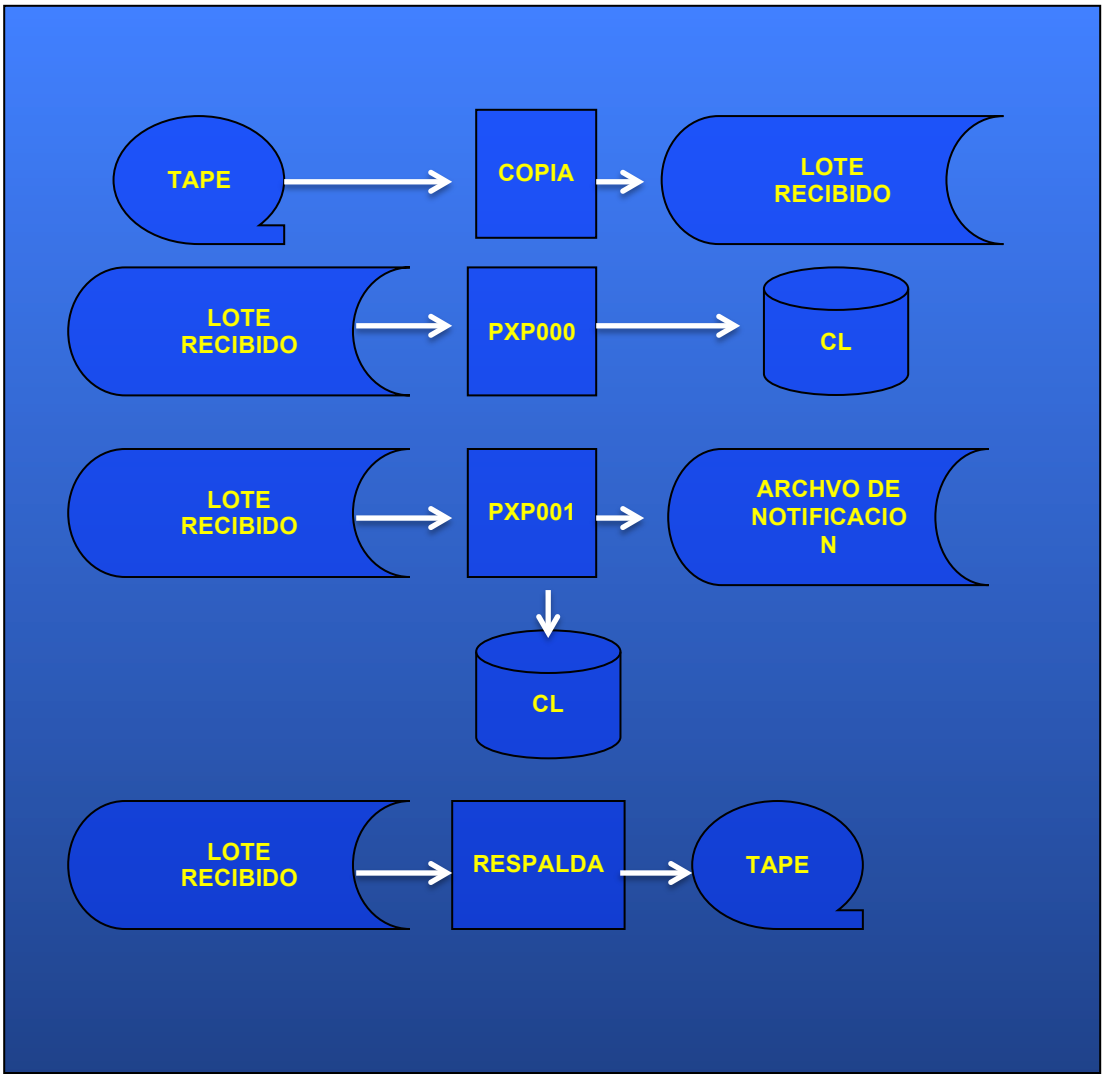


Figura 6. Proceso de recepción y diagnostico de lotes.

EJECUCIÓN DE UN JOB EN EL SISTEMA

El JES (Job Entry subsystem) es un subsistema del MVS que administra al JOB antes y después de la ejecución. La Figura 7 muestra un proceso de ejecución de un JOB:

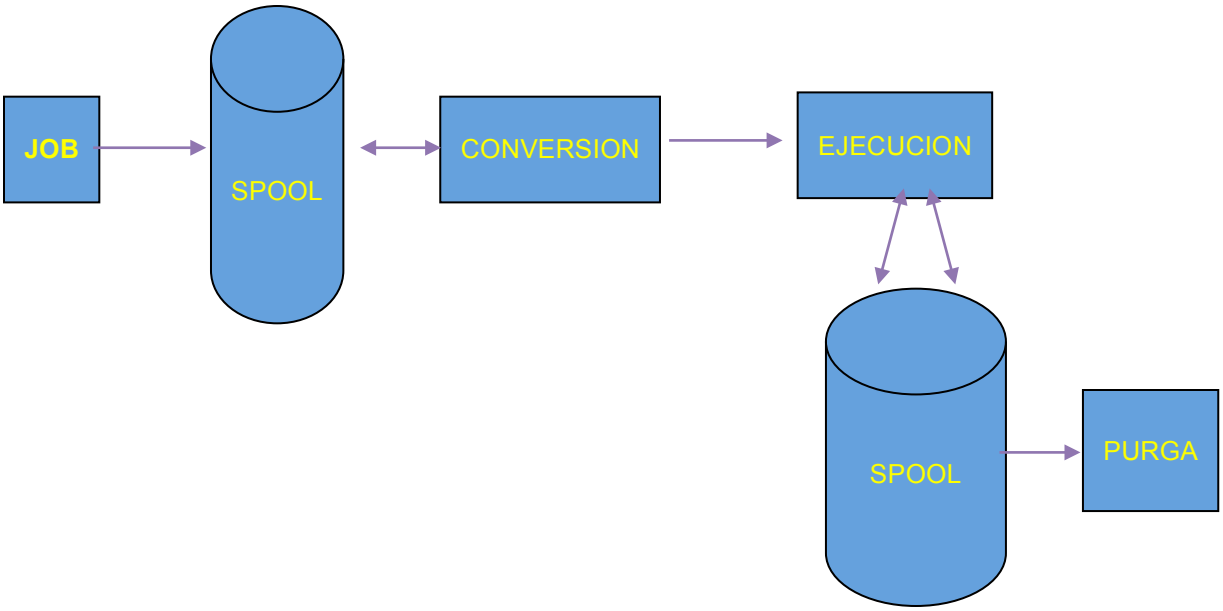


Figura 7. Ejecución de un JOB.

Primeramente JES recibe el JOB y lo prepara para su ejecución (conversión), después el JOB se almacena temporalmente en el SPOOL hasta que el MVS este listo para ejecutarlo, cuando existe mas de un JOB en espera administra su orden de ejecución. Mientras el MVS ejecuta el JOB el JES recibe las salidas generadas y cuando la ejecución del JOB termina, el JES lo purga del sistema (SPOOL).

ENUNCIADOS DE UN JCL

Los enunciados principales de JCL son los siguientes: JOB, EXEC y DD. Las características de los enunciados son las siguientes:

- Un enunciado se puede codificar de las columna 1 a la 72.
- Si es necesario codificar un enunciado en más de una línea, las líneas de continuación deben iniciar entre la columna 4 y 16.
- Cada enunciado puede dividirse en cinco partes diferentes.

- *Identificador:* Indica que se trata de un enunciado de JCL.
    - // en las columnas 1 y 2 indica un enunciado.
    - /* en las columnas 1 y 2 indica el final de un conjunto de datos in-stream (delimitador).
    - //* en las columnas 1, 2 y 3 indica un enunciado de comentario.
- *Nombre:* Nombre que identifica a un enunciado.
    - Debe empezar en la columna 3 y su longitud es de 1 a 8 caracteres.
    - El primer carácter debe ser alfanumérico.
    - Debe ser seguido por al menos un espacio en blanco.
- *Operación:* Sigue al campo de nombre e identifica el tipo de enunciado o comando (JOB, EXEC, DD, PROC, JCLLIB, ETC).
- *Parámetros:* contiene los parámetros del enunciado que deben estar separados por comas y pueden ser de dos tipos:
    - Posicionales
    - Keyword
- *Comentario:* Sigue al campo de parámetro y permite información de comentario.

CÓDIGO DE ENUNCIADO JOB:

```
//TTCPD070 JOB (000),‘HILDEBRANDO',
//         TYPRUN=SCAN,
//         RESTART=TTCPD070.TCUT0010,
//         REGION=0M,
//         NOTIFY=P002MMXT,
//         MSGCLASS=X,
//         CLASS=C,
//         MSGLEVEL=(1,1)
```

Existen dos tipos de parámetros: los posicionales y los keyword. Todos los parámetros posicionales se codifican primero que los keyword. Los parámetros posicionales son aquellos que de ser necesarios (según la sintaxis de cada enunciado de JCL), debe codificarse siempre en la misma posición, ejemplo:

//PASO0020 EXEC **PGM=IEBGENER**,COND=(0,LT)

Nota: Según la sintaxis del enunciado **EXEC**, el parámetro posicional **PGM** debe codificarse inmediatamente después de la palabra **EXEC**

//PASO0020 EXEC COND=(0,LT),**PGM=IEBGENER**

Nota: Esta sería una forma incorrecta de codificar el parámetro PGM.
Según la sintaxis, un parámetro puede codificarse en forma obligatoria u opcional. Si se omite un parámetro posicional, y se codifica el parámetro posicional siguiente, se debe codificar una coma en el lugar de parámetro omitido.

No codificar la coma si:

- El parámetro posicional omitido es el último parámetro posicional según la sintaxis.
- Todos los parámetros siguientes también son omitidos.
- Solo siguen parámetros keyword.
- Todos los parámetros posicionales omitidos.

Los parámetros keyword se colocan después de los parámetros posicionales y no requieren de un orden específico para su codificación.

```
//P003XEG  JOB (0), 'JOB DE PRUEBA',
```
Parámetros posicional

```
//          CLASS=Q, MSGLEVEL=(1,1),     Parámetros keyword
//          MSGCLASS=X                   Parámetro keyword
```

CAPÍTULO 5. ENUNCIADO JOB

El enunciado JOB debe ser el primer enunciado de JCL en cada trabajo (también llamado JOB), marca el inicio de un trabajo y especifica información para el proceso además debe ser codificado en la posición 12. El nombre del JOB puede tener hasta 8 caracteres, empezando siempre por un carácter alfabético. No se considera el guión como carácter especial. Un primer ejemplo de cómo se codifica un JOB es el siguiente:

```
//JOBNADA    JOB (000),'NO HACE NADA',
//                     CLASS='A',
//                     REGION=0M,
//                     MSGLEVEL=(1,1),
//                     NOTIFY=USUARIO
//*--------------------------------------
//* ESTE JOB NO HACE NADA
//*--------------------------------------
//
```

para este JOB tenemos:

JOBNADA Nombre del JOB

(000) Posicional-opcional. Informa la contabilidad para el JOB.

'NO HACE NADA' Posicional-opcional**.** Descripción del JOB (de 1 a 20 caracteres).

CLASS= Keyword-opcional**.**Asigna la clase de ejecución para el JOB, es de un carácter y puede ser de A - Z o 0 – 9.

REGION= Keyword-opcional. Especifica el tamaño máximo de memoria virtual para el JOB.

MSGLEVEL=Keyword-opcional. Controla la salida del JOB.

NOTIFY= Keyword-opcional. Designa al usuario que se notificará el final del JOB.

MSGLEVEL

Este parámetro especifica el tipo de mensajes que se desea tener en el log del JOB, es decir, los tipos de mensajes que se desea recibir de la ejecución del JOB.

MSGLEVEL=(X,Y)

El primer subparámetro (**X**) controla los enunciados que serán desplegados en el LOG del JOB. Una vez ejecutado el JOB se pueden ver los resultados en SPOOL.

- **MSGLEVEL=0** Imprime solo el enunciado del JOB.
- **MSGLEVEL=1** Imprime todos los enunciados del JCL y JES incluyendo todos los enunciados del procedimiento.
- **MSGLEVEL=2** Imprime solo los enunciados del JCL y JES del usuario quien ejecute el JOB, los enunciados en los procedimientos no son impresos.

El segundo subparámetro (**Y**) controla los mensajes que serán impresos en el LOG del JOB.

- **MSGLEVEL=(,0)** Si el JOB termina en forma normal, se imprimen sólo los mensajes de JCL de lo contrario, imprime todos los mensajes.

- **MSGLEVEL=(,1)** Todos los mensajes son impresos sin importar si el JOB terminó en forma normal o no.

Ejemplos de codificación:

```
//EJEMPLO1 JOB (1,2),'pruebas',MSGLEVEL=(0,0)
//EJEMPLO2 JOB ,'pruebas',MSGLEVEL=(,1)
//EJEMPLO3 JOB MSGLEVEL=2,ADDRSPC=REAL
```

Una vez que es ejecutado el JOB, en la SPOOL podemos ver el resultado de éste. La Figura 8 muestra un fragmento de la salida de un JOB. En él se distinguen las siguientes partes:
- LOG del JOB: en esta parte se registra la hora de comienzo de cada evento.
- Interpretación del JCL ejecutado: en el que se ve cómo el JES ha numerado cada una de las sentencias ejecutables, con independencia de que ocupen 1 o mas fichas de la cadena.
- La tercera parte se corresponde con la salida de mensajes devueltos por el sistema.

```
JES2 JOB LOG -- SYSTEM X

14.21.48 JOB07054  IRR010I  USERID DESJMP   IS ASSIGNED TO THIS JO
14.21.50 JOB07054  ICH70001I DESJMP   LAST ACCESS AT 14:21:41 ON W
14.21.50 JOB07054  $HASP373 DESJMPCA STARTED - INIT   CK - CLASS K
14.21.50 JOB07054  IEF403I DESJMPCA - STARTED - TIME=14.21.50
14.21.53 JOB07054  -                                          --TIM
14.21.53 JOB07054  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU
14.21.53 JOB07054  -DESJMPCA          PASO0010    00     35    .00
14.21.59 JOB07054  IEF404I DESJMPCA - ENDED - TIME=14.21.59
14.21.59 JOB07054  -DESJMPCA ENDED.  NAME-JMP-PRU             TOT
14.21.59 JOB07054  $HASP395 DESJMPCA ENDED
      1 //DESJMPCA JOB (30,123),'JMP-PRU',CLASS=K,MSGCLASS=X,
```

```
        //             NOTIFY=DESJMP,MSGLEVEL=(1,1)
        //* - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      2 //PASO0010  EXEC PGM=DESNAT,REGION=3000K,TIME=1400,
        //           PARM='STACK=INT,IM=D,MAXCL=0,A'
      3 //STEPLIB    DD DSN=JMP.NATURAL.DES,DISP=SHR

      4 //DDKARTE    DD DUMMY
      5 //DDDRUCK    DD SYSOUT=*
      6 //DDPRINT    DD SYSOUT=*
        ...
ICH70001I DESJMP   LAST ACCESS AT 14:21:41 ON WEDNESDAY, AUGUST 10
 IEF236I ALLOC. FOR DESJMPCA PASO0010
 IGD103I SMS ALLOCATED TO DDNAME STEPLIB
 IEF237I DMY  ALLOCATED TO DDKARTE
 IEF237I JES2 ALLOCATED TO DDDRUCK
 IEF237I JES2 ALLOCATED TO DDPRINT
...
```

Figura 8. Fragmento de la salida de un JOB en SPOOL.

CLASS

El parámetro CLASS indica la clase en el JES2 a la cual se enviará el JOB a ejecución. La clase es de un carácter entre A-Z O 0-9 y son definidas por el administrador del sistema, según la clase utilizada por el JOB, este tendrá características de ejecución distintas como máximo tiempo de ejecución, máxima memoria, virtual disponible, prioridad de ejecución etc.

El administrador del sistema establece las prioridades de uso de las clases. De esto depende si un JOB puede o no usar una clase en especifico. Cuando se lanza un JOB con una clase no definida por el administrador del sistema, el JOB quedará en espera permanente de ejecución, sin que se genere un mensaje de

error, si se omite el parámetro, el JOB es asignado a una clase definida por default.

MSGCLASS

Este parámetro especifica la clase de salida en la que el JES dejara los mensajes devueltos por el sistema, así como la interpretación de las sentencias del JOB.
Por defecto asume la clase que tenga asignada el usuario de TSO que envía el trabajo a ejecución. Valores: A - Z ó 0-9

Ejemplo: `//EJEMPLO4 JOB CLASS=B,MSGCLASS=8`

NOTIFY

Este parámetro especifica el identificador del usuario de TSO al que el sistema enviará un mensaje cuando finalice el JOB.
La codificación de este parámetro se ajusta al siguiente formato:

NOTIFY=identificador_usuario

El identificador del usuario corresponde con la identificación que se da al sistema cuando se hace LOGON para iniciar una sesión de TSO.

Ejemplo:

```
//EJEMPLO5 JOB MSGCLASS=8,CLASS=X,
//          MSGLEVEL=(1,1),NOTIFY=T976614
```

TIME

Especifica el tiempo máximo de ejecución en CPU del JOB. Se puede especificar a nivel JOB o a nivel EXEC.

- Si se especifica a nivel JOB, significa que el tiempo total para todos los pasos.
- Si se especifica a nivel EXEC, significa que es el tiempo máximo para ese paso.

NOLIMIT Indica que el JOB puede usar al procesador por un periodo ilimitado de tiempo. Codificar TIME=NOLIMIT es lo mismo que codificar TIME=1440.

1440 Indica que el JOB puede usar al procesador por un periodo ilimitado de tiempo. 1440 equivale a 24 horas.

MAXIMUM Indica que el JOB puede usar el procesador por el máximo periodo de tiempo permitido. Permite al JOB ejecutarse por 357912 minutos.

La codificación de TIME puede ser:

TIME=(MINUTOS,SEGUNDOS)
=(NO LIMIT)
=(1440)
=(MAXIMUM)

El siguiente ejemplo muestra cómo limitar el tiempo de CPU a 5 minutos y 30 segundos.

```
//EJEMPLO6 JOB (123,456),'JMP-PRU',CLASS=X,
```

```
//          MSGLEVEL=(1,1),NOTIFY=T976614,
//          RESTART=PASO2,TIME=(5,30)
            ...
```

Y este otro, le limita a 5 minutos.

```
// EJEMPLO7 JOB (123,'JMP'),CLASS=Y,NOTIFY=TEST87,
//          RESTART=PASO3.NATURALP,TIME=5
```

TYPRUN

Este parámetro Indica al JES la manera en que se debe procesar el JOB. En el caso de que no se especifique este parámetro el trabajo pasa a ejecución inmediatamente.

Los valores que puede tomar son:

- **TIPRUN=SCAN** Se revisa la sintaxis del **JCL** pero no se ejecuta.
- **TYPRUN=HOLH** Se revisa la sintaxis del **JCL** y si no hay ningún error se forma en una cola de entrada en el **SPOOL** y solo se ejecuta hasta que el operador lo libera desde el panel del SDSF.
- **TYPRUN=COPY** El JCL ejecutado es copiado a la clase de salida especificada en el MSGCLASS. No es ejecutado**.**
- **TYPRUN=JCLHOLD** Manda el **JOB** a estado **HOLD** pero sin revisar sintaxis.

Ejemplo:

```
// EJEMPLO8 JOB (123,456),'JMP-PRU',CLASS=X,
```

```
//            MSGLEVEL=(1,1),NOTIFY=T976614,
//            RESTART=PASO2,TYPRUN=SCAN
```

En este caso, sólo se revisa la sintaxis del JCL, no válida si existen los ficheros o datasets especificados en las distintas fichas DD de cada paso.
En el siguiente ejemplo, el JOB se interpreta y pasa a la cola de entrada, pero no se ejecuta hasta que no sea liberado.

```
// EJEMPLO9 JOB (123,'JMP'),CLASS=Y,NOTIFY=TEST87,
//           TYPRUN=HOLD
```

RESTART

El parámetro RESTART se utiliza para indicar al sistema el paso donde debe reiniciar la ejecución del JOB. Para utilizar el parámetro *RESTART* , los nombres de los pasos deben ser únicos.

Normalmente, este parámetro se utiliza para re arrancar un JOB desde el paso que terminó mal, ya que los primeros pasos se ejecutaron correctamente.
La codificación de este parámetro se ajusta al siguiente formato:

RESTART=*
RESTART=nombre_paso
RESTART=nombre_paso.nombre_proc

Donde:

* indica que el JOB se re arranca desde el primer paso.
nombre_paso especifica el nombre del paso desde el que se desea re arrancar.
nombre_paso.nombre_proc este formato se usa cuando se desea re arrancar

desde un paso que invoca a un procedimiento.

Ejemplo 1 : El parámetro *RESTART* indica que la ejecución del JOB debe iniciarse en el paso PASO0010 por lo que los pasos anteriores no corren (PASO0020)

```
//EJEMPLO0    JOB(000),'NO HACE NADA',
//                     CLASS= A,
//                     REGION=0M,
//                     MSGLEVEL=(1,1),
//                     MSGCLASS=X,
//                     RESTART=PASO0010,
//                     NOTIFY=USUARIO
//PASO0020 EXEC PGM=IEBFBR14                (CREA ARCHIVOS)
//PASO0010 EXEC PGM=IKJEFT01               (CORRE UN PROGRAMA)
```

Ejemplo 2 : El parámetro RESTART indica que la ejecución del JOB debe iniciarse en el paso TCSO0110 del PROC que se llama en el paso TTCPD300 del JOB.

```
//TTCPD300  JOB (000),'ENTREGA TRAN C',
//                    RESTART=TTCPD300.TCSO0110,
//                    REGION=0M,
//                    NOTIFY=P002XJG,
//                    MSGLEVEL=(1,1),
//                    MSGCLASS=X,
//                    CLASS=C
//TTCPD300   JOBLIB ORDER=(ISTSAR.PROPSAR.PROCLIB)
//TTCPD300   EXEC PROC=TTCPD300
//           PROBLIB=ISTAR,        (BIBLIOTECA DEL PROYECTO)
//           GPOBIB=ISTSAR,        (BIBLIOTECA GRUPO)
```

```
//           NBBVOLTRB=DESAPASS,    (VOLUMENES RESERVADOS DE TRAB)
//           PROARCH=ISTSAR,        (NOMBRE DEL ARCHIVO PROY
//           NBLDB2=SPTDB2.IUO.V310.SDSNLOAD,    (BIBL CARGA DB2)
//           SALX=X                 (CLASE DE SALIDA STANDARD(
```

REGION

Cuando cada paso de un JOB necesita diferentes espacios de memoria, el parámetro REGION debe especificarse a nivel EXEC o JOB. La región puede ser especificada en K (Kilo bytes) o M (Mega bytes). Si se especifica REGION=0M se indica al sistema que tome la memoria necesaria para que el JOB se ejecute. Se codifica de la siguiente manera:

REGION = VALOR K
= VALOR M

Especifica la memoria virtual necesaria para la ejecución del JOB. Puede codificarse en el enunciado JOB o en el enunciado EXEC.

- Si se codifica a nivel JOB, la memoria especificada se aplicará para cada paso del JOB y sobre escribe cualquier REGION especificada en un enunciado EXEC.
- Si se especifica un tamaño de memoria menor que el necesario para la ejecución del JOB o la memoria especificada no puede ser obtenida, el JOB termina en forma anormal (ABEND)

Ejemplo:

```
//EJEMPLOA JOB (123,456),'JMP-PRU',CLASS=X,
//            MSGLEVEL=(1,1),NOTIFY=T976614,
//            PERFORM=12,REGION=32M
```

CAPÍTULO 6. ENUNCIADO EXEC

El enunciado EXEC define el inicio de un paso dentro de un JOB y puede haber un máximo de 255 pasos dentro de un JOB. Con este enunciado especificamos que se quiere ejecutar un programa, un procedimiento catalogado o un procedimiento in-stream.

Al igua que el enunciado JOB tiene parámetros posicionales y de palabra clave. Los parámetros posicionales(PGM y PROC) son excluyentes entre si, y sólo sirven para especificar si se va a ejecutar un procedimiento o un programa. Estos deben ser declarados antes que el resto de párametros de palabra clave, cualquiera de los dos que se codifique, deberá ser el primero, y siempre debe codificarse uno de ellos.

Por su parte los parámetros de palabra clave tienen idéntica sintaxis tanto si son de un programa o si actúan como parámetros de un procedimiento. Si estos son codificados después del parámetro PGM son aplicados al programa que se va a ejecutar. Sin embargo cuando son declarados después del parámetro PROC los parámetros de palabra clave sustituyen a los que por defecto se hayan definido en el procedimiento.

PGM

Como se mencionó anteriormente, este parámetro debe ser el primero del enunciado EXEC. Se aplica cuando el paso deberá ejecutar un programa, en este caso será un programa escrito en COBOL el cuál debe estar almacenado en una librería del sistema. Por ejemplo el siguiente JOB ejecuta el PGM PCP00A(codificado con COBOL) el cual muestra en pantalla “HOLA”.

```
//JOBHOLA   JOB (000), 'JOB QUE DICE HOLA',
//                    CLASS=A,
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*         EJECUTA EL PROGRAMA DE SALUDO CONSOLA                *
//*-------------------------------------------------------------*
//PASO0010  EXEC  PGM=PCP00A,PARM=(12,23),REGION=512K
//                               TIME=MAXIMUM,COND=(0,LT)
```

PROC

Al igual que el parámetro anterior también ocupa la primera posición del enunciado EXEC. Se aplica cuando el paso deberá ejecutar un procedimiento. El procedimiento puede estar dentro del mismo JOB y es lo que denominamos 'in-stream'. Si el procedimiento se encuentra en una librería se denomina un procedimiento catalogado (Un procedimiento catalogado es un conjunto de sentencias de JCL que ha sido ubicado en una librería y puede ser recuperado por su nombre).

Un ejemplo de la codificación para este parámetro es la siguiente:

```
//JOBPROC   JOB (000),CLASS=A,
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*         EJECUTA UN PROCEDIMIENTO CATALOGADO                  *
//*-------------------------------------------------------------*
//PASO0010  EXEC  PROC=PROCESO1,PARM=(12,23),REGIO=512K
//                               TIME=MAXIMUM,COND=(0,LT)
```

Donde PROCESO1 se encuentra guardado en una librería del sistema.

ACCT

Cuando se quiere contabilizar el tiempo de ejecución del paso se usa ACCT, en caso de que hubiera que codificar sub parámetros, el valor de este parámetro ira delimitado por paréntesis y los sub parámetros separados por comas. La longitud total máxima de este parámetro es de 142 bytes.

La sintaxis para el parámetro es:

ACCT[.nom_paso_proc]=valor

Ejemplo:

```
//JOBPROC   JOB (000),CLASS=A,
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*   EJECUTA UN PROCEDIMIENTO IN-STREAM Y PARÁMETRO ACCT       *
//*-------------------------------------------------------------*
//PROCESO PROC
//PASONN  EXEC ...
...
//PASONM  EXEC ...
...
//        PEND
//* ------------------ fin procedimiento ------------*
//PASO1 EXEC PGM=PROGRAM1,ACCT=135
...
//PASO2 EXEC PGM=PROGRAM2,ACCT=(123,567)
...
//PASO3 EXEC PROC=PROCESO,ACCT=(123,999)
...
//PASO4 EXEC PROCESO,ACCT.PASONM=(123,555)
...
```

Donde:

- En el PASO1 se determina la información contable para PASO1
- En PASO2 la información contable tiene sub parámetros.
- En PASO3 se asigna información contable para todos los pasos del procedimiento PROCESO.
- En PASO4 se asigna información contable al PASO@1 del procedimiento. El resto de los pasos se contabiliza donde indique la información de la ficha JOB.

ADDRSPC

Este parámetro especifica como se ha de ejecutar un paso, si es en memoria virtual o memoria real. El valor se determina en que si se ejecuta en memoria real, no hay paginación, mientras si se ejecuta en memoria virtual si hay paginación lo cual implica un mayor tiempo de ejecución. Por defecto toma el valor de VIRT (Virtual) a menos que se haya especificado a nivel de JOB.

La sintaxis de este parámetro es la siguiente:

ADDRSPC[.nom_paso_proc]=valor

Ejemplo:

```
//JOBADDR   JOB (000),CLASS=A,
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*   EJECUTA UN PROCEDIMIENTO EN MEMORIA VIRTUAL O REAL        *
//*-------------------------------------------------------------*
```

```
//PROCESO PROC
//PASO@0  EXEC ...
...
//PASO@1  EXEC ...
...
// PEND
//* ------------------ fin procedimiento-------------------*
//PASO1 EXEC PGM=PROGRAM1,ADDRSPC=REAL
...
//PASO2 EXEC PGM=PROGRAM2,ADDRSPC=VIRT
 ...
//PASO3 EXEC PROC=PROCESO,ADDRSPC=VIRT
...
//PASO4 EXEC PROCESO,ADDRSPC.PASO@1=REAL
...
```

Donde:

- PASO1 determina que el paso se ejecute en memoria real.
- PASO2 determina que el paso se ejecute en memoria virtual.
- PASO 3 determina que todos los pasos del procedimiento se ejecuten en memoria virtual.
- PASO4 determina que PASO@1 del procedimiento se ejecute en memoria real.

COND

Condiciona la ejecución del programa al código de retorno del paso anterior, determina si el paso será ejecutado o no. Si el subparámetro Paso no es codificado, entonces la condición es evaluada para cada uno de los códigos de retorno arrojados por los pasos anteriores. Si alguno de los pasos anteriores cumple con la condición, entonces, el paso actual no se ejecuta.

Si un paso termina en forma anormal (ABEND), todos los pasos siguientes no son ejecutados. Cuando un paso no es ejecutado, su código de retorno es FLUSH.

Se puede especificar hasta 8 condiciones distintas, y solo se especificaran los paréntesis internos si se incluye mas de una condición. En este caso las distintas condiciones funcionan como la operación lógica OR.

Su sintaxis es la siguiente:

COND=(Valor,Operador[,Paso]…[,(Valor,Operador[,Paso]

Donde:

- Valor: 0-4095
- Operador:
  - GT (MAYOR)
  - LT (MENOR)
  - EQ (EQUAL)
  - NE (DIFERENTE)
  - GE (MAYOR O IGUAL)
  - LE (MENOR O IGUAL)
- Paso: Nombre del paso anterior.

Condiciones especiales:

- COND=EVEN ejecuta el paso no importando si algún paso anterior halla terminado con error (ABEND)
- COND=ONLY ejecuta el paso solo si algún paso anterior halla terminado en forma anormal (ABEND)

Ejemplos:

Se ejecuta PROCESO1 si todos los pasos anteriores han devuelto un código de retorno distinto de 0.

```
//PASO07    EXEC PROCESO1,COND=(0, NE)
```

Se ejecuta PROGRAMA si el código devuelto por PASO01 Y PASO02 ha sido 4.

```
//PASO22    EXEC PROGRAMA,COND=((4,EQ,PASO01),(4,EQ,PASO02))
```

Se ejecuta el programa IEFBR14 si el programa ejecutado en el paso PASO04 devolvió un código de retorno igual a 4.

```
//CARGAA4   EXEC PGM=IEFBR14,COND=(4,EQ,PASO04),
```

Se ejecuta PROCESO4 si todos los pasos anteriores han devuelto un código de retorno menor que 7.

```
//CARGAB5   EXEC PROCESO4,COND=(7,LT),
```

Se ejecuta el programa IEBGENER siempre.

```
//COPIDO1   EXEC PGM=IEBGENER,COND=EVEN
```

PARM

Este parámetro sirve para pasar valores a un programa que se está ejecutando. Se pueden pasar como máximo 100 caracteres como valores de un parámetro. Si el programa esta escrito en COBOL, estos parámetros son recogidos a través de la LINKAGE SECTION (Los parámetros de comunicación son declarados en la Linkage Section, en donde sólo se pasa la dirección de los datos y la información actual de estos).

El valor representa la información que se pasa al programa. En el caso de que esté formado por más de una expresión, éstas iran separadas por comas, las cuales pasan al programa, y todo el conjunto debe ir entre paréntesis o apóstrofes.

La sintaxis de PARM es la siguiente:

PARM = valor

Ejemplo:

En el caso en que el JCL no use DB2:

```
//JOBPROC   JOB (000),CLASS=A,
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*   EJECUTA UN PROGRAMA CON UNA FECHA COMO PARAMETRO          *
//*-------------------------------------------------------------*
//P01DE01  EXEC PGM=BG4CINT0,PARM='20131012',COND=(0,EQ)
 . . .
```

En el caso en el que el JCL use DB2:

```
//JOBPROC   JOB (000),CLASS=A,
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//* EJECUTA UN PROGRAMA CON UNA FECHA COMO PARAMETRO CON DB2    *
//*-------------------------------------------------------------*
//SYSTSIN  DD  *
   DSN SYSTEM(DB2D)
   RUN PROGRAM(BG4CINT0) PLAN(BVDBGPB) PARM('20131012')
   END
// . . .
```

El código en el programa COBOL sería la siguiente:

```
*********************************************
*CAPA LINKAGE SECTION                        *
*********************************************
LINKAGE SECTION.
*
01 REG-FECHA.
     05 PARM-ANNO                    PIC X(4).
     05 PARM-MES                     PIC X(2).
     05 PARM-DIA                     PIC X(2).
*********************************************
*              PROCEDURE DIVISION
*********************************************
PROCEDURE DIVISION USING REG-PARM.

. . .
```

REGION

Este parámetro tiene distinto significado según se haya especificado en el parámetro ADDRSPC el valor REAL o el valor VIRT.

Si se ha especificado ADDRSPC=REAL, el tamaño definido en el parámetro REGION indica el tamaño de la memoria real que se planifica para la ejecución del paso. Si se ha especificado ADDRSPC=VIRT, el tamaño definido en el parámetro REGION indica el máximo tamaño de memoria virtual que se puede admitir en las GETMAIN que se ejecuten en la ejecución del paso.

La sintaxis de este parámetro es la siguiente:

REGION=n
REGION=nK
REGION=mM

donde n es el numero de octetos, de Kb o de Mb que representa el tamaño

asignado o limitado.

Ejemplo:

```
//PASO0010  EXEC PGM=NBATBASE,REGION=3000K,
//          PARM='IM=D,MADIO=0,MT=0,INTENS=1'
```

TIME

Este parámetro expresa el tiempo máximo que un paso del Job puede hacer uso de la CPU. En el caso de que un paso alcanzara el valor especificado el paso terminaría su ejecución.

La codificación de este parámetro se ajusta al siguiente formato:

TIME=([minutos],[segundos])

Donde: minutos puede tomar los valores de 1 a 1440
segundos puede tomar los valores de 0 a 59

En el caso de que no se especifique el sub parámetro segundos, no es necesario especificar los paréntesis. Cuando se especifica TIME=1440 , es decir 24 horas, se esta indicando que no hay límite para el uso de la CPU en ese paso.

Ejemplo:

```
//JMP16  JOB (123,456),'JMP-PRU',CLASS=X,
//            MSGLEVEL=(1,1),TIME=5
//PROCESO PROC
//PASO00  EXEC DESNAT
...
//PASO01  EXEC PGM=PPP0,TIME=(1,30)
...
//        PEND
//* ------------------ fin procedimiento
```

```
//PASO1    EXEC PGM=PROGRAM1,TIME=(,30)
...
//PASO2    EXEC PGM=PROGRAM2,TIME=3
```

CAPÍTULO 7. ENUNCIADO DD

Además de indicarle al sistema operativo cuáles son los programas que queremos que ejecute (con //EXEC PGM= ... etc.), también debemos identificar a los archivos de datos que queremos que utilice el programa, con el enunciado definición de datos DD podemos hacerlo, ya que se usa para describir cada archivo al que queremos que acceda el programa.

El enunciado DD se usa para identificar el nombre del archivo de datos que se va a usar, el tipo de unidad de entrada y/o salida para el archivo, el volumen en el que reside el archivo de datos y si el archivo ya esta creado o si ya existe. Y solamente puede ser declarada en nivel de paso.

Este enunciado permite la flexibilidad de identificar y definir al archivo de datos y a su ubicación cuando el programa está procesado en lugar de hacerlo cuando está siendo escrito, esta capacidad ofrece una gran ventaja al programador ya que un JOB puede usar un archivo diferente cada vez que es ejecutado. Esto es, el programador codifica cualquier NOMBREdd en el programa. El codificador de JCL debe usar dicho NOMBREdd en una sentencia DD para dicho programa y luego describir al archivo de datos real que se usará para la ejecución en curso.

Este enunciado se codifica de la siguiente manera:

//NOMBREdd DD parametros

Ejemplo:

```
//JOBPROC   JOB (000),CLASS=A,
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*       EJECUTA UN PROGRAMA CON UN ARCHIVO DE ENTRADA         *
//*-------------------------------------------------------------*
//PASO0010  EXEC  PGM=PROGRAM1,PARM=(12,23),REGIO=512K,
//                                TIME=MAXIMUM,COND=(0,LT)
//ENTRADA DD DSN=DSINFB.BGEPR.DETRASP.BPI.DIARIO,DISP=SHR
. . .
```

ENTRADA es la “etiqueta” con la cual se liga al nombre físico del archivo con el nombre lógico del programa COBOL. Así pues este archivo se declara dentro del programa en la capa Input-Output Section (donde se describen las características de cada uno de los archivos que serán usados por el programa tanto de entrada como de salida) .

Ejemplo:

```
Input-Output Section.

SELECT IN-FILE ASSIGN TO E1DQ0001
...
```

Otra virtud de DD es que se pueden concatenar los archivos, supongamos que tenemos 3 archivos de entrada con el mismo formato y tamaño de registros, pero contienen información diferente además cada uno de los archivos son el resultado de la ejecución anterior de 3 JOB’s diferentes, esto se resuelve solo declarando el NOMBRE DD del archivo y se declaran los n archivos restantes con su enunciado DD y parámetros.

Ejemplo:

```
//JOBPROC   JOB (000),CLASS=A,
```

```
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*       EJECUTA UN PROGRAMA CON 3 ARCHIVOS DE ENTRADA         *
//*-------------------------------------------------------------*
//PASO0010  EXEC  PGM=PROGRAM1,PARM=(12,23),REGIO=512K,
//                                TIME=MAXIMUM,COND=(0,LT)
//ENTRADA DD DSN=DSINFB.BGEPR.JOBEJM1.BPI.DIARIO,DISP=SHR
//        DD DSN=DSINFB.BGEPR.JOBEJM2.BPI.DIARIO,DISP=SHR
//        DD DSN=DSINFB.BGEPR. OBEJM3.BPI.DIARIO,DISP=SHR
. . .
```

DSN

DSN o DSName es el nombre con que el sistema debe localizar o alojar los archivos. El administrador del sistema puede restringir los nombres de los archivos a ciertos estándares preestablecidos, por lo que no puede usarse cualquier nombre.

Existen dos tipos de nombres:

- **Unqualified name** nombre de 1 a 8 caracteres alfanuméricos nacionales. El primer carácter debe ser una letra o un carácter nacional, ejem: DSN=ARCHIVO1
- **Qualified name** Son múltiples nombres tipo unqualified name unidos por puntos. La máxima longitud de un qualified name son 44 caracteres incluyendo los puntos, ejem: DSN=TRE00.DP.A000.ARCH1

El formato de este parámetro es:

(1) DSN=nombre_fichero
(2) DSN=librería(miembro)
(3) DSN=nombre_fichero(numero_versión)

(4) DSN=nombre_temporal

(5) DSN=*.nombre_DD

(6) DSN=*.nombre_paso.nombre_DD

(7) DSN=*.nombre_paso.Nombre_paso_proc.nombre_DD

(8) DSN=NULLFILE

Ejemplo:

```
//JOBPROC   JOB (000),CLASS=A,
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*       EJECUTA UN PROGRAMA CON UN ARCHIVO DE ENTRADA         *
//*-------------------------------------------------------------*
//PASO0001 EXEC  PGM=PROGRAM1,PARM=(12,23),REGIO=512K,
//                                TIME=MAXIMUM,COND=(0,LT)
//ARCHIVO1 DD DSN=DSINFB.BGEPR.DETRASP.BPI.DIARIO,DISP=SHR
. . .
```

Ejemplo formato (3):

Especifica el número de versión para los ficheros generacionales GDG. Como se menciono en el capitulo 3, un fichero generacional es aquél que bajo el mismo nombre mantiene diversas versiones, hasta 255. Estas versiones se referencian de la siguiente manera:

(0) la ultima versión

(+1)la próxima versión

(-1) la versión anterior

La longitud máxima del nombre de un fichero GDG es de 35 caracteres.

```
//JOBPROC   JOB (000),CLASS=A,
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
```

```
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*       EJECUTA UN PROGRAMA CON ARCHIVOS GDG                  *
//*-------------------------------------------------------------*
//COPIAF01 EXEC PGM=IEBGENER
//SYSUT1 DD DSN=DESJMP.HISTORIA(0),DISP=SHR
//       DD DSN=DESJMP.MVTOS,DISP=(OLD,DELETE,KEEP)
//SYSUT2 DD DSN=DESJMP.HISTORIA(+1),
//            DISP=(,CATLG,DELETE),
//            DATACLASS=DFBXP,LRECL=133
//SYSIN    DD DUMMY
//SYSPRINT DD DUMMY
```

DISP

El parámetro DISP, Describe al sistema el estatus de un archivo, además tiene tres funciones especificadas en los sub parámetros. El primer indicador nos dice el estado del archivo en el momento de la ejecución, el segundo indicador nos dice que acción se debe de tomar con el archivo si el paso termina correctamente. Por último, el tercer indicador nos dice que acción se debe de tomar con el archivo si el paso termina anormalmente.

Los diferentes disposiciones de los archivos, en el primer indicador son:

- NEW: El archivo es creado.
- OLD: El archivo ya existe y mientras se esta utilizando ninguna otra tarea puede acceder a dicho archivo.
- SHR: El archivo existe y además puedes ser compartido. Sólo lectura.
- MOD: Si el archivo existe ninguna otra tarea puede accesarlo y si el archivo no existe lo considera NEW. En la adición de registros nuevos, obliga al sistema de I/O a posicionarse en el último registro que exista.

Las diferentes disposiciones de los archivos, en el segundo y tercer indicador, son:

- DELETE: EL archivo es borrado.
- KEEP: Indica que el archivo debe mantenerse al final del paso.
- PASS: Indica que el archivo se utilizará en los pasos posteriores. Se borra al terminar el JOB.
- CATLG: El archivo se guarda y es ctalogado.
- UNCATLG: El archivo se guarda y no es catalogado.

Si el primer parámetro no es codificado se asume NEW. Si el segundo parámetro no es codificado se asume DELETE, si en el primer parámetro se informa NEW. Si el segundo parámetro no es codificado se asume KEEP, sólo si en el primerparámetro se informa OLD.

El formato de este parámetro es:

DISP=(dis_inicial,disp_fin_normal,disp_fin_anormal)

Ejemplo:

```
//JOBPROC   JOB (000),CLASS=A,
//                  REGION=0M,
//                  MSGLEVEL=(1,1)
//                  MSGCLASS=X,
//                  NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*                  EJECUTA  PROGRAM2                          *
//*-------------------------------------------------------------*
//PASO0001 EXEC  PGM=PROGRAM2,PARM=(12,23),REGIO=512K,
//                               TIME=MAXIMUM,COND=(0,LT)
//ARCHIVO2 DD DSN=DSINFB.BGEPR.DETRASP.BPI.DIARIO,DISP=SHR
//ARCHIVO3 DD DSN=DSINFB.BGEPR.RMNFRDJ.BPI.DIARIO,
//            DISP=(NEW,CATLG,DELETE)
```

```
//ARCHIVO4 DD DSN=DSINFB.FGTYH.DELANTH.BPI.DIARIO,
//            DISP=(OLD,DELETE,DELETE)
```

Donde:

- El ARCHIVO2 solamente será utilizado para ser leído.
- El ARCHIVO3 se esta creando y es común utilizar los valores CATLG y DELETE en el segundo y tercer parámetro respectivamente.
- EL ARCHIVO4 se borra y es recomendable usar los valores que se han declarado en el ejemplo.

UNIT

Con el parámetro UNIT, se solicita al sistema que aloje el archivo en algún dispositivo específico o en un cierto tipo o grupo de volúmenes reservados. Las unidades asignadas para la creación de archivos pueden variar de ambiente en ambiente y a través del tiempo. Es responsabilidad del constructor del JCL averiguar el nombre de las unidades que debe utilizar en el ambiente de trabajo (pruebas, preproducción y/o producción). Este parámetro depende de la instalación.

Un grupo de volúmenes reservados es un nombre único que se le da a un conjunto de dispositivos de almacenamiento. ejem: Los discos DPRC9O, DPRC91 y DPRC92 son agrupados con un sólo nombre: DESAPASS. Si en la creación de un archivo se codifica UNIT=DASAPASS, el sistema decidirá en qué disco, dentro del grupo DESAPASS, se almacenará el archivo. Cuando se quiere hacer referencia a un dispositivo en específico y no a un grupo de volúmenes reservados, la codificación del parámetro UNIT debe ir acompañado por el parámetro *VOL =SER* de la siguiente forma:

UNIT= (Tipo de dispositivo),V0L=SER=(identificador del dispositivo)

Ejemplo:

```
//JOBPROC   JOB (000), CLASS=A,
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*                   EJECUTA  PROGRAM3                         *
//*-------------------------------------------------------------*
//PASO0001 EXEC  PGM=PROGRAM3,PARM=(12,23),REGIO=512K,
//                           TIME=MAXIMUM,COND=(0,LT)
//BGE13402 DD DSN=DSIFNB.BGEPR.DETRASP.BPI.DIARIO,
//         DISP= (NEW, CATLG, DELETE),
//              UNIT=3390,VOL=SER=DPRC93,
//              SPACE=(CYL,(100, 0),RLSE),
//         DCB=(RECFM=FB,LRECL=49,BLKSIZE=0,DSORG=PS)
. . .
```

Donde:

- VOL, se utiliza para identificar el disco en el que se encuentra el archivo (Para archivos catalogados no es necesario se codifique esta sentencia).
- SER, indica el número de serie del disco. Normalmente, tiene el mismo valor que el volumen.

VOLUME

Este parámetro especifica el volumen en el que residirá el fichero. Este parámetro es opcional, ya que el sistema, en el caso de que no se especifique, hará la asignación del espacio solicitado en cualquiera de los volúmenes que en ese momento se encuentren montados. También puede usarse la palabra abreviada VOL.

El sintaxis de este parámetro es:

VOL=([PRIVATE][,RETAIN][,sec][,cuenta][,ser_ref])

Donde:

- PRIVATE: especifica que el volumen es un volumen privado, y por tanto éste se asigna como de utilización exclusiva para el paso en el que figura este parámetro.
- RETAIN: especifica que el volumen en cinta no se desmonte, ya que de no especificar este parámetro, el volumen se desmontará cuando acabe el paso.
- Sec: especifica la posición del fichero dentro del volumen de cinta montado.
- Cuenta: Número de volúmenes que necesita un fichero multi volumen, es decir, aquel fichero que ocupa varias cintas.
- Ser_ref: Este parámetro puede tomar los siguientes formatos:

    - SER=num_serie Especifica el número de serie del volumen.
    - REF=nom_fichero Con este formato le indicamos al sistema que se desea asignar para el nuevo fichero el mismo volumen que el que tiene asignado el fichero cuyo nombre especificamos en este parámetro.
    - REF=*.nombre_DD Es parecido al anterior, con la diferencia que en este caso especificamos el fichero mediante el nombre de la DD que le referencia.

En el caso de que se necesite especificar mas de un número de serie, estos se especificaran separándolos por comas, y encerrándolos entre paréntesis para seguir las normas generales de codificación de parámetros posicionales.

Ejemplos:

El siguiente paso de JOB muestra cómo se puede copiar un fichero sobre cinta. En este caso son varios ficheros físicos los que se copian, aunque, al tener DDs concatenadas, son todos considerados como un único fichero lógico. El fichero de salida puede tener hasta 30 volúmenes o cintas.

```
//*****************************************************
//* COPIAR FICHERO DE DISCO A CINTA
//*- - - - - - - - - - - - - - - - - - - - - - - - - - -
//PASO1    EXEC PGM=IEBGENER
//STEPLIB  DD DSN=SYS1.LINKLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//*      ----------- FICHERO DE ENTRADA
//SYSUT1   DD DSN=DESJMP.WTZZ0D0X.PRO,DISP=OLD
//         DD DSN=DESJMP.WTZZ0D0X.PRO@01,DISP=OLD
//         DD DSN=DESJMP.WTZZ0D0X.PRO@02,DISP=OLD
//         DD DSN=DESJMP.WTZZ0D0X.PRO@03,DISP=OLD
//*      ----------- FICHERO DE SALIDA
//SYSUT2   DD DSN=DESJMP.WTZZ0D0X.PRO.REPORT11,DISP=(NEW,CATLG,),
//         UNIT=(ACL1,,DEFER),VOL=(,,,30),LABEL=(1,SL),
//         DCB=*.SYSUT1
//SYSIN    DD DUMMY
```

El siguiente ejemplo ejecuta la utilidad IEBCOPY. Esta utilidad ejecuta lo que se le indica en el fichero que contiene los comandos de entrada, identificado por la DDname SYSIN. Lo que se dice a través de esta entrada es que copie la DDname indicada en IN-DD sobre la DDname especificada en OUT-DD. En este caso se especifica que no desmonte el volumen porque, posiblemente, se quiera usar el fichero siguiente.

```
//*****************************************************
//PASO1     EXEC PGM=IEBCOPY
//STEPLIB   DD DSN=SYS1.LINKLIB,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//*         ----------- FICHERO DE ENTRADA
//LIB1      DD DSN=LINKLIB,DISP=(OLD,PASS),UNIT=482,
//           LABEL=(1,SL),VOL=(,RETAIN,SER=ENE001)
//*         ----------- FICHERO DE SALIDA
```

```
//LIB2      DD DSN=DESJMP.SECLOAD,DISP=SHR
//SYSIN     DD *
  COPY INDD=LIB1,OUTDD=LIB2
/*
//*************************************************
```

DCB

Cada archivo que usa un programa es descrito por un bloque de control de datos (DCB) ubicado dentro del programa. El DCB es una tabla que se incorpora al programa de aplicación cuando se compila. El sistema operativo completa la información del DCB cuando se abre el archivo. El sistema operativo necesita esta información referente a las características del archivo para poder procesar los datos en forma correcta.

El DCB describe varios atributos de un archivo, aquí se discutirán los tres que se necesitan con más frecuencia: La medida del bloque (Blocksize) y la medida del registro (logical record length).

Antes de mencionar estos atributos debemos saber algunos conceptos:

- Bloques o Registros Físicos: Los datos que están en un dispositivo de acceso directo o en una cinta se almacenan en registros físicos. Usualmente se almacenan varios registros lógicos en un único registro físico. El registro físico es también llamado BLOQUE. El sistema operativo transfiere todos los datos de un bloque a la vez, entre la memoria y el dispositivo. Bloquea y desbloquea estos datos de forma tal que al programa se le presenta un registro lógico por vez. Por lo tanto el programa se debe ocupar solamente de los registros lógicos.

- Registros Lógicos: Son los datos transferidos por una única instrucción de entrada o salida por un programa. Puede representar una única imagen de tarjeta, una línea de impresión o cualquier cantidad lógica de datos.

Una vez aclarado los conceptos de bloques y registro lógico continuaremos con DCB y sus parámetros. Dentro de la información que se define esta:

- RECFM (Formato del registro). Especifica el formato y características en que los registros del nuevo archivo serán almacenados. En general, para este parámetro pueden definirse diferentes valores:

    - RECFM=F (Fixed Length Records) Todos los registros en el archivo tienen la misma longitud. Se almacena un registro por bloque (ver BLKSIZE) por lo que se desperdicia espacio. Su uso es poco común.
    - RECFM = V (Variable Length Records) No todos los registros en el archivo tienen la misma longitud. Se almacena un registro por bloque (ver BLKSIZE) por lo que se desperdicia espacio. Su uso es poco común.
    - RECFM=U (Undefined Length Records) Indica que los registros tienen una longitud indefinida (Valor por default).
    - RECFM=FB (Fixed Length Records Blocked) Todos los registros en el archivo tienen la misma longitud. Se almacenan varios registros por bloque (ver BLKSIZE) por lo que no se desperdicia espacio.
    - RECFM= VB (Variable Length Records Blocked) No todos los registros en el archivo tienen la misma longitud. Se almacenan varios registros por bloque (ver BLKSIZE) por lo que no se desperdicia espacio.

Su sintaxis es la siguiente:

//NOMBREDD DD DCB=(RECFM = parámetro)

- LRECL. Especifica la longitud de registro lógico para cada registro del archivo. La longitud de registro lógico es el número de bytes que lee o graba el programa en cada operación de entrada y/o salida.

  Su sintaxis es la siguiente:

  //NOMBREDD DD DCB=(LRECL = número)

- BLKSIZE. Especifica el número de bytes del bloque. Este valor se debe calcular de la manera más óptima posible para disminuir al mínimo el desperdicio de espacio en el dispositivo de almacenamiento. El cálculo depende del tipo de dispositivo que se usa pero si se omite la codificación del parámetro o se define el parámetro igual a cero el MVS determina el tamaño óptimo del bloque tanto en DASD como en cinta o cartuchos.

  Su sintaxis es la siguiente:

  //NOMBREDD DD DCB=(BLKSIZE = longitud)

Ejemplos:

El siguiente ejemplo muestra 2 DCBs típicas de un fichero en cinta y otro en disco. Sólo hay que ajustar los valores del tamaño del bloque en función del tamaño del registro, lo cual, dependiendo de las instalaciones, se suele hacer usando el comando de TSO BLKSIZE, aunque en otras instalaciones se usa SPACE2, o alguno parecido.

```
//JOBPROC   JOB (000),CLASS=A,
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*                         EJEMPLO DCB                          *
//*-------------------------------------------------------------*
//EXEC PGM= PROGRAMA
//CMWKF04 DD DSN=DESJMP.WRK49406.D22,DISP=(NEW,CATLG,CATLG),
//        UNIT=(ACL1,,DEFER),VOL=(,,,30),LABEL=(1,SL),
//        DCB=(RECFM=FB,LRECL=4538,BLKSIZE=18152)
//*
//SYSUT2  DD DSN=DESJMP.WTZZ0D0X.PRO.XXX,DISP=(NEW,CATLG,),
//        UNIT=SYSDA,SPACE=(CYL,(4,1),RLSE),
//        DCB=(LRECL=33,BLKSIZE=23464,RECFM=FB)
. . .
```

SPACE

Este parámetro pide el espacio en la memoria de acceso directo para un archivo de datos. El codificador de JCL primero debe determinar el tipo de asignación que va a pedir ya sea bloque, pista(TRK) o cilindro (CYL) y luego de esto debe calcular la cantidad de espacio necesaria para el archivo.

La sintaxis del parámetro SPACE es:

```
//NOMBREDD DD SPACE=(TRK,(primario, secundario, RLSE))
```

Donde:

Primario: Especifica el número de unidades que se asignan inicialmente al fichero.

Secundario: Especifica el número de unidades que se asignaran al fichero en el caso de que no fuera suficiente la cantidad primaria asignada.

El sistema hará hasta 16 extensiones de un mismo fichero en cada volumen, dando en cada extensión la cantidad especificada como cantidad secundaria. Esto esta establecido así a fin de no reservar espacio innecesariamente, e ir ampliando el fichero cuando se necesite.

RLSE: Especifica el sistema que cuando se cierra el fichero se libere el espacio que no haya sido utilizado. Para seguir las normas generales de los parámetros posicionales, en el caso de no especificarse este parámetro, debe escribirse una coma, si es que siguen otros parámetros.

Ejemplo:

```
//JOBPROC   JOB (000),CLASS=A,
//                    REGION=0M,
//                    MSGLEVEL=(1,1)
//                    MSGCLASS=X,
//                    NOTIFY=USUARIO
//*-------------------------------------------------------------*
//*                     EJEMPLO SPACE                           *
//*-------------------------------------------------------------*
//EXEC PGM= PROGRAMA
//CMWKF04 DD DSN=DESJMP.WRK49406.D22,DISP=(NEW,CATLG,CATLG),
//        UNIT=(ACL1,,DEFER),VOL=(,,,30),LABEL=(1,SL),
//        DCB=(RECFM=FB,LRECL=4538,BLKSIZE=18152)
//*
//SYSUT2  DD DSN=DESJMP.WTZZ0D0X.PRO.XXX,DISP=(NEW,CATLG,),
//        UNIT=SYSDA,SPACE=(CYL,(4,1),RLSE),
//        DCB=(LRECL=33,BLKSIZE=23464,RECFM=FB)
. . .
```

SYSOUT

Usualmente la salida de los programas incluye cierta información impresa. Una instalación generalmente tiene muchos programas que están ejecutando en forma concurrente, y varios otros esperan para imprimir su salida, pero un número limitado de dispositivos de impresión. En vez de dejar que los programas esperen que haya una impresora disponible el sistema operativo (subsistema de entrada de trabajo) controla el uso de los dispositivos de registro

unitario (dispositivos de tarjeta e impresora). Este control se realiza cuando el JCL del usuario especifica que se desean registros de SYSOUT.

Los registros de SYSOUT son cualquier salida que será impresa bajo el control del subsistema de entrada de trabajo. Cuando se especifica SYSOUT como un operando de una sentencia DD, está indicando que los datos creados para este archivo van a ser impresos. El subsistema de entrada del trabajo ubica temporalmente a esos datos en una unidad llamada SPOOL. El subsistema de entrada del trabajo los toma del archivo de SPOOL, y los imprime.

Los archivos de salida del trabajo que se ubican en la SPOOL se agrupan por clasificación, por ejemplo, a los archivos que requieran una salida en tarjeta perforada se les puede dar una clasificación diferente de la que se le dará a los que se transcribirán en una impresora.

La sintaxis de este parámetro es:

SYSOUT = *
SYSOUT = (clase, programa, formulario)

Dónde:

- Clase: especifica la clase de la cola de salida del SPOOL a la que se envía el fichero para su impresión. En el caso de que se especifique * la salida se envía a la misma clase a la que se envían los mensajes del JOB, y que se ha definido en la sentencia JOB con el parámetro MSGCLASS.
- Programa: En el caso de que dicha salida deba ser tratada por algún programa para escribirle, ese será el nombre del programa que se especifique aquí. Existen dos nombres especiales el INTRDR nemónico de INTernal ReaDeR y STDWTR nemónico de STandarD WriTeR.

- Formulario: Este es el nombre del formulario que debe usarse para imprimir la salida. Este nombre estará compuesto por una palabra de hasta 4 caracteres.

Cuando se especifica INTRDR lo que le indicamos al sistema es que el fichero de salida sea enviado a la entrada del JES ya que el resultado es un conjunto de fichas de control. Es decir, es un JOB.

```
//JMP15    JOB (123,456),'DESARROLLO',CLASS=X,
//         MSGLEVEL=(1,1),NOTIFY=T976614
//*-------------------------------------------------------------*
//*                   EJEMPLO SYSOUT                            *
//*-------------------------------------------------------------*
//PASO1    EXEC PGM=IEBGENER
//SYSUT2   DD SYSOUT=(A,INTRDR)
//SYSUT1   DD DATA,DLM='$$'
//JMP15C  JOB (999,H,234),'PRODUCCION',CLASS=Q,
//            MSGLEVEL=(1,0)
...
/*
//PASO2
//SYSUT2  DD SYSOUT=*
//SYSUT1  DD *
...
/*
```

CAPÍTULO 8. UTILIDADES DEL JCL

Las utilidades son programas que proporciona el sistema operativo para facilitar ciertas tareas al usuario. Estos programas son usadas para realizar funciones en común como mantener y manipular datos del sistema, reorganizar, cambiar, comparar o manipular datos y ficheros de usuario. Los programas de utilidad se controlan mediante dos tipos de sentencias, las sentencias de JCL y las sentencias de utilidad.

IEBGENER

Esta utilería se utiliza sobre todo para crear, copiar o imprimir ficheros secuenciales y particionados. Copia datos de un dispositivo a otro, crea un PDS a partir de un fichero secuencial, amplia o incorpora miembros a un PDS, crea ficheros editados, cambia la distribución en bloques o modifica las longitudes de registro lógicos de un fichero. El siguiente código es como normalmente se declara la utilería IEBGENER.

```
//PASO1 EXEC PGM=IEBGENER
//* Sentencia DD que suele estar en la mayoría de los programas
//* de utilidades. En SYSOUT indicamos la clase de salida de los
//* mensajes de utilidad
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD ...
//SYSUT2 DD ...
//SYSIN DD *
      SENTENCIA DE CONTROL DE UTILIDAD
/*
```

Donde:

- SYSUT1 define el fichero de entrada.
- SYSUT2 define el fichero de salida y puede codificarse como salida en fichas, salida impresa, un fichero secuencial o un fichero particionado.
- SYSIN define la entrada de las sentencias de control de la utilidad. La sentencia SYSIN DD puede codificarse como DUMMY si el programa (IEBGENER) no requiere edición Ejemplo: //SYSIN DD DUMMY. Si se codifica la sentencia SYSIN DD como DUMMY, todo el fichero de ENTRADA se copia secuencialmente. SYSIN contendrá las fichas de control GENERATE y RECORD.

  GENERATE tiene un único parámetro MAXFIELDS que indica que la ficha de control RECORD va a poseer n parámetros FIELD.

  RECORD es un tipo editor con el que especificamos el diseño del registro de salida mediante la concatenación de n parámetros FIELD, su codificación es la siguiente:

  FIELD=(num_bytes,pos_ini,conv,pos_sal)

  Donde:

  num_bytes: Tamaño del campo
  pos_ini : Posición inicial en el registro de entrada
  conv : Posible conversión
  pos_sa : Posición del registro de salida donde depositarla

El siguiente ejemplo muestra cómo copiar un fichero editando la salida de modo que: En la posición 1 del registro de salida se coloquen los 30 octetos que se encuentran en la posición 1 y siguientes del registro de entrada. Y En la posición 50 del registro de salida se coloquen los 175 bytes que se encuentran en la posición 31 y siguientes del registro de entrada. Los octetos 31 a 49 ambos inclusive del registro de salida quedaran rellenos a ceros binarios.

```
//PASO1 EXEC PGM=IEBGENER
//STEPLIB DD DSN=SYS1.LINKLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//*FICHERO DE ENTRADA
//SYSUT1    DD DSN=DESJMP.FILE01,DISP=SHR
//*FICHERO DE SALIDA
//SYSUT2    DD DSN=DESMGO.CMWKF01,
//          DISP=(NEW,CATLG,DELETE),
//          UNIT=SYSDA,SPACE=(CYL,(4,1),RLSE),
//          DCB=(DMYDCB,DSORG=PS,RECFM=FB,
//               LRECL=205,BLKSIZE=23370)
//SYSIN     DD *
  GENERATE MAXFLDS=2
  RECORD   FIELD=(30,1,,1),FIELD=(175,31,,50)
/*
```

IEBCOPY

Esta utilería sirve para copiar miembros de particionados a otro PDS o para fusionarlos, para copiar de un PDS a un secuencial (descarga), copia de uno o más ficheros secuenciales creados por una operación de descarga a un PDS (carga) y compresión de un PDS: copiándolo sobre si mismo. El formato para codificar la utilería es el siguiente:

```
//PASO1 EXEC PGM=IEBCOPY
//* Sentencia DD que suele estar en la mayoría de los programas
//* de utilidades. En SYSOUT indicamos la clase de salida de los
//* mensajes de utilidad
 //SYSPRINT DD SYSOUT=*
//NOMBRE1 DD ...
//NOMBRE2 DD ...
...
//NOMBREN DD ...
//SYSIN DD *
      COPY OUTDD=NOMBRE1
            INDD=(ONMBREDD,(NOMBREDD,R),...)
      SELECT  MEMBER=(NOMBRE1,NOMBRE2,...)
      EXCLUDE MEMBER=(NOMBRE1,NOMBRE2,...)
/*
```

Donde:

NOMBRE1, NOMBRE2 : fichero de entrada.
NOMBREN: fichero temporal
COPY: Indica al programa cuales son las DDnames de entrada y salida, mediante los parámetros INDD (entrada) y OUTDD (salida)
SELECT: Indica que miembros queremos copiar por medio del parámetro MEMBER.
EXCLUDE: Indica que miembros se quieren excluir de la copia mediante el parámetro MEMBER.

En el siguiente ejemplo se copia toda la librería de entrada sobre el fichero secuencial especificado como salida. Las DDnames SYSUT3 y SYSUT4 son definiciones que puede necesitar para trabajo el propio programa, y que en caso de no especificarlas en el JCL, cuando se interrumpe el trabajo por necesitarlas, presenta en el log del JOB el mensaje explicativo de que no se ha especificado la SYSUTn correspondiente.

```
//PASO3     EXEC PGM=IEBCOPY
//SYSPRINT  DD SYSOUT=*
//SYSUT3    DD UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSUT4    DD UNIT=SYSDA,SPACE=(TRK,(1,1))
//SEC@IN    DD DSN=DESJMP.ENTRADA,DISP=SHR
//SEC@OUT   DD DSN=DESJMP.SALIDA,DISP=OLD
//PDS@IN    DD DSN=DESJMP.ENTRADA,DISP=SHR
//PDS@OUT   DD DSN=DESJMP.SALIDA,DISP=OLD
//SYSIN     DD *
  COPY INDD=PDS@IN   OUTDD=SEC@OUT
/*
```

IEBCOMPR

El programa utilería EIBCOMPR nos va a servir para comparar dos archivos, tanto dos secuenciales como dos particionados. Dos archivos secuenciales los compararemos directamente. En el caso do los PDS no mira el contenido de los miembros, sino el de los directorios.

Por lo tanto IEBCOMPR considera iguales dos ficheros secuenciales si contienen el mismo número de registros y si cada registro tiene la misma información. Y para archivos PDS los considera iguales si los directorios se los archivos son iguales, si los miembros correspondientes de los dos ficheros son iguales.

El siguiente ejemplo muestra como se codifica normalmente esta utilería:

```
//PASO1 EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=*
//*FICHERO DE ENTRADA
//SYSUT1    DD DSN=DESJMP.FILE01,DISP=SHR
//*FICHERO DE ENTRADA
//SYSUT2    DD DSN=DESMGO.CMWKF01,
//          DISP=SHR
//SYSIN     DD *
  COMPARE=TYPORG=PS
/*
```

Donde:

SYSPRINT: archivo de salida de mensaje de IEBCOMPR.

SYSUT1 y SUSUT2: Sentencia DD para asignación de ficheros.

SYSIN: Sentencia DD para entrada de sentencias de control de utilidades.

COMPARE: Indica la función a realizar, en este caso es comparar.

TYPORG: Indica la organización del archivo, para lo cual puede tomas el valor de PS (secuencial) o PO (particionado).

IEFBR14

Esta utilidad es un comodín, sirve para catalogar, crear y borrar archivos por medio de las sentencias DD y parámetros DISP. Un ejemplo de cómo se usa esta utilidad es el siguiente:

```
//BORRAR    EXEC PGM=IEFBR14
//STEPLIB   DD DSN=SYS1.LINKLIB,DISP=SHR
//D1        DD DSN=DESJMP.CMWKF01,DISP=(MOD,DELETE,)
//          UNIT=SYSDA,SPACE=(CYL,(4,1),RLSE),
//          DCB=(DSORG=PS,RECFM=FB,LRECL=205,BLKSIZE=23370)
//* COPIAR FICHEROS //*
//PASO1     EXEC PGM=IEBGENER
//STEPLIB   DD DSN=SYS1.LINKLIB,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//* FICHERO DE ENTRADA
//SYSUT1    DD DSN=DESJMP.FILE01,DISP=SHR
//* FICHERO DE SALIDA
//SYSUT2    DD DSN=DESJMP.CMWKF01,
//          DISP=(NEW,CATLG,DELETE),
//          UNIT=SYSDA,SPACE=(CYL,(4,1),RLSE),
//          DCB=(DSORG=PS,RECFM=FB,LRECL=205,BLKSIZE=23370)
//SYSIN DD DUMMY
```

El primer paso, ejecuta el programa IEFBR14 y la única DD que existe, esta asociada a un fichero con disposición MOD, lo cual quiere decir que:

- Si el fichero no existe, debe crearle con las especificaciones que siguen (DCB,UNIT,SPACE).
- Si el fichero existe, los datos se apegan a los ya existentes. (Este programa no genera información).

A continuación, se ejecuta el programa, y termina, con lo que el fichero anterior, en base al parámetro DISP, se borra, pues ha terminado bien. En una palabra, este paso asegura que cuando se termina de ejecutar no existe el fichero de la DD especificada en el mismo.

DFSORT

Este programa es utilizado para copiar los registros de un archivo de entrada en otro de salida, cambiando la secuencia en a que se encuentran estos. Los DDnames necesarios para ejecutar este programa son:

El programa SORT tiene varias sentencias de control:

- SORT: Indica los parámetros para la clasificación del archivo.
- MERGE: Esta sentencia es igual que la sentencia SORT, con la única diferencia de que en este caso se mezclan registros de hasta 16 archivos en lugar de los registros de un único archivo.
- RECORD: Indica el formato del registro de salida, en caso que se quiera editar el archivo de entrada.
- INCLUDE: Esta sentencia especifica los campos que permiten seleccionar los registros del archivo en un sort.
- OMIT: Esta sentencia especifica los campos que permiten omitir los registros del archivo en un sort.
- INREC: Esta sentencia reformatea el registro de entrada antes de ejecutar el sort, de modo que solo se ejecute la clasificación con aquellos campos que interesan.
- OUTREC: Esta sentencia reformatea el registro de salida después de ejecutar el sort.
- SUM: Esta sentencia especifica al programa que escriba en el fichero de salida un registro totalizando los n registros de igual clave del fichero de entrada.

A continuación se muestra un ejemplo de cómo se codifica DFSORT:

```
//ORDENAR  EXEC PGM=SORT
//SYSOUT   DD SYSOUT=*
//SYSIN    DD *
//SORTIN   DD DSN=...
//SORTOUT  DD DSN=...
  SORT FIELDS=(1,5,CH,A,7,8,CH,D)
  INCLUDE COND=(1,5,CH,EQ,C'LOPEZ')
  SUM FIELDS=(14,9,PD)
  OUTREC FIELDS=(1,100)
//SORTWKnn DD UNIT=SYSDA,SPACE=(CYL,1,1))
```

Como podemos observar SORT va a continuación de la sentencia EXEC. El programa que ejecuta esta utilería es SORT, y dependiendo de la instalación será invocado como programa o procedimiento.

Los mensajes de salida que fueron generados en la ejecución son dejados en el archivo de SYSOUT. Estos mensajes contienen información acerca de errores de la codificación, número de registros de entrada y de salida. SORTIN define el archivo de entrada que se desea ordenar y SORTOUT define el archivo de salida en el que se desea guardar el archivo ya ordenado y reformateado.

Los enunciados de control se codifican de la columna 2 a la 71 (se deja al menos un espacio antes de codificar el enunciado), si el espacio entre la columna 2 y 71 no es suficiente para codificar el enunciado de control, se puede continuar en la siguiente línea. Si al final de una línea hay una coma y un espacio, la siguiente línea se toma como continuación, la línea de continuación debe codificarse también entre las columnas 2 y 71.

El enunciado SORT FIELDS se utiliza para indicar los campos por los que se desea ordenar el archivo y el orden que debe establecerse.

La información del archivo se ordena de acuerdo a:

SORT FIELDS=(P,N,T,O)

Donde:

P Posición de inicio del campo llave

N Número de caracteres del campo llave

T Tipo de campo (CH = character, PD= packed decimal, ZD=zoned decimal, BI = Binario

O Forma de ordenamiento (A = ascendente, D = descendente)

Usualmente se usa el código EBCDIC (Extended Binary Coded Decimal Interchange Code) para definir la secuencia de ordenamiento.

Para nuestro ejemplo particular en la linea de código:

SORT FIELDS=(1,5,CH,A,7,8,CH,D)

En la primera posición se indica el comienzo del registro, a partir de donde empieza el campo por el que se quiere ordenar, en este caso el número “1” y “7”. A partir de la segunda posición se indica la longitud en bytes, del campo por el que se quiere ordenar el número “5” y “8”. Después el carácter “CH” indica el formato del campo por el que se quiere ordenar en nuestro caso es un carácter. Por último para la sentencia SORT FIELDS es importante indicar el tipo de ordenación, las cuales pueden ser A (ascendente) o D (Descendente).

Sigamos con la siguiente línea de código:

INCLUDE COND=(1,5,CH,EQ,C'LOPEZ')

Como se había mencionado a través de la sentencia INCLUDE se puede realizar la selección de registros que cumplan ciertas condiciones, o bien, realizar lo

contrario con la sentencia OMIT. Las sentencias de INCLUDE y OMIT no pueden ser codificadas a la vez. En la primera posición se indica el comienzo del registro, es decir, la primera posición a partir de donde empieza el campo que se quiere incluir en este caso es “1”. La segunda posición indica la longitud del campo al que se hace referencia el número “5”. La tercera posición indica el formato del campo “CH” carácter. En la cuarta posición se indica el operador de comparación (EQ,NE,GT,GE, LT, LE) y a continuación la constante de control.

La línea de código para sumar es la que a continuación se presenta:

SUM FIELDS=(14,9,PD)

La sentencia SUM permite sumar los valores de un campo, la primera posición “14” indica el comienzo de campo a sumar y la segunda “9” la longitud de dicho campo, continuada por el formato. Los campos a sumar deben ser numéricos, empacados o decimales. El resultado de la suma es guardado en un registro y el resto son borrados.

La siguiente línea es :

OUTREC FIELDS=(1,100)

A través de la sentencia OUTREC, se puede realizar el reformateo de registros después de ser ordenados. Se debe codificar la localización y longitud de los campos que van a aparecer en el registro de salida, el resto de campos no especificados son eliminados, la sentencia OUTREC permite la inserción o borrado de campos en blanco en el registro. Su formato es:

OUTREC FIELDS=(*nn*X*,1,100,nn*X)

Donde

nn es el número de caracteres a insertar

X indicador de carácter en blanco

Por ejemplo:

20X inserta 20 caracteres blancos antes del campo
10X inserta 10 caracteres blancos después del campo

La sentencia INREC tiene la misma función que la sentencia OUTREC, con la diferencia que realiza el reformateo antes de ser ordenados. Si se realizan INREC y SUM a la vez, la suma se realiza sobre los registros ya formateados. En cambio en OUTREC se procesa después de realizar la suma. La sentencia INREC se procesa antes de SORT, SUM, OUTREC y después de INCLUDE y OMIT.

Por último la línea de la sentencia SORTWK*nn*, son utilizadas como áreas de trabajo temporales donde se guardan los registros a ordenar estas se deben definir como archivos temporales sin longitud de registro y se pueden codificar desde SORTWK01 hasta SORTWK16.

CAPÍTULO 9. REPOSICIONAMIENTO BATCH

En este capítulo hablaremos de reposicionamiento Batch, se explicará sobre la problemática y soluciones. Para ello ejemplificaremos como funciona el reposicionamiento por medio de ejemplos, nos basaremos en código COBOL para realizar los programas, en tablas DB2 que nos ayudan a controlar el reposicionamiento y con archivos secuenciales.

Cuando se trabaja con procesos Batch que manipulan grandes cantidades de datos y usen tablas DB2 surgen dos problemas:

- Las tablas DB2 usadas en el proceso se bloquean durante largos periodos de tiempo, impidiendo que otros procesos Batch accedan a ellas.
- Su recuperación, cuando ocurre un abend o error durante su ejecución, es costosa, ya que el DB2 debe restaurar todas las tablas utilizadas a su estado inicial.

Para evitar largos bloqueos de las tablas, los programas deben hacer confirmaciones de los cambios realizados cada cierto número de registros (COMMIT), lo que liberará los bloqueos existentes hasta ese momento.

Si llegara a haber un error el programa devuelve el control al sistema operativo, y toda la información que ha sido procesada se perdería, para aligerar su recuperación en caso de error, los programadores desarrollan una serie de utilidades que, incorporadas al proceso, permiten relanzarlo desde la última confirmación (COMMIT), posicionando la entrada en el siguiente registro al último confirmado y tratando los archivos de salida de forma que no se pierda la información confirmada.

El reposicionamiento Batch consiste en codificar los programas COBOL y los JCL's de tal modo que en caso de abend o error durante la ejecución, se pueda volver a arrancar el proceso desde el último dato tratado y confirmado.

Para ello , durante el proceso se harán confirmaciones periódicas de los datos modificados(COMMIT), de forma que cuando el proceso falla en su ejecución, el DB2 se encarga de restablecer las tablas a su estado en la última confirmación, deshaciendo todos los cambios (ROLL BACK).

Un JCL puede o no tener archivos de salida pero el ejemplo que estudiaremos será el caso cuando no se tienen archivos de salida, en este caso el DB2 deshace los cambios no confirmados en caso de error (ROLL BACK), por ello, para reposicionar los procesos que sólo actualizan datos en DB2, bastará con guardar la última clave tratada y confirmada, para continuar desde ella al rearrancar. Para este fin se crea una tabla, donde los programas deberán guardar la última clave tratada justo antes de la confirmación (COMMIT), junto con la información que consideren oportuna para reposicionarse en el rearranque.

UTILIDADES Y OBJETOS PARA EL REPOSICIONAMINTO BATCH

TABLAS DB2

Los programas guardarán toda la información necesaria para reposicionar sus procesos Batch en caso de rearranque en tablas DB2, para nuestro ejemplo llamaremos a la tabla TABESTADO.

La tablas TABESTADO se guardará el estado actual de cada proceso y toda la información necesaria para su reposicionamiento, cuando este fuese necesario.

Llamaremos clave de entrada a la información almacenada en TABESTADO tabla 1, que se utiliza para restablecer la situación del proceso al momento en el que se produjo el error, para su rearranque.

ATRIBUTO	CLAVE	FORMATO	LONG.	DESCRIPCION
PLANNAME	SI	CHAR	8	Nombre del plan DB2.
PROCES	SI	DEC	2.0	Proceso, un plan puede tener distintos procesos.
ESTADO	NO	CHAR	1	Estado actual de este proceso. Puede tomar los siguientes valores: P:Edo. Inicial, no necesita reposicionarse. C:Edo. Intermedio, necesita reposicionarse. F: Edo. Final, pendiente de concatenar los ficheros intermedios, en este estado no debe relanzarse el proceso.
NUMCOMM	NO	SMALLINT		Número de confirmaciones hechas antes del error o finalización.
PUNTEROS	NO	VCHAR	254	Información que sirve para reposicionar las tablas DB2 y los ficheros de entrada del proceso en el momento de su rearranque.

Tabla 1. Tabla TABESTADO.

INCLUDES Y COPYS

A continuación se describen las INCLUDES y COPYS que deben utilizarse para estandarizar el uso de las tablas y de la rutina UR0000 dentro de los programas:

DCLGEN de las Tablas DB2.

- D0204200 DCLGEN de la tabla TABESTADO generada por arquitectura.
- URDCLGEN DCLGEN de la tabla TABESTADO que se usará en los programas.

INCLUDES a Utilizar en los Programas con Reposicionamiento son:

- URCOPYS que contiene la definición de las variables RURCOMM y RUROPER de comunicación con la rutina UR0000.
- URWORK define las variables de trabajo para reposicionamiento.
- URCURSOR define el cursor a utilizar en el programa con reposicionamiento.

COPYS a Utilizar en los Programas con Reposicionamiento.

- URSQLCOD definición de la variable para controlar el SQLCODE de los cursores de reposicionamiento.
- URSWITCH definición de los SWITCHES utilizados por el programa con reposicionamiento.
- URMENSA define las variables de trabajo a utilizar para la cancelación.

Para saber más sobre copys de COBOL y consultar el contenido de todas las copys anteriores ver el anexo 3 copys reposicionamiento.

FUNCION CANCELAR

Se va a usar esta función para cancelar la ejecución de un programa por cualquier tipo de error. Se usarán los siguientes parámetros:

PARÁMETRO	FORM	DESCRIPCIÓN
TIPO_ERROR	X(01)	Tipo del error que provoca la cancelación del programa. Posibles valores: D:Error de acceso a DB2. R:Error en llamada a una rutina. F:Error en acceso a un archivo. Q:Resto de errores no catalogados.

Parámetros generales, para todos los tipos de error

COD_RETORNO	X(04)	Código de retorno del error.
RESPONSABLE	X(30)	Responsable de los errores del programa.
DESCRIPCION	X(80)	Breve descripción del error.
PROGRAMA	X(08)	Nombre del programa.
PARRAFO	X(20)	Nombre del párrafo donde se localiza error.

Parámetros para errores DB2 (Tipo “D”).

SQLCA	X(148)	Área de comunicación entre DB2 y programa.
TABLA_DB2	X(18)	Nombre de la tabla DB2.
DATOS_ACCESO	X(104)	Datos usados en el acceso que provoca el error.

Parámetros para errores en la llamada a rutinas(Tipo “R”)

RUTINA	X(08)	Nombre de la rutina cuya llamada provoca error.
PARAMETROS	X(114)	Datos usados en la llamada a la rutina.

Parámetros para errores en el acceso a archivos(Tipo “F”).

DDNAME	X(08)	Nombre de la DDNAME en el JCL.
FILE_STATUS	X(02)	Valor del FILE-STATUS en el acceso.
DATOS_REGISTRO	X(112)	Contenido del último registro leído.

Parámetros para el resto de errores sin catalogar(Tipo “Q”)

RESTO_DATOS	X(122)	Datos importantes para el error.

POSIBLES ESTADOS DE UN PROCESO

Los procesos Batch con reposicionamiento pasan por distintos estados, según el estado en el que se encuentre el proceso, el programa de reposicionamiento debe ejecutar unas acciones u otras.

Los posibles estados del reposicionamiento son:

- Estado inicial que se identifica en la TABESTADO con una “P”
- Estado Intermedio que se identifica en la TABESTADO con una “C”
- Estado final que se identifica en la TABESTADO con una “F”

Al comenzar la ejecución de un programa de reposicionamiento se comprobará el estado del proceso para ejecutar las acciones oportunas.

ESTADO INICIAL “P”

Un proceso esta en estado inicial cuando su última ejecución se completó correctamente, sin ningún error. Implica que el proceso se encuentra en condiciones de ser relanzado, los archivos resultantes del último proceso han sido borrados y se puede ejecutar de nuevo el plan.

Si el programa de reposicionamiento comprueba que el proceso se encuentra en estado inicial no reposicionará su entrada, puesto que debe comenzar el proceso desde el principio.

ESTADO INTERMEDIO “C”

Un proceso esta en estado intermedio durante su última ejecución acabó con un abend o un error después de haber hecho una o varias confirmaciones (COMMIT).

Si el programa de reposicionamiento comprueba que el proceso se encuentra en estado intermedio reposicionará su entrada, puesto que debe comenzar en el siguiente registro de entrada al último confirmado.

ESTADO FINAL “F”

Un proceso se encuentra en estado final cuando después de su ultima ejecución correcta aún no se han fusionado sus ficheros de salida.

Si el programa de reposicionamiento comprueba que el proceso se encuentra en estado final debe terminar sin procesar ningún dato. Puesto que los ficheros de salida definitivos aún no se han creado.

En este caso debería arrancarse el JCL desde el paso posterior al programa con reposicionamiento, que fusionará los juegos de ficheros creados por el programa en los ficheros de salida definitivos.

ESTRUCTURA DEL PROGRAMA COBOL

PARÁMETROS DE ENTRADA

Los programas Batch con reposicionamiento reciben como entrada los parámetros necesarios para su control. No son parámetros de aplicación, si necesitase otros parámetros para su proceso estos deberán pasarse al programa mediante ficheros de entrada.

Los parámetros para el control del reposicionamiento son:

- PLAN Nombre del plan a ejecutar con reposicionamiento. Debería corresponder con el propio.

- NUMREG Número de registros que se van a procesar antes de cada confirmación. Realmente no tiene porque indicar el número de registros de entrada: solo el tope del controlador del bucle de proceso de registros.

- NUMPROC Número del proceso asociado a ese Job. Normalmente siempre es '1' salvo el caso de programas que bifurcan y tratan tablas distintas según alguna entrada que lean. En este caso se pueden lanzar dos o más Jobs en paralelo con el mismo programa/plan pero distinto número de proceso para poder reposicionarse cada cual correctamente e independiente del otro Job.

- PREFIX Prefijo de los ficheros de salida generados.

ESQUEMA BÁSICO DEL PROGRAMA

I.1 Añadir las COPYS e INCLUDES necesarias para el reposicionamiento. Ver la siguiente Figura 9 donde se muestra el código del programa.

```
*****************************************************************
***       INCLUDE DE LA COPY de la DCJGEN DE "DATAESTADO"
*****************************************************************
     EXEC SQL INCLUDE URDCLGEN  END-EXEC.
*****************************************************************
***         INCLUDE DE LAS COPYS PARA REPOSICIONAMIENTO
*****************************************************************
     EXEC SQL INCLUDE URWORK  END-EXEC.
     EXEC SQL INCLUDE URCOPYS END-EXEC.
     EXEC SQL INCLUDE URCURSOR END-EXEC.
*****************************************************************
***                   COPY DE MENSAJES DE ERROR
*****************************************************************
     EXEC SQL INCLUDE URMENSA END-EXEC.
*****************************************************************
***       COPY QUE CONTIENE LOS  " SWITCHES " UTILIZADOS
*****************************************************************
     EXEC SQL INCLUDE URSWITCH END-EXEC.
*****************************************************************
***       CONTROL SQLCODE CURSORES REPOSICIONAMIENTO
*****************************************************************
     EXEC SQL INCLUDE URSQLCOD  END-EXEC.
*****************************************************************
```

Figura 9. Declaración de COPYS en código COBOL.

I.2 Obtención de los parámetros de reposicionamiento pasados al programa. Estos parámetros son pasados al programa por el JCL y recibidos en LINKAGE SECTION (PARMLIST). Ver la Figura 10, donde se muestra como se codificaría la obtención de los parámetros dentro del programa COBOL.

Para obtenerlos se invoca la rutina UR0000 con RUR-INT, ésta deja los parámetros desglosados en subniveles de la variable RURCOMM. Esta llamada inicial a la rutina UR0000 se tiene que hacer siempre.

La variable RUR-CALL ya contiene el nombre 'UR0000' y pertenece a una de las copys . La llamada a la rutina debe ser el primer paso en el cuerpo del programa y el formato puede ser uno de los siguientes:

```
EXEC-FUN UR_PARM PARMS(PARAMETROS) END-FUN
                        O
CALL RUR-CALL USING RUR-INIT RURCOMM PARAMETROS
```

Las variables obtenidas son:

- CA-PLANNAME: Nombre del plan.

- CA-PROCESO: Número de proceso.

- CA-NUMREG: Número de registros tratados en cada confirmación (COMMIT).

- CA-PREF: Prefijo de los archivos, dependerá del subsistema donde se este ejecutando (DES: Desarrollo, OR: Preproducción, EXP: Explotación).

```
*****************
LINKAGE SECTION.
*****************
****************************************************************
***   PARAMETROS QUE SE PASAN EN EL JCL:
***   (PARA PROGRAMAS QUE UTILIZANDO REPOSICIONAMIENTO
***   NO NECESITAN HACER LLAMADAS A LA RUTINA 'UR0000').
***   -  NOMBRE DEL PLAN.
***   -  EL PROCESO NORMALMENTE SERA 1, AUNQUE PUEDE SUB-
***      DIVIDIRSE EN VARIOS TOMANDO UN NUMERO CONSECUTIVO.
***   -  NUMERO DE REGISTROS PROCESADOS TRAS LOS CUALES SE
***      DECIDA HACER COMMIT.
****************************************************************
01  PARMLIST.
  05 PARM-LNG     PIC  XX.
  05 PARAMETRO    PIC X(200).
****************************************************************
```

Figura 10. Parámetros que pasa el JCL al programa.

I.3 Inicialización de la variable de reposicionamiento. Se debe inicializar la variable que contendrá la información necesaria para reposicionarse. Ver Figura 11, para observar como se codifica en COBOL.

```
*-------------------------------------------------------------*
*---          INICIALIZAR VARIABLES DE PROCESO             ---*
*-------------------------------------------------------------*
SET NO-HAY-ERROR-PROCESO TO TRUE
SET NO-FIN-PROCESO       TO TRUE
SET NO-RELANZAMIENTO     TO TRUE
SET NO-FIN-DATOS         TO TRUE
SET NO-ERROR             TO TRUE
```

```
MOVE ZEROS                TO UR-COMMIT
MOVE 'N'                  TO SW-FIN-TABLA
MOVE CA-PLANNAME          TO CPR-PROCESO
MOVE 'N'                  TO CPR-RELANZMTO
MOVE CA-PROCESO           TO CPR-NUM-PROCESO
MOVE ZEROS                TO CA-COMMIT
*-------------------------------------------------------------*
```

Figura 11. Inicialización de variables dentro cuerpo del programa COBOL.

I.4 Consulta del estado del proceso.

Antes de comenzar el tratamiento de los datos de entrada, se consultará en qué estado se encuentra el proceso, para saber si es necesario reposicionar o por el contrario se comienza el proceso desde el principio.

Para ello se accede a la tabla TABESTADO utilizando el cursor “REP” (ver Figura 12), proporcionando en el INCLUDE URCURSOR. Se abrirá el cursor para el plan y el número de proceso pasados en los parámetros de reposicionamiento y se recuperará una única fila.

```
*-------------------------------------------------------------*
*--- ABRE CURSOR “REP” Y EVALUA EL ERROR SQLCODE AL ABRIRLO  ---*
*-------------------------------------------------------------*
     EXEC SQL
         OPEN REP
     END-EXEC
        EVALUATE SQLCODE
            WHEN ZEROS
                 CONTINUE
            WHEN OTHER
                 MOVE 'D'                    TO  WK-TIPO-ERROR
```

```
                MOVE '9999'                 TO  WK-COD-RETORNO
                MOVE 'ERROR OPEN C-TABESTADO' TO  WK-DESCRIPCION
                MOVE SQLCA                  TO  WK-SQLCA


            MOVE 'A8001-ABRIR'          TO  WK-PARRAFO
                MOVE 'TABESTADO'              TO  WK-TABLA
                MOVE  SPACES                TO  WK-DATOS-ACCESO
                PERFORM 9-ABEND1
        END-EVALUE
*---------------------------------------------------------------*
*--- LEE EL CURSOR "REP"  RECUPERANDO LOS VALORES "RUR-"     ---*
*---------------------------------------------------------------*
        EXEC SQL
        FETCH REP INTO :RUR-PLANNAME,
                       :RUR-PROCES,
                       :RUR-ESTADO,
                       :RUR-NUMCOMM,
                       :RUR-PUNTEROS
        END-EXEC
*---------------------------------------------------------------*
```

Figura 12. Abre y lee el cursor de DATAESTADO para recuperar el estado del proceso.

Según el resultado de esta consulta se darán los siguientes pasos:

I.4.1 Si no se encuentra la fila en la tabla TABESTADO se dará de alta el proceso de forma inicial con los siguientes datos (ver Figura 13):
-Estado intermedio(C) (RUR_ESTADO).
-Número de validaciones a cero(RUR_NUMCOMM).
-Los punteros con los valores iniciales de la variable de reposicionamiento (RUR_PUNTEROS).

```
*-----------------------------------------------------------------*
*---     SI NO EXISTE EN TABESTADO SE CREA UNA ENTRADA         --
-*
*---     CON ESTADO "C" GUARDANDO PUNTEROS.                  ---*
*-----------------------------------------------------------------*
     PERFORM A8095-INSERT-TABESTADO
        THRU A8095-INSERT-TABESTADO-EXIT
     PERFORM A8000-COMMIT
        THRU A8000-COMMIT-EXIT
*-----------------------------------------------------------------*
Donde las funciones son las siguientes:
*-----------------------------------------------------------------*
 A8095-INSERT-TABESTADO.
*-----------------------------------------------------------------*
     EXEC SQL
     INSERT INTO TABESTADO
      (RUR_PLANNAME,
       RUR_PROCES,
       RUR_ESTADO,
       RUR_NUMCOMM,
       RUR_PUNTEROS)
     VALUES  (:CA-PLANNAME,
       :UR-PROCESO,
```

```
       'C',
       :UR-CEROS,
       :UR-VALOR-CURSOR)
     END-EXEC
     EVALUATE SQLCODE
     WHEN ZEROS
          CONTINUE
     WHEN OTHER
          MOVE 'D'            TO  WK-TIPO-ERROR
          MOVE '9999'         TO  WK-COD-RETORNO
          MOVE 'ERROR INSERT TABESTADO'
          TO  WK-DESCRIPCION
          MOVE SQLCA          TO  WK-SQLCA
          MOVE 'A8095-INSERT' TO  WK-PARRAFO
          MOVE 'TABESTADO'      TO  WK-TABLA
          MOVE  SPACES        TO  WK-DATOS-ACCESO
          PERFORM 9-ABEND1
     END-EVALUATE.
 A8095-INSERT-TABESTADO-EXIT.
 EXIT.
*-----------------------------------------------------------------*
*-----------------------------------------------------------------*
 A8000-COMMIT.
*-----------------------------------------------------------------*
     EXEC SQL COMMIT END-EXEC
     IF SQLCODE NOT = 0
       MOVE 'D'                    TO WK-TIPO-ERROR
       MOVE SQLCODE                TO W-SQLCODE
       STRING 'ERROR COMMIT PROCESO, SQLCODE = ' W-SQLCODE
       DELIMITED BY '  '   INTO WK-DESCRIPCION
       MOVE SQLCA                  TO WK-SQLCA
```

```
       MOVE 'A8000-COMMIT'          TO WK-PARRAFO
       MOVE 'TABESTADO'               TO WK-TABLA
       MOVE 'COMMIT'                TO WK-DATOS-ACCESO
       PERFORM 9-ABEND1
       END-IF.
A8000-COMMIT-EXIT.
EXIT.
*---------------------------------------------------------------*
```

Figura 13. Cuando no se encuentra en TABESTADO el proceso.

I.4.2 Si la consulta a la tabla TABESTADO indica que el proceso se encuentra en estado inicial (P), se modificará para inicializar el número de validaciones a cero (ver Figura 14).

Además se pondrá el switch de reposicionamiento a “N”, para que el proceso comience desde el principio. (SW-RELANZAMIENTO = NO-RELANZAMIENTO).

```
*---------------------------------------------------------------*
*---   "P" :  IMPLICA QUE LOS FICHEROS RESULTANTES DEL       ---*
*---          PROCESO HAN SIDO BORRADOS Y SE PUEDE EJECUTAR  ---*
*---          DE NUEVO EL PLAN.                              ---*
*---------------------------------------------------------------*

MOVE ZEROES TO UR-CEROS
MOVE ‘C’    TO RUR-ESTADO
*---  ACTUALIZAR A ESTADO "C" Y GUARDAR PUNTEROS.
PERFORM A8003-MOD-C-TABESTADO
   THRU A8003-MOD-C-TABESTADO-EXIT
PERFORM A8000-COMMIT
   THRU A8000-COMMIT-EXIT
MOVE ZEROES TO RUR-NUMCOMM
```

Donde las funciones son las siguientes:

```
*-------------------------------------------------------------*
 A8003-MOD-C-TABESTADO.
*-------------------------------------------------------------*


     EXEC SQL
     UPDATE TABESTADO
     SET    RUR_ESTADO   = :RUR-ESTADO,
            RUR_NUMCOMM  = :UR-CEROS,
            RUR_PUNTEROS = :UR-VALOR-CURSOR
     WHERE CURRENT OF REP
     END-EXEC
     EVALUATE SQLCODE
     WHEN ZEROS
          CONTINUE
     WHEN OTHER
          MOVE 'D'               TO  WK-TIPO-ERROR
          MOVE '9999'            TO  WK-COD-RETORNO
          MOVE 'ERROR UPDATE TABESTADO'
          TO  WK-DESCRIPCION
          MOVE SQLCA             TO  WK-SQLCA
          MOVE 'A8003-MOD-C'     TO  WK-PARRAFO
          MOVE 'TABESTADO'         TO  WK-TABLA
          MOVE  SPACES           TO  WK-DATOS-ACCESO
          PERFORM 9-ABEND1
          END-EVALUATE.
 A8003-MOD-C-TABESTADO-EXIT.
 EXIT.
*-------------------------------------------------------------*
```

```
*-------------------------------------------------------------*
 A8000-COMMIT.
*-------------------------------------------------------------*
     EXEC SQL COMMIT END-EXEC
     IF SQLCODE NOT = 0
       MOVE 'D'                    TO WK-TIPO-ERROR
       MOVE SQLCODE                TO W-SQLCODE
       STRING 'ERROR COMMIT PROCESO, SQLCODE = ' W-SQLCODE
       DELIMITED BY '  '   INTO WK-DESCRIPCION
       MOVE SQLCA                  TO WK-SQLCA
       MOVE 'A8000-COMMIT'         TO WK-PARRAFO
       MOVE 'TABESTADO'              TO WK-TABLA
       MOVE 'COMMIT'               TO WK-DATOS-ACCESO
       PERFORM 9-ABEND1
       END-IF.
 A8000-COMMIT-EXIT.
*-------------------------------------------------------------*
```

Figura 14. Cuando el estado del proceso es 'P'.

I.4.3 Si la consulta a la tabla TABESTADO indica que el proceso se encuentra en estado intermedio(C) será necesario reposicionarse antes de comenzar el tratamiento (ver Figura 15). Para ello:

-Se mueve la información del puntero (RUR_PUNTEROS) a la variable utilizada para el reposicionamiento.
-Se recupera el número de validaciones realizadas (RUR_NUMCOMM).

-Se pone el switch de reposicionamiento a “S”, para que el proceso se reposicione antes de comenzar el tratamiento.(SW-RELANZAMIENTO = SI-RELANZAMIENTO).

```
*-----------------------------------------------------------------*
*---   "C" :  IMPLICA QUE SE ESTAN REALIZANDO COMMITS        ---*
*-----------------------------------------------------------------*
     WHEN 'C'
     MOVE RUR-NUMCOMM         TO CA-COMMIT
     MOVE RUR-PUNTEROS-TEXT   TO UR-CLAVE-REPOSIC
     IF RUR-NUMCOMM > 0
       SET SI-RELANZAMIENTO   TO TRUE
     END-IF
     PERFORM A8004-CERRAR-TABESTADO
        THRU A8004-CERRAR-TABESTADO-EXIT

*-----------------------------------------------------------------*
 A8004-CERRAR-TABESTADO.
*-----------------------------------------------------------------*
     EXEC SQL CLOSE REP END-EXEC
     EVALUATE SQLCODE
         WHEN ZEROS
              CONTINUE
         WHEN OTHER
              MOVE 'D'            TO  WK-TIPO-ERROR
              MOVE '9999'         TO  WK-COD-RETORNO
              MOVE 'ERROR CLOSE TABESTADO' TO  WK-DESCRIPCION
              MOVE SQLCA          TO  WK-SQLCA
              MOVE 'A8004-CERRAR' TO  WK-PARRAFO
              MOVE 'TABESTADO'      TO  WK-TABLA
```

```
                MOVE  SPACES        TO  WK-DATOS-ACCESO
                PERFORM 9-ABEND1
     END-EVALUATE.
 A8004-CERRAR-TABESTADO-EXIT.
 EXIT.
*---------------------------------------------------------------*
```

Figura 15. Cuando el estado del proceso es 'C'.

I. 5. Confirmación inicial.

Antes de comenzar se debe hacer una primera confirmación (COMMIT) para liberar el bloqueo sobre la tabla TABESTAD. Ésta no contabiliza en el proceso del número de confirmaciones, de hecho, si no hay reposicionamiento, se ha inicializado a cero en la tabla TABESTADO.

I.6 Abrir los ficheros de entrada.

I.7 Inicializar las variables propias del proceso de la aplicación.

Debe tenerse en cuenta si se va a reposicionar o se va a comenzar el proceso desde el principio (SW-RELANZAMIENTO).

Si va a reposicionar desde el principio se iniciarán las variables normalmente.

Si se va a reposicionar, hay que recordar que se trata de reconstruir la situación en el punto donde se quedó, por lo tanto debemos restablecer el valor de las variables en ese momento, tanto de las tablas internas, como de los contadores y variables intermedias.

Es posible que en el momento del rearranque no dispongamos de esta información, en este caso necesitaremos utilizar el puntero de la tabla TABESTADO, si el volumen de la información nos lo permite. Si esto no fuera posible será necesario utilizar una tabla DB2 y trabajar con ella de la misma forma que con los cursores de reposicionamiento.

I.8 Reposicionamiento en los datos de entrada.

Sólo cuando el reposicionamiento es necesario, indicado por el switch SW-RELANZAMIENTO, se debe posicionar en los datos de entrada en el registro siguiente al último confirmado en la ejecución anterior del proceso.

Será necesario recuperar la posición inicial para el proceso tanto en las tablas DB2 como en los ficheros de entrada. El puntero de TABESTADO debe contener la información necesaria para ello.

El reposicionamiento en los archivos secuenciales de entrada se ejecutará un bucle que leerá todos los registros hasta situarse en el buscado.

El reposicionamiento en las tablas DB2 deben diseñarse los accesos con cursores en cascada para el reposicionamiento, de forma que en este caso se puedan abrir uno detrás de otro hasta situarse en la fila de la tabla señalada por el puntero de TABESTADO.

BUCLE EXTERIOR E INTERIOR

El programa contendrá, después de su inicio, un bucle externo y otro interno. En el bucle externo, que se repetirá hasta que no haya más datos de entrada (SW-FIN-DATOS) o hasta que ocurra un error, se incluirán las siguientes tareas, además de las propias de la aplicación.

Ext. 1. Iniciación del contador de registros procesados.

Ext. 2. Alocación de los ficheros de salida, si existen en el proceso.

Se hará una llamada a la rutina UR0000 con el parámetro RUR-NEW para alocar los ficheros de salida. Formato:

CALL RUR-CALL USING RUR-NEW RURCOMM DCLTABESTADO

La rutina UR0000 llamada de ésta manera construirá el nombre del fichero correspondiente para cada DDNAME y número de confirmación, alocando dinámicamente cada uno de ellos.

Si durante este proceso se produjese algún error aparecerá reflejado en el fichero de mensajes (RURMSG) y el JOB se cancelará. Así mismo por cada fichero alocado se enviará un mensaje informativo (UR0015).
No necesitan hacer ésta llamada los programas que no utilizan ficheros secuenciales de salida.

Ext. 3. Abrir los ficheros de salida, si existen en el proceso.

El **bucle interior** procesa todos los registros de entrada entre validaciones, hasta que el contador de registros procesados alcanza al valor de CA-NUMREG (parámetro introducido en el JCL), se acaban los datos de entrada o hay algún error.

Int. 1. Lectura del siguiente dato de entrada.

Int. 2. Proceso propio de la aplicación. Instrucciones para el tratamiento de los datos de entrada.

Int. 3. Actualización del contador de registros tratados. Suma uno al contador de registros tratados.

Ext. 4. Cerrar los archivos de salida, si los hay.

Ext. 5. Validar las modificaciones DB2. Se hace COMMIT.

Ext. 6. Si no hay archivos de salida, sumar 1 al número de validaciones (CA-COMMIT). Sólo si hay archivos de salida, se debe sumar 1 al número de validaciones, variable CA-COMMIT proporcionada por los parámetros del JCL.
En caso de que existan ficheros de salida no se deben actualizar el número de validaciones, por que lo hace la rutina UR0000 y si se vuelve a hacer en el programa provocará errores.

Ext. 7. Sólo si hay archivos de salida, actualizar el número de validaciones y el puntero en TABESTADO. Actualiza la fila de la tabla TABESTADO correspondiente a este plan/proceso sumando uno al número de validaciones y anotando en el puntero la última clave de entrada tratada correctamente. El plan/proceso con el que estamos trabajando son datos que se leyeron de los parámetros del JCL al principio del programa, en el paso I.2.

Final del Programa**, t**erminado el bucle exterior del programa, se darán los siguientes pasos:

F. 1. Cerrar los ficheros y cursores de entrada.

F. 2. Actualizar en la tabla TABESTADO el plan/proceso.

Si el proceso tiene ficheros de salida debe actualizarse con estado-final (F), que indica que el proceso está pendiente de la refundición de los ficheros de salida.

Si el proceso no tiene ficheros de salida debe actualizarse con estado-inicial (P), que indica que el proceso está lista para comenzar de nuevo desde el principio.

F. 3. Finalizar el proceso de reposicionamiento.

CODIFICACIÓN DEL JCL PARA PROCESOS CON REPOSICIONAMIENTO

El programa de aplicación con reposicionamiento deberá llamarse de la manera en que se muestra la Figura 16.

```
//PGM      EXEC PGM=IKJEFT01,DYNAMNBR=200,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSABOUT DD SYSOUT=*
//SYSDBOUT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(Nombre-DB2)
RUN PROGRAM(programa-inicial-aplicacion) PLAN(Plan-aplicacion) -

PARMS('PLAN=Plan,NUMREG=NumReg,NUMPROC=NumProceso,PREFIX=Prefijo')

END


..
```

Figura 16. Parámetros de un JCL.

Los parámetros, en esta llamada, deben completarse con la siguiente información:

PROGRAM: Programa de la aplicación, con reposicionamiento.

PLAN: Nombre del plan a ejecutar con reposicionamiento.

NUMREG: Número de registros que se van a procesar antes de cada confirmación.

NUMPROC: Número del proceso.

PREFIX: Prefijo de los ficheros dinámicos.

POOL: Con este parámetro se puede cambiar el POOL de discos gestionado por SMS. Este parámetro es opcional, por defecto tomará el valor “DISC”.

Finalmente en la Figura 17, tenemos como se codifica un JCL sin ficheros de entrada.

```
//HHBLIEA  JOB (201,'EXW-ISBAN'),CLASS=%%MU,MSGCLASS=A,
//         MSGLEVEL=(1,1),REGION=0M,SCHENV=MQR2
//* %%GLOBAL COMPUTER
//*
//JOBLIB DD DSN=MQSERIES.MQR2.BADAPTER,DISP=SHR
//*
//*****************************************************************
//**
//**  EJECUCION PROGRAMA HHBLIE
//**
//*****************************************************************
//*
//HHBLIE   EXEC PGM=IKJEFT1B,DYNAMNBR=20,REGION=30M,COND=(0,NE)
//SYSPRINT DD SYSOUT=8
//SYSOUT   DD SYSOUT=8
//RURMSG   DD SYSOUT=8
//SYSTSPRT DD SYSOUT=8
//SYSUDUMP DD SYSOUT=8
//SYSABOUT DD SYSOUT=D
//SYSDBOUT DD SYSOUT=D
//SYSIN    DD *
 DSN SYSTEM(DBGR)
```

```
 RUN  PROGRAM(HHBLIE) PLAN(HHBLIER) -
      PARMS('PLAN=HHBLIER,NUMREG=030,NUMPROC=1')
 END
//*
```

Figura 17. Código de un JCL sin ficheros de entrada.

CONCLUSIONES

El objetivo general de este escrito ha sido ofrecer a los egresados de la carrera de ingeniería en computación una referencia de JCL y procesos Batch ó lotes dada la falta de literatura que hay actualmente.

Como resultado se creó un escrito donde se muestra lo referente a los procesos Batch así como lo referente al Lenguaje de Control de Trabajos que va desde lo más básico hasta la creación de ejemplos mas complejos que se utilizan en el ámbito laboral.

En conclusión el escrito queda como un material bibliográfico de apoyo para introducir a los egresados de la carrera de Ingeniería en Computación al mundo laboral en el área de procesos Batch.

REFERENCIAS

IBM (2012) FUNDAMENTOS DE OS/VS CODIFICACIÓN JCL. IBM, Unites States of America.

ISBAN (2005) REPOSICIONAMIENTO BATCH. Versión 1.1, MADRID ESPAÑA.

Rafael Campillo Lorenzo (2007) MANUAL DE SUPERVIVENCIA CONCEPTOS BÁSICOS PARA COBOLEROS. INSA, Cáceres España.

Servicios Educacionales (2007) CURSO JCL y COBOL. México D.F.
Ann McIver McHoes, Ida M.Flynn (2011) SISTEMAS OPERATIVOS. CENGAE Learning. Mexico D.F.

Andrew S.Tanenbaum (1991) SISTEMAS OPERATIVOS MODERNOS. Prentice hall hispanoamericana S.A. México

William Stallings (2005) SISTEMAS OPERATIVOS ASPECTOS INTERNOS Y PRINCIPIOS DE DISEÑO. Pearson Prentice Hall. México D.F.

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/handheld/Connected/BOOKS/IGYLR205/CCONTENTS?DT=20000927030801

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/handheld/Connected/FINDBOOK?filter=JCL

http://ibmmainframes.com/manuals.php

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/handheld/Connected/BOOKS/IEA2B550/CCONTENTS?DT=20090527042445

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/handheld/Connected/BOOKS/IEA4B501/1.3.1?SHELF=&DT=19920925150907&CASE=
http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/handheld/Connected/BOOKS/ICE1CA30/1.1?SHELF=&DT=20080528171007&CASE=

ANEXO 1 . DEFINICION DE UN ARCHIVO VSAM

```
/*----------
//*         CREA ARCHIVO VSAM
//*---------
//PASO01 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=X
//SYSIN DD *
     DELETE ISTSAR.DP.R000.TVSORNOT
            CLUSTER
            PURGE
     DEFINE CLUSTER
            (NAME(ISTSAR.DP.R000.TVSORNOT)
            KEY(19  0).
            RECSZ(150 150)
            REUSE
            SHR(1 3)
            SPEED
            TRAKS (5  5)
            VOLUMES (DPRC93))
     DATA
           (NAME(ISTAR.DP.R000.TVSORNOT.DATA))
     INDEX
           (NAME(ISTAR.DP.R000.TVSORNOT.INDEX))
/*
```

- Todos los archivos con formato KSDS VSAM son identificados con un nombre único por el cual son referenciados, este nombre es conocido como ***Cluster name*** .
- Un cluster es la combinación de el DATA component y el INDEX component.
- El cluster proporciona una manera de tratar a los componentes INDEX y DATA como un componente único con nombre propio.
- Data Component es donde se almacenan los registros, y el Index Component es donde se almacena la llave principal definida.
- En el caso de los VSAM tipo KSDS el Cluster se compone de una parte de **DATA** y otra de **INDEX**, cada una con su propio nombre.
- Antes de que pueda usarse un archivo VSAM es necesario generar su Cluster y para esto primero hay que cerciorarse de que no exista, es por ello que se le antepone un **DELETE Cluster**.
- Se utiliza el programa IDCAMS para generar el Cluster VSAM
- La instrucción para crear el cluster es DEFINE CLUSTER.

- En el cluster VSAM se definen los siguientes parámetros:
  Parámetros necesarios:
  - ***NAME*** Nombre del cluster.
  - ***CYLINDERS*** (primario secundario)
  - ***RECORDS*** (primario secundario)
  - ***TRACKS*** (primario secundario) Espacio requerido para el VSAM
  - ***VOLUMES*** Es el volumen o volúmenes en donde será almacenado el **VSAM**

  Parámetros opcionales (los más importantes):

- ***KEY*** *(longitud desplazamiento)* Especifica la llave principal del archivo. La longitud de la llave puede ser entre 1 y 255 bytes. El desplazamiento es la posición del primer byte de la llave dentro del registro (La primer posición de un registro es CERO).
- ***CISZ (Tamaño)*** Con este parámetro se especifica el tamaño del Intervalo de Control (Unidad de información de un VSAM que se transfiere entre la memoria virtual y el disco). El CI Size para el INDEX y el DATA pueden ser especificados en forma separada. Para el INDEX los valores permitidos son 512, 1024, 2048 ó 4096 bytes. Para el DATA deben ser múltiplos de 512 o 2048 bytes, y pueden tener un tamaño entre 512 y 32768 bytes.
- ***RECSZ*** *(Longitud de* registro *promedio, longitud de* registro *máxima)* Especifica la longitud promedio y la máxima longitud de los registros almacenados en un VSAM. Cuando se almacenan registros de longitud uniforme, los valores en longitud promedio y longitud máxima deben ser el mismo
- ***SHR*** *(crossregion crosssystem)* Especifica cómo un archivo puede ser compartido entre usuarios. Los valores que pueden tomar los subparámetros *crossregion y crosssystem* son varios, pero usualmente se utilizan el 1 y el 3 respectivamente.
- ***Crossregion*** En un mismo sistema .
  El VSAM será compartido por cualquier número de usuarios con propósitos de lectura y sólo uno podrá usarlo en modo de lectura-escritura (atrapa el recurso)
  El VSAM será compartido por cualquier número de usuarios con propósitos de lectura y también puede ser accesado por un usuario para un proceso de escritura (no atrapa el recurso).
  El VSAM es compartido por cualquier número de usuarios en modo de lectura y escritura. No se garantiza la integridad del archivo si dos usuarios escriben al mismo tiempo, ya que el CLUSTER puede sufrir daños irreversibles.
  Con esta opción se ofrece protección cuando un archivo por varios usuarios en modo de lectura y escritura. Es necesario que los programas aplicativos utilicen las macros ENQ y DEQ del sistema para mantener la integridad de los datos.
- ***Crosssystem Entre sistemas diferentes.***

- 1 Reservada.
- 2 Reservada.
- 3 Permite ejecutar cualquier tipo de modo de acceso entre usuarios. Se delega al usuario la responsabilidad de controlar la integridad de la información.

Con esta opción se ofrece protección cuando un archivo es usado por varios usuarios en modo de lectura y escritura. Es necesario que los programas aplicativos utilicen las macros RESERVE y DEQ del sistema para mantener la integridad de los datos.

- ***SPEED*** Especifica que el espacio reservado para el VSAM no esta reformateado, es decir, puede contener información de un uso previo de ese espacio. Con esta opción, si la carga inicial del VSAM falla, los datos deben cargarse de nuevo desde el principio, ya que no se puede determinar cual fue el último registro cargado. La carga inicial se hace más rápida.
- ***REUSE/NOREUSE*** Establece si el archivo destino será abierto como reusable.
  REUSE: Define que el espacio definido para el Cluster puede ser reusado.
  NOREUSE: Define que el espacio definido para el Cluster no puede ser reusado.

CARGA DE UN ARCHIVO VSAM

```
//*-------
//*   CARGA DATOS AL ARCHIVO VSAM
//*-------
//PEUT0010  EXEC  PGM=IDCAMS,COND=(0,LT)
//SYSPRINT  DD  SYSOUT=&SALX
//SYSOUT    DD  SYSOUT=&SALX
//SYSUT1    DD  DSN=ISTSAR.DP.R000.INITVSAM,DISP=SHR
//          DD  DSN=ISTSAR.DP.P000.APOVSAM,DISP=SHR
//SYSUT2    DD  DSN=ISTSAR.DP.R000.TVSORNOT,DISP=OLD
//SYSIN     DD   *
     REPRO    INFILE(SYSUT1)  OUTFILE(SYSUT2) REUSE
/*
```

- ❑ El manejo de archivos VSAM en un JCL hace mediante el programa IDCAMS
- ❑ El enunciado DD etiquetado como SYUT1 se hace referencia al archivo fuente que puede ser secuencial o VSAM.
- ❑ En caso de que el archivo fuente sea secuencial, los registros a cargar en el VSAM deben de estar ordenados de acuerdo a la llave definida para el VSAM (Parámetro KEYS)

- ❑ En el enunciado DD etiquetado como SYUT2 se hace referencia al archivo destino, que puede ser secuencial o VSAM
- ❑ En el enunciado DD etiquetado como SYSIN se dan las instrucciones al IDCAMS para que realice la carga de información.
- ❑ La cantidad máxima de bytes que un archivo VSAM puede almacenar es de **2** (incluyendo la información de control).

Cuando un programa trata de abrir un archivo VSAM vacío, el sistema genera un error en OPEN 35. Para evitar este problema, es recomendable insertar un registro en blanco:

```
//SYSUT1    DD DSN=ISTSAR.DP.R000.INITVSAM,DISP=SHR
//                DSN=ISTSAR.DF.R000.APOVSAM,DISP=SHR
```

- La tarjeta SYSUT1 en el ejemplo de la carga del VSAM, esta compuesta por dos archivos: el INITVSAM y el APOVSAM.
- El archivo APOVSAM es un secuencial donde están todos los registros que se quieren cargar al VSAM.

El archivo INITVSAM debe tener la misma longitud de registro que el archivo APOVSAM y su contenido debe ser un registro en blanco.

El comando **REPRO** sirve para copiar tanto archivos VSAM como archivos no-VSAM.

Parámetros necesarios .

- ❑ ***INFILE*** Se especifica el nombre del enunciado DD que hace referencia el archivo fuente.
- ❑ ***OUTFILE*** Se especifica el nombre del enunciado DD que hace referencia el archivo destino

Parámetros opcionales (más usados):

- ***REPLACE/NOREPLACE*** Especifica si un registro del archivo fuente (INFILE) será reemplazado en el archivo destino (OUTFILE).
  • ***REPLACE*** Cada registro en el archivo destino cuya llave coincida con algún registro en el archivo fuente es reemplazado.
  • ***NOREPLACE*** Cada registro en el archivo destino cuya llave coincida con algún registro en el archivo fuente no es reemplazado. Por cada coincidencia se despliega un mensaje de 'duplicate record'**.**
- ***REUSE/NOREUSE*** Establece si el archivo destino será abierto como reusable.
  - ***REUSE*** Si el Cluster esta definido como REUSE: Si el VSAM esta vacío, se cargan los registros.

Si el VSAM contiene datos previos, se sobrescriben los registros.

- ***NOREUSE*** Si el VSAM esta vacío, se cargan los registros, de lo contrario, se adicionan los registros.
  Si el Cluster esta definido como NOREUSE:
  Si el VSAM esta vacío, se cargan los registros.
  Si el VSAM contiene datos previos, IDCAMS devuelve un código de retorno 232 al tratar de abrir el VSAM.

- ***SKIP (número)*** Especifica el número de registros que se deben ignorar antes de empezar a copiar registros. Por ejemplo, si se desea empezar a copiar desde el registro 100, se deberá codificar SKIP(99).
- ***COUNT (número)*** Especifica el número de registros que se quieren copiar.

BORRADO DE UN VSAM

```
//BORRADO     EXEC PGM=IDCAMS
//SYSPRINT    DD   SYSOUT=*
//SYSIN       DD   *
       DELETE HILDE.MASTER.EMPL
```

- El parámetro **DELETE** permite borrar archivos VSAM.
- Al borrar el **CLUSTER** se eliminan también el DATA y el INDEX que se encuentran asociados al archivo.
- Esta sintaxis también puede ser usada para el borrado de archivos secuenciales o particionados

COPIA DE UN VSAM

```
//DEFINIR    EXEC PGM=IDCAMS
//SYSPRINT   DD   SYSOUT=*
//file1      DD   DSN=...
//file2      DD   DSN=...
//SYSIN      DD   *
              REPRO INFILE(file1)   -
                    OUTFILE(file2)   -
               SKIP(nnn)        -
               COUNT(nnnn)
/*
```

- El parámetro **REPRO** permite:

- Copiar de VSAM a VSAM
- Copiar de secuencial a secuencial
- Convertir de secuencial a VSAM

- Los parámetros **INFILE** y **OUTFILE** especifican los archivos de entrada /salida.
- El parámetro **SKIP** es opcional y es usado cuando se quiere indicar el número de registros que deben "saltarse" antes de comenzar la copia.
- Otra opción es utilizar en el caso de estos archivos el parámetro **FROMKEY**, el cual indica la clave inicial a partir de la cual se quiere copiar.

```
FROMKEY ('LOPEZ      ')
```

- El parámetro **COUNT** indica el número de registros que quieren copiarse.
- En vez de este parámetro puede usarse **TOKEY**, en el cual se puede indicar la clave final.

```
TOKEY ('LOPEZ99999')
```

ANEXO 2. GDG

CREACIÓN DE UN GDG

```
//CREAGDG JOB (000),'DEFGDG',
//                  MSGCLASS=X,
//                  N0TIFY=USUARIO,
//                  MSGLEVEL= (1, 1),
//                  CLASS=Q,
//                  REGION=0M
//*------
//*   DEFINE UN GENERATION DATA GROUP
//*------
//PASO01     EXEC  PGM=IDCAMS
//SYSPRINT   DD SYSQUT *
//SYSIN DD    *
  DEFINE GDG                                -
          (NAME(ISTSAR.P002XJG.F980313)     -
           LIMIT(255)                       -
           NOEMPTY                          -
           SCRATCH)
/*
```

- Antes de poder crear generaciones de un GDG es necesario definir la base de mismo.
- Se utiliza el programa **IDCAMS** para generar la base del GDG.
- En la base del ***GDG*** se definen los siguientes parámetros**:**

  Parámetros necesarios:

  - *NAME* Nombre del GDG que se esta definiendo (máximo de 35 caracteres incluyendo puntos).
    Existen dos tipos de nombres:
    • *Unqualified name* Nombre de 1 a 8 caracteres alfanuméricos o nacionales ($, #, @). El primer carácter debe ser una letra o un carácter nacional.
    Ejem**:** *DSN=ARCHIVO1*
    • *Quallfied name* Son múltiples nombres tipo *unqualified name* unidos por puntos. Ejem**:** *DSN=TRE00.DP.A000.ARCH1***.** La longitud máxima de un GDG es de 35 caracteres incluyendo los puntos.
  - ***LIMIT*** Máximo número de generaciones, entre 1 y 255, permitidas para el GDG.
  - ***EMPTY*** Cuando el límite de generaciones es excedido, se descatalogan todas las generaciones.

- ***NOEMPTY*** Cuando el límite de generaciones es excedido se descataloga la generación más antigua.
- ***SCRATCH*** Elimina las generaciones que han sido descatalogadas.
- ***NOSCR4TCH*** No elimina las generaciones que han sido descatalogadas.

NOMBRES RELATIVOS Y NOMBRES ABSOLUTOS.
Las generaciones de un GDG tienen 2 tipos de nombres:

Nombre relativo
• Es la forma más común de hacer referencia a una generación desde el JCL
• Su formato es:
DSN=NOMBRE DE GDG(+ n) o DSN=NOMBRE DE GDG(-n)
(+ n) - Agrega una nueva generación DSN= NOMBRE DE GDG(+ 1)
(o) - Usa la generación más actual DSN= NOMBRE DE GDG(0)
(- n) - Usa una generación antigua DSN= NOMBRE DE GDG(-1)

Nombre absoluto
• El nombre absoluto de una generación es el nombre verdadero (nombre físico) del archivo en el medio de almacenamiento donde es alojado.

DSN= NOMBEDEGDG.G0001V00
• El número de generación inicia desde 1 y se incrementa de uno en uno por cada nueva generación.

RELACION ENTRE NOMBRES RELATIVOS Y ABSOLUTOS
Entre más grande sea el valor del número de generación, la versión almacenada es la más actual.
Ejemplo:

Nombre relativo	Nombre absoluto
NOMBRE.DE.GDG(+0)	NOMBRE.DE.GDG.C0003V00
NOMBRE.DE.GDG(-1)	NOMBRE.DE.GDG.G0002V00
NOMBRE.DE.GDC(-2)	NOMBRE.DE.GDG.C0001V00

Ejemplo para crear una nueva generación de GDG

```
//PNP12OR1 DD DSN=&PREARCH..DP.PNP12OR1(+1),
//                    DISP=(NEW,CATLG,DELETE),
//                    SPACE=(TRK, (10,5),RLSE),
//                    UNIT=&NBVOLTRB,
//                    DCB=(&PREARCH. .DP.MODEL,
//                    LRECL=(133,RECFM=FBA, BLKSIZE=0)
```

- La creación de una nueva generación es muy parecida a la creación de un archivo secuencial.

- El nombre relativo de la generación utiliza el (+1).
- Cuando se crean más de una generación en un mismo JOB, incrementar en 1 el número relativo por cada nueva versión (EJ. (+2), (+3), ... (+n))..
- Los nombres relativos son constantes dentro de la ejecución de un mismo JOB.
- Siempre utilizar DISP=OLD al procesar las generaciones de un GDG para evitar que otros JOBS procesen el mismo GDG al mismo tiempo.
- Dentro de !a DCB, el primer subparámetro es el nombre de un archivo secuencial creado previamente y cuyos valores en la DCB son tomados corno modelo. Las características de este archivo son**:**
    - SPACE=0
    - LRECL=80
    - RECFM=FB
    - BLKSIZE=0

El archivo modelo es único y puede ser usado por todos los GDG´s de la aplicación.

ANEXO 3. COPYS JCL

Una copy es un pedazo de código escrito en COBOL que puede sólo ser usado cuando es incluido como parte de un programa, contenien variables o funciones y se encuentran almacenadas en diversas librerías, además no pueden ser ejecutadas por ningún programa, ni por si mismos.

Puede ser usado en la Working Storage o Procedure Division (divisiones de un programa COBOL).

Su sintaxis:

COPY *nombre-copy*.

Sus ventajas:

- Reducen el tiempo de codificación de programación.
- Fácil mantenimiento ya que sólo existe una sola copia del código.

Los siguientes códigos son ejemplos de Copys que nos ayudarán en el ejemplo para el reposicionamiento:

URDCLGEN

Dclgen de TABESTADO: Tabla para guardar los punteros para el reposicionamiento.

```
************************************************************
***
* DCLGEN TABLA(TABESTADO)
****
************************************************************
EXEC SQL DECLARE DTABESTADO TABLE
( RUR_PLANNAME    CHAR(8) NOT NULL,
  RUR_PROCES      DECIMAL(2, 0) NOT NULL,
  RUR_ESTADO      CHAR(1) NOT NULL,
  RUR_NUMCOMM     SMALLINT NOT NULL,
  RUR_PUNTEROS    VARCHAR(254) NOT NULL
 ) END-EXEC.
************************************************************
* COBOL DECLARACION PARA TABLA TABESTADO
************************************************************
01 DCLTABESTADO.
```

```
           10 RUR-PLANNAME           PIC X(8).
           10 RUR-PROCES              PIC S99V USAGE COMP-3.
           10 RUR-ESTADO              PIC X(1).
           10 RUR-NUMCOMM             PIC S9(4) USAGE COMP.
           10 RUR-PUNTEROS.
              49 RUR-PUNTEROS-LEN     PIC S9(4) USAGE COMP.
              49 RUR-PUNTEROS-TEXT    PIC X(254).
**************************************************************
```

URCOPYS

Esta copy contiene las copys RURCOMM y RUROPER que se utilizan para comunicarse con la rutina UR0000 que gestiona el tratamiento de los ficheros de salida del proceso, dejando en ellas la información necesaria para el mismo.

En RURCOMM deja información acerca del número de commit por el que vamos, nombre del plan, el número de registros que se pasa en el JCL por los que hace COMMIT e información sobre los ficheros de salida como el nombre de la ddname, el blocaje, el tipo de dispositivo, número de registros por pista, etc. En RUROPER hay información sobre las llamadas a la rutina UR0000, cuando está en inicio, al final, etc.

```
**************************************************************
****** ****** COPY RUROPER***
**************************************************************
******

01 RUR-OPER.

  05 RUR-CALL           PIC X(8) VALUE 'UR0000'.
  05 RUR-INIT           PIC X(4) VALUE 'INIT'.
  05 RUR-NEW            PIC X(4) VALUE 'NEW '.
  05 RUR-READ.
      10 FILLER         PIC X(4) VALUE 'READ'.
      10 RUR-READ-DD    PIC X(8) VALUE SPACES.
  05 RUR-SHR            PIC X(4) VALUE 'SHR '.
  05 RUR-DEL            PIC X(4) VALUE 'DEL '.
  05 RUR-END            PIC X(4) VALUE 'END '.

**************************************************************
****** ****** COPY RURCOMM***
**************************************************************
******

01 RURCOMM.
  05 CA-OPER            PIC X(4).
  05 CA-PARAM.
```

```
    10 CA-PLANNAME     PIC X(8).
    10 CA-PROCESO      PIC 99.
    10 CA-COMMIT       PIC(4).
    10 CA-NUMREG       PIC S9(8) COMP.
    10 CA-PREF         PIC (4).
  05 CA-PARMOK         PIC X.
    88 PARMOK          VALUE 'S'.
  05 CA-MSG            PIC X(6).
  05 CA-PARM1          PIC X(20).
  05 CA-PARM2.
    10 CIRC            PIC 999.
    10 FILLER          PIC X.
    10 CIEC            PIC 9999.
    10 FILLER          PIC X.
    10 CIIC            PIC 999.
    10 FILLER          PIC X(2).
    10 CIRCX           PIC S9(4) COMP.
    10 CIECX           PIC S9(4) COMP.
    10 CIICX           PIC S9(4) COMP.
  05 CI-AMSG           PIC X(4).
  05 CA-OCCUR          PIC S9(4) COMP.
  05 CA-DDNAMES.
   06 CA-ELE-DDNAMES OCCURS 50 TIMES.
     10 CIDD           PIC X(8).
     10 CILDSN         PIC X.
     10 CIDSN          PIC X(30).
     10 CIPRI          PIC S9(4) COMP.
     10 CISEC          PIC S9(4) COMP.
     10 CIPOOL         PIC X(8).

     10 CIBLK          PIC S9(4) COMP.
     10 CILRECL        PIC S9(4) COMP.
     10 CIREGTRK       PIC S9(4) COMP.
     10 CIDISPOS       PIC X(4).

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

URWORK

Copy que contiene campos de trabajo estándar específicos para programas Batch que utilicen reposicionamiento.

```
*****************************************************************
***    URWORK:
***    COPY QUE CONTIENE LOS CAMPOS DE TRABAJO UTILIZADOS
***    EN PROGRAMAS BATCH QUE UTILIZAN REPOSICIONAMIENTO.
***
```

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

 01 UR-CONT-REG             PIC S9(8) COMP.
 01 UR-CEROS                PIC S9(4) COMP VALUE ZEROES.
 01 UR-PROCESO              PIC S9(2) COMP-3.
 01 UR-COMMIT               PIC S9(4) COMP.

*--- CAMPOS PARA GUARDAR LA ULTIMA CLAVE LEIDA DE LA TABLA.
*--- SOLO PARA PROGRAMAS QUE UTILIZAN TABLAS DB2 DE ENTADA.

 01 UR-VALOR-CURSOR.
    49 UR-VALOR-CURSOR-LEN  PIC S9(4) COMP.
    49 UR-VALOR-CURSOR-TEXT PIC X(254).

*--- CAMPOS PARA GUARDAR EL ULTIMO REGISTRO LEIDO DEL ARCHIVO
*--- PARA PROGRAMAS QUE USAN ARCHIVOS SECUENCIALES DE ENTRADA

 01 UR-VALOR-REGISTRO.
    49 UR-VALOR-REGISTRO-LEN  PIC S9(4)COMP.
    49 UR-VALOR-REGISTRO-TEXT PIC X(254).

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

URCURSOR

Copy que incluye el cursor para UPDATE de TABESTADO.

```
*****************************************************************
***  COPY QUE INCLUYE EL CURSOR PARA UPDATE DE TABESTADO
***  (TABLA EN LA QUE SE GUARDAN LOS PUNTEROS UTILIZADOS
***  PARA REALIZAR REPOSICIONAMIENTO EN PROGRAMAS BATCH).
***
***  PARA PROGRAMAS QUE LLAMAN A LA RUTINA "UR0000".
***
*****************************************************************
      EXEC SQL
      DECLARE REP CURSOR FOR
      SELECT RUR_PLANNAME,
             RUR_PROCES,
             RUR_ESTADO,
             RUR_NUMCOMM,
             RUR_PUNTEROS
      FROM TABESTADO
      WHERE RUR_PLANNAME = :CA-PLANNAME
        AND RUR_PROCES = :UR-PROCESO
        FOR PDATE OF RUR_ESTADO, RUR_NUMCOMM, RUR_PUNTEROS
```

```
      END-EXEC.
*****************************************************************
```

URSQLCOD

Campos de control para SQLCODE.

```
*****************************************************************
*** COPY PARA EL CONTROL DE SQLCODE UTILIZADO PARA
*** PROGRAMAS BATCH QUE UTILICEN REPOSICIONAMIENTO
*** Y UTILICEN TABLAS DB2 DE ENTRADA AL PROCESO.
***
*****************************************************************

      01 UR-SQL-CUR1    PIC S9(9) COMP.
      01 UR-SQL-CUR2    PIC S9(9) COMP.
      01 UR-SQL-CUR3    PIC S9(9) COMP.
      01 UR-SQL-CUR4    PIC S9(9) COMP.

*** SWITCH QUE INDICA EL CURSOR QUE ESTA ABIERTO EN ESE MOMENTO

      01 SW-UR-SCR      PIC X.
            88 UR-CSR1          VALUE'1'
            88 UR-CSR2          VALUE'2'
            88 UR-CSR3          VALUE'3'
            88 UR-CSR4          VALUE'4'
*****************************************************************
```

URSWITCH

Copy que contiene "switches" generales utilizados por los programas de aplicación, y un switch especifico de relanzamiento para el caso de los programas que utilizan reposicionamiento y leen un fichero secuencial.

```
*****************************************************************
*** COPY QUE CONTIENE LOS SWITCHES UTILIZADOS
*** POR PROGRAMAS BATCH QUE USAN REPOSICIONAMIENTO.
***
*****************************************************************


      01 FILLER              PIC X.
        88 HAY-ERROR-PROCESO     VALUE 'S'.
        88 NO-HAY-ERROR-PROCESO  VALUE 'N'.
```

```
      01 FILLER              PIC X.
        88 FIN-PROCESO           VALUE 'S'.
        88 NO-FIN-PROCESO        VALUE 'N'.

      01 SW-ERROR            PIC X.
        88 SI-ERROR              VALUE 'S'.
        88 NO-ERROR              VALUE 'N'.

      01 SW-FIN-DATOS        PIC X.
        88 FIN-DATOS             VALUE 'S'.
        88 NO-FIN-DATOS          VALUE 'N'.

*--- SOLO PARA PROGRAMAS QUE UTILIZEN FICHERO SECUENCIAL
*--- DE ENTRADA

      01 SW-RELANZAMIENTO    PIC X.
        88 SI-RELANZAMIENTO      VALUE 'S'.
        88 NO-RELANZAMIENTO      VALUE 'N'.

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

URMENSA

Esta copy contiene campos generales para mensajes de error.

```
****************************************************************
***COPY DE MENSAJES DE ERROR PARA PROGRAMAS BATCH
*** QUE USEN REPOSICIONAMIENTO.
****************************************************************

*** RUTINA DE CANCELACION UTILIZADA POR PROGRAMAS BATCH.
      01 XXCANCEL               PIC X.

*** CODIGOS DE MENSAJES :
      01 UR-CODICAN             PIC 9(4) VALUE 3700.

      01 UR-OPEN-ERROR          PIC 9(4) VALUE 3701.
      01 UR-FETCH-ERROR         PIC 9(4) VALUE 3702.
      01 UR-CLOSE-ERROR         PIC 9(4) VALUE 3703.

      01 UR-SELECT-ERROR        PIC 9(4) VALUE 3704.
      01 UR-UPDATE-ERROR        PIC 9(4) VALUE 3705.
      01 UR-INSERT-ERROR        PIC 9(4) VALUE 3706.

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