Tips for Mainframe Programmers

Compiled by
Rijo Joseph
Rijo.Joseph@in.ibm.com

CONTENTS

1. INTRODUCTION TO MAINFRAMES

COMPUTER SYSTEMS USED FOR BUSINESS PURPOSES

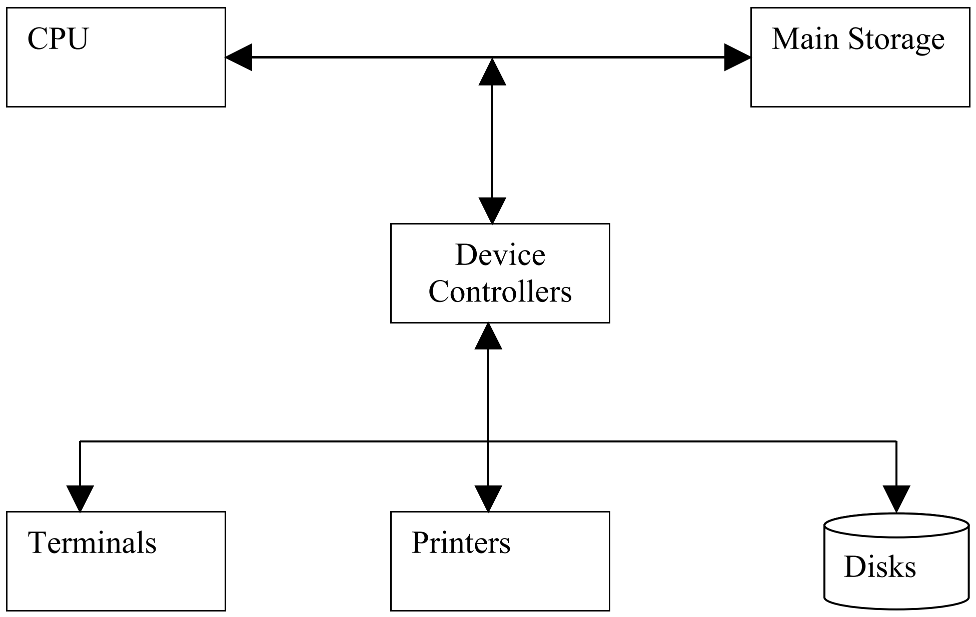
1) Micro Computers
2) Mini Computers
3) Mainframe Computers

These divisions are loosely based on the size of the computer systems

HARDWARE CONFIGURATIONS

Regardless of size, all computers consists of two basic types of components

- Processors
- I/O Devices



Although all computer systems consists of these basic components, the way those components are combined for a particular computer system varies depending on the system's requirements.

MICRO COMPUTERS

Small, single user systems that provide a simple processor and just a few input/output devices.
This is also known as PC.
This category is termed micro because the CPU was created on a single chip

Micro computers can be grouped into

- Workstations : Used for Extensive calculations
- Desktops : Business Processing
- Servers
- Laptops and Notebooks

MINI COMPUTERS

Also known as Mid-Range computers were first developed as special purpose mainframe computers. They were used, for instance to control machines in a manufacturing unit. Now they are widely used as general-purpose computers.
Faster than the microcomputer with access to more storage space and more input and output devices. The minicomputer is used when large groups need access to data simultaneously. The minicomputer can do this because the hardware is designed for plugging in more devices and the CPU and the support chips are designed for this kind of work

For example, large super markets around the world need to have their cash register send sales information to the same computer (so that the data is collected at one place)
Mini computers are used by medium sized business. They are also used in factories to control automated assembly lines, manufacturing, process control, etc,

Popular makers of minicomputer include DEC (Digital Equipment Corporation) – that built the popular VAX minicomputer. IBM also creates a very popular mini computer range with a branding of the AS/400. HP has a popular mini computer range branded HP9000

Unlike Micro computers, most mini computers provide more than one terminal so that several people can use the system at one time. A system like this is called a multi user system

IBM eServer i5 520

MAINFRAME CONFIGURATIONS

Mainframe computers can process several million program instructions per second. Large organizations rely on these room size systems to handle large programs with lots of data. Mainframes are mainly used by insurance companies, banks, airline/railway reservation systems, etc

Mainframes have even more access to storage space and to input/output devices. To work with these extra devices, mainframes also have more powerful processors. This power is useful and required by large corporations who have large amounts of data to process.

Consider the example of a large international bank that has millions of customer accounts require very powerful machines to process this data. These large banks would have a mainframe maintain their customer account records so that the customer can turn-up at any of the branches to withdraw/deposit money.

Characteristics

- Huge processing power
- Supports several users
- High security features

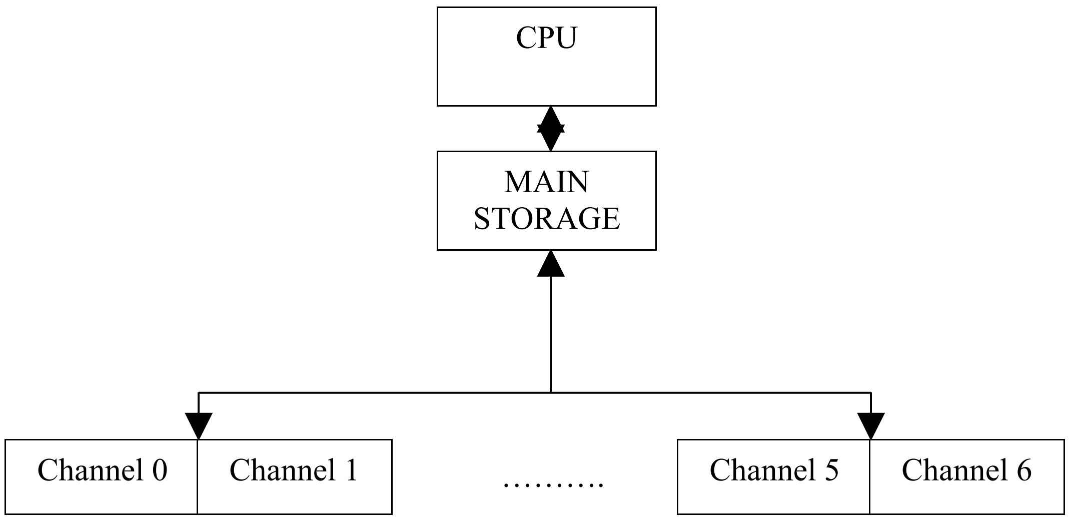
The most popular makers of Mainframe are IBM.

IBM eServer zSeries 890

THE SYSTEM /360-370 FAMILY

The most popular family of mainframe computers ever is the System/360-370 family, introduced by IBM more than 20 years ago.

The central components of mainframe computer system are the processors.



The figure shows the basic arrangement of the subcomponents that make up a typical System/370 type processor. As you can see, the processor consists of 3 main parts: CPU, Main Storage and Channels.

The CPU contains the circuitry needed to execute program instructions that manipulate data stored in main storage. Although the figure doesn't show it, most System/370 processors use a special purpose high speed memory buffer called a cache that operates between the CPU and Main Memory. This relatively small amount of storage operates at speeds even faster than the storage in main memory, so the overall speed of the processor is increased

Special circuitry constantly monitors access to main memory and keeps the most frequently accessed sections in the cache.

MULTIPROCESSING

To understand what a multiprocessor does, let us focus on the term single processor. A single processor or uniprocessor contains its own main storage and is controlled by a single OS. A Multiprocessor, on the other hand, can share system resources and work. For example, while one of the processors initiates an I/O operation, another processor can handle an interrupt and so on.
There are two types of multiprocessing systems

- Loosely coupled multiprocessing : where processors operate under separate operating systems, yet share access to data
- Tightly coupled multiprocessing : where processors operate under the same OS

CHANNELS

The purpose of a channel is to provide a path between the processor and an I/O device. In the above figure, there are 7 channels. As a result, there are 7 different paths along which data can pass between the processor and I/O device. Each channel can connect up to 8 devices called control units that connect to I/O devices. As a result, the processor in the above figure can connect up to 56 control units. Depending on the processor and the device, a control unit may be housed within the processor's cabinet, the I/O device's cabinet, or in its own cabinet.

The basic channel design of the System/370 requires that I/O devices to be connected to channels using heavy copper cables that can no longer than 400 feet in length. These channels use a parallel architecture which means that the cable transmits all of the bits that make up a byte simultaneously. To do that, the cable must have a separate wire for each bit – Sixteen in all (the channel sends two bytes simultaneously) - Plus additional wires for control signals. The result is that parallel channel cable is both heavy and expensive.

In 1990, IBM announced a new channel architecture, called ESCON (Enterprise System Connection), which is based on fiber optic rather than copper table. Fiber optic cable is not only 80 times lighter than copper table, but also 50 times less bulky. ESCON will allow many data centers to replace literally tons of unmanageable copper cable with neatly organized fiber optic cable. Besides its reduced size and weight, ESCON provides two other advantages. First it extends the 400-feet cable limit of standard channels to 26 miles. This lets installations locate disk devices on another floor or even in another building. Second, ESCON channels are nearly 4 times as fast as standard channels, transmitting data at 17 MB per second rather than 4.5 MB per second

In the new S/390 systems, a new channel architecture – S/390 Fiber Connectivity (FICON) – provides a new high performance I/O channel, optimized for efficiency at high speed.

MEMBERS OF THE IBM MAINFRAME FAMILY

PROCESSOR	CPUs	MAIN MEMORY	MAX CHANNELS
4381	1 OR 2	4 - 64 MB	18
3084	4	32 - 128 MB	48
3090	6	64 - 512 MB	128
ES/9000	4 - 8	2048 MB	256
S/390 G5/G6	4 - 12	1 - 32 GB	256
e-Server zSseries 990 (latest in z series)	32	16GB – 256GB	1024(ESCON), 240 (FICON)

- Most of these values are ranges or numbers because the processors come in a variety of models that offer various features

PR/SM

At one time, multiprocessor configurations were considered exotic, and the facilities for managing them were often inadequate. So many installations resorted to VM, an OS designed to emulate multiple computer systems. VM could be configured to take advantage of multiple processors, but it didn't provide all of the controls needed to take full advantage of modern multiprocessor configurations.

To address this problem, current IBM multiprocessors include a feature called Processor Resource/Systems Manager, or PR/SM. PR/SM allows an installation to divide a multi-CPU processor into several partitions, or LPARs, each of which can function as an independent system. PR/SM can be configured so that a partition is reserved as a backup for a primary partition. If the primary partition fails, the backup partition can automatically take over the work that was being processed at the time of the failure.

PR/SM also allows the I/O channels assigned to the various partitions to be reconfigured without disrupting the work

INPUT/OUTPUT DEVICES

I/O devices are the devices that connect to a processor to provide it with input, receive output or provide secondary storage.
The common types of I/O devices found on IBM mainframes are

- Unit Record Devices

  Card Devices and Printers. The name Unit record device implies that each record processed by the device is a physical unit. URDs usually have built in control units that attach directly to channels, so separate control units are not required

  - Card Devices
    - Card Readers
    - Card Punches
    - Reader/Punches
  - Printers
    - Impact : produce out put by striking an image of characters to be printed against a ribbon, which in turn transfers ink to paper
    - Non Impact Printers: use laser technology. IBM's 3800 printing subsystem can print at rates of up to remarkable 20,000 lines per minute

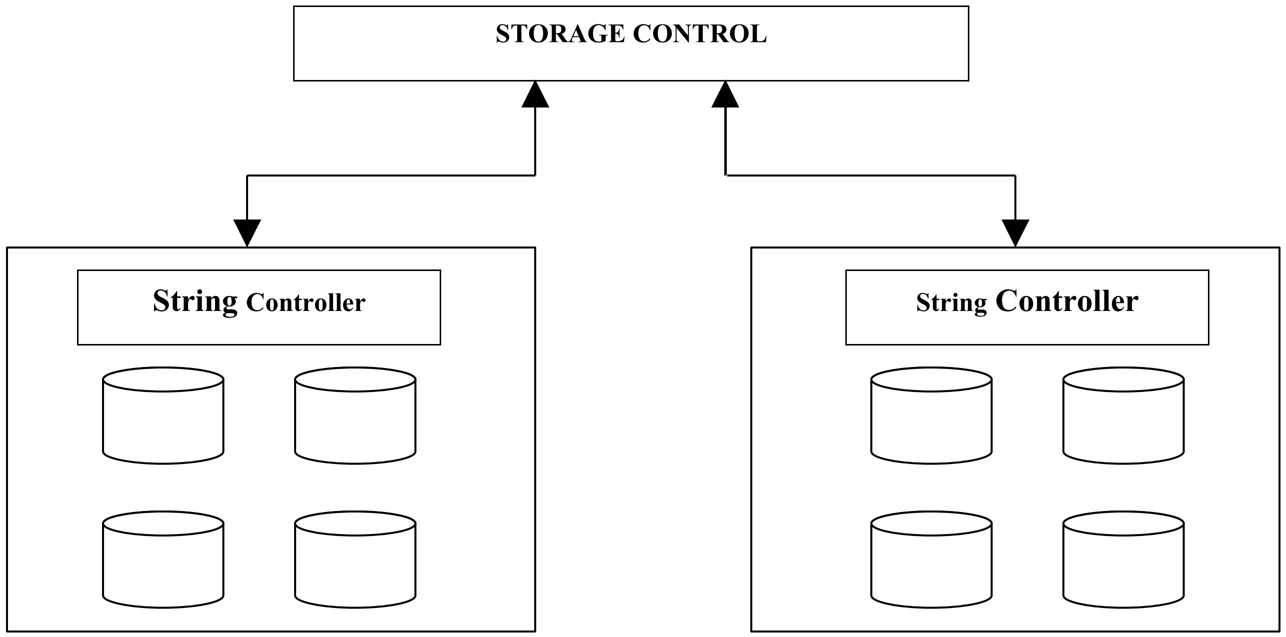
- Magnetic Tape Devices: How much data a reel or cartridge of tape can contain depends on the length of the tape and the density used to record the data. Density is a measurement of how many bytes are recorded in one inch of tape. Tape densities for standard reel tapes are usually 1600 or 6250 bpi. Data records are normally written to tape in groups called blocks. Empty spaces called gaps are required to separate blocks from one another. The larger the block, the less the amount of wasted space on a tape. However there is an extra cost involved when blocking is used: a buffer is required in main storage to contain the entire block. As a result, the larger the block, the more main storage that is required to contain it
- Direct Access Device: The official IBM term for a disk drive is DASD. Because it allows direct and rapid access to large quantities of data. Data is recorded on the usable surfaces of a disk pack in concentric circles called tracks. The number of tracks per surface varies with each device type. The component that reads and writes data on the tracks of a disk pack is called the actuator. The actuator has one read/write head for each recording surface. When the actuator moves, all of its heads move together so they are all positioned at the same track of each recording surface. As a result the disk drive can access data on all of those tracks without

moving the actuator. The tracks that are positioned under the heads of the actuator at one time make up a cylinder. As a result, there are as many tracks in a cylinder as there are usable surfaces on the pack, and there as many cylinders in a pack as there are tracks on a surface. So a pack that has 19 surfaces, each with 808 tracks has 808 cylinders, each with 19 tracks.

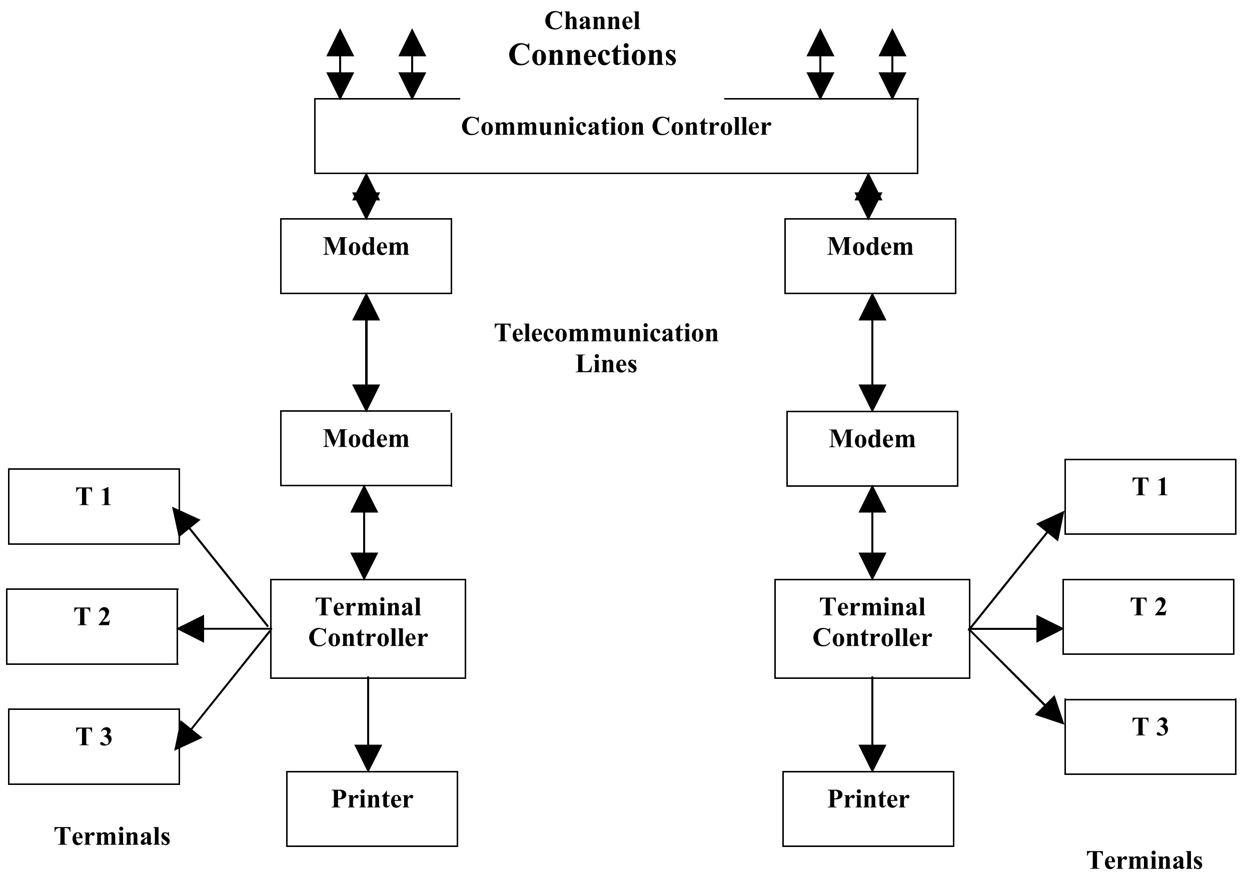
IBM manufactures two basic types of disk drives Count-Key-Data (CKD) devices and Fixed-Block-Architecture (FBA) devices. CKD devices store data in variable-length blocks. FBA devices store data in fixed-length blocks

Each type of DASD devices requires two kinds of control units to attach it to a processor channel. The first called a string controller, attaches a group of DASDs of the same type: the resulting group is called a string. The second kind of control unit, called a storage control, connects up to eight strings of DASD units to a channel

Special circuitry keeps track of what disk data is accessed most frequently and tries to keep this data in the cache storage. Then, when that data is referenced, it can be read directly from cache, the DASD unit doesn’t have to be accessed at all



- Data Communications Equipment : an installation creates a data communications network (telecommunications network) that lets users at local terminal (terminals at the computer site) and remote terminals (terminals that are not at the computer site) access a computer system

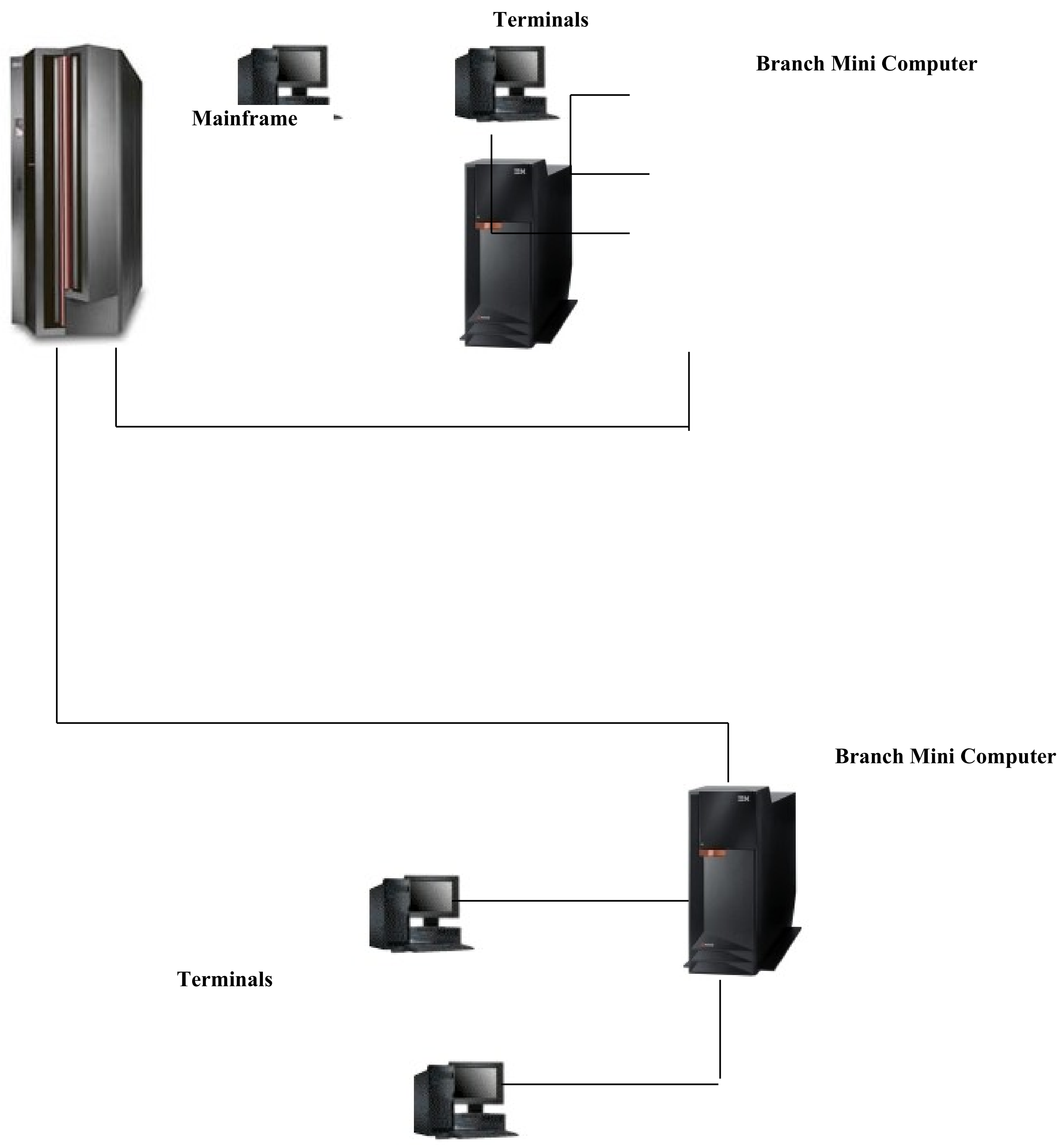


At the center of the network is the host system, a System/370 processor. The control unit that attaches to the host system's channels is called a communications controller; it manages the communications functions necessary to connect remote terminal system via modems and telecommunications lines.

Whether attached locally or remotely, the most commonly used terminal system on IBM mainframe is the 3270 Information Display system

APPLICATIONS

A distributed multi – branch banking system is shown. Bank executives use microcomputers to do 'what-if' analysis, make decisions etc. Each branch may have a mini computer to support a variety of needs for the individual branch. And the bank may have a centralized mainframe computer that supports all the banks branches providing for an even broader range of needs. In other words, the bank uses micro computers for applications at the individual level, minicomputers at the departmental or branch level, and mainframe computer at the corporate level. Mainframe at the branch headquarters performs all the consolidated accounting functions and is responsible for processing both transactions through the bank's network of automatic teller machines (ATM) and credit card transactions.



Multi Branch Banking System

CHARACTERISTIC FEATURES OF MAINFRAME OS

- VIRTUAL STORAGE

  In most computer systems, the processor's main storage is among the most valuable of the system's resources. As a result, modern mainframe computer operating systems provide sophisticated services to make the best use of the available main storage. Among the most important of these services include virtual storage

  Virtual storage is a technique that lets a processor simulate an amount of main storage that is larger than the actual amount of real storage. To do this, the computer uses disk storage as an extension of real storage

  The key to understanding virtual storage is realizing that at any given moment, only the current program instruction and the data it accesses need to be in real storage. Other data and instructions can be placed temporarily on disk storage, and recalled into main storage when needed. In other words, virtual storage Operating Systems transfer data and instructions between real storage and disk storage as they are needed.

- MULTIPROGRAMMING

  Multiprogramming means simply that the computer lets more than one program execute at the same time. Actually, that is misleading; at any given moment, only one program can have control of the CPU. A multiprogramming system appears to execute more than one program at the same time

  The key to understanding multiprogramming is to realize that some processing operations – like reading data from an input device – take much longer than others. As a result, most programs that run on mainframe computers are idle a large percentage of the time, waiting for I/O operations to complete. If programs were run one at a time on a mainframe computer, the CPU would spend most of its time waiting. Multiprogramming simply reclaims the CPU during these idle periods and lets another program execute

- SPOOLING

  A significant problem that must be overcome by multiprogramming systems is sharing access to input and output devices for the programs that execute together. One way to avoid this problem is to give one of the programs complete control of the printer. Unfortunately, that defeats the purpose of multiprogramming because the other program has to wait until the printer is available

To provide shared access to printer devices, spooling is used. Spooling manages printer output for applications by intercepting printer output and directing it to a disk device instead. Then, when the program finishes, the operating system collects its spooled print output and directs it to the printer. In a multiprogramming environment, the OS stores the spooled output separately on disk so it can print each program's output separately

Another benefit of spooling is that programs can execute faster. That is because; disk devices are much faster than printers. The OS component that actually prints the spooled output is multi programmed along with the application programs, so the printer is kept as busy as possible. But the application programs themselves are not slowed down by the relatively slow operation of the printer

- BATCH PROCESSING

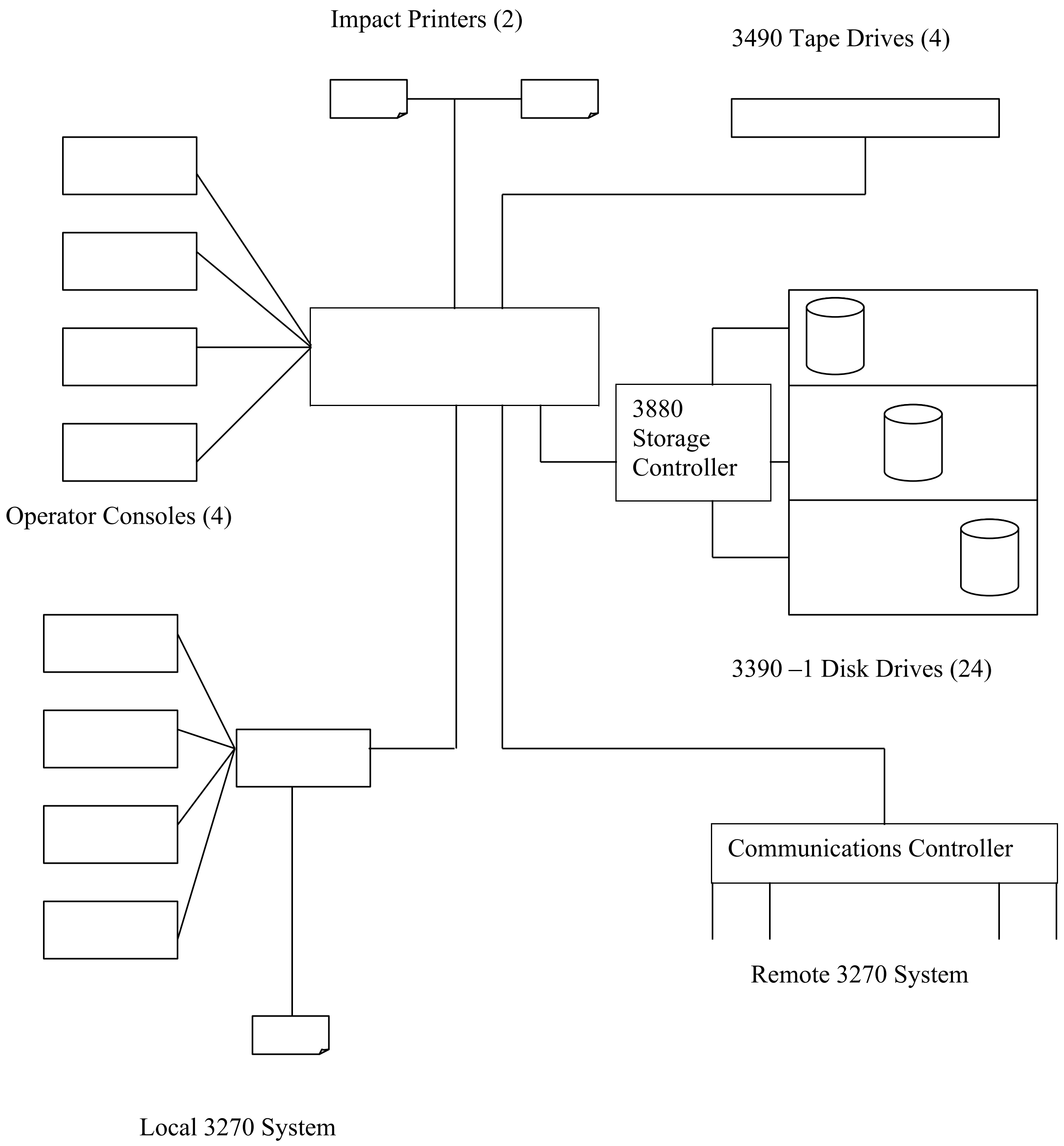
  When you use batch processing, your work is processed in units called jobs. A job may cause one or more programs to execute in sequence. For example, one job may invoke the programs necessary to update a file of employee records, print a report listing employee information, and produce payroll checks.

  One of the problems that arises when batch processing is used is managing how work flow through the system. In a typical mainframe computer system, many users compete to use the system's resources. To manage this, the Job Entry Subsystem, or JES processes each user's job in an orderly fashion

- TIME SHARING

  In a time sharing system, each user has access to the system through a terminal device. Instead of submitting jobs that are schedules for later execution, the user enters commands that are processed immediately. As a result, time sharing is sometimes called interactive processing, because it lets users interact directly with the computer. Sometimes, timesharing processing is also called foreground processing, while batch job processing is called background processing

A SMALL MAINFRAME COMPUTER CONFIGURATION



For Direct access storage, the configuration above uses three strings of 3390 model 1 DASDs, each containing 8 drives. Since each 3390 model 1 has a capacity of 946 MB, the total DASD capacity of this system is about 22 billion bytes. 3390 are fixed-media DASDs; their disk packs can't be removed. To provide a way to create backup copies of data on the 3390s, a string of four tape drives is used

The 4 operator consoles let system operators control the operation of the system. Some of the consoles might be dedicated to specific tasks, such as managing the tape drives or controlling the 2 high speed impact printers.

A local 3270 system, directly attached to the 4381 processor, provide 12 terminals and one printer. This terminals are used by the programming staffs, who are based in the same building that houses the computer.
k
The 3725 communications controller allows remote 3270 systems to access the system via telephone lines.

MVS CONCEPTS

MVS stands for multiple virtual storage. Under MVS, the concepts of virtual storage and multiprogramming are closely related.

Virtual Storage is a facility that simulates a large amount of main storage by treating DASD storage as an extension or real storage. In other words when virtual storage is used, the processor appears to have more storage than it actually does.

Multiprogramming is a facility that lets two or more programs use the processor at the same time. The key to understanding multiprogramming is realizing that most programs spend most of their time waiting for I/O operations to complete. So while one program waits for an I/O operation, the CPU can execute instructions for another program.

ADDRESS SPACES

Main storage consists of millions of individual storage locations, each of which can store one character, or byte, of information. To refer to a particular location, you use an address that indicates the storage location's offset from the beginning of the memory.

An Address space is simply the complete range of addresses – and as a result, the number of storage locations – that can be accessed by the computer. The maximum size of a computer's address space is limited by the number of digits that can be used to represent an address. To illustrate, suppose a computer records its addresses using six decimal digits, such a computer could access storage with addresses from 0 to 999,999

The original System/370 processors used 24 bit binary numbers to represent addresses. Since the largest number that can be represented in 24 bits is abut 16 million (16 M), an address space on a System/370 can't contain more than 16M bytes of storage. Because this 16 MB address space limitation severely restricted the capabilities of the System/370, IBM replaced it in the early 1980s with a new architecture known as 370-XA. 370-XA processors can operate in 370 mode using standard 24 bit addresses or in XA, or Extended Architecture, mode using 31-bit addresses, in XA mode, the largest address that can be represented – and therefore the largest address space that can be used – is about 2G.

In the late 1980s and early 1990s, IBM extended this architecture even further with the introduction first of ESA/370, then ESA/390. These designs utilize the same 31 – bit addresses that 370-XA uses.

MVS/370 was designed to operate on System/370 processors that utilize 24-bit addresses, and MVS/XA was designed to operate on 370-XA processors that use 31 – bit addresses. MVS/ESA also uses 31-bit addresses, but runs only on ESA/370 and ESA/390 processors.

To maintain compatibility with MVS/370, MVS/XA recognizes 24-bit processing. Whether it interprets an address as 24 or 31 bit depends upon the setting of the addressing mode bit in the current PSW at the time an instruction executes. Programs running in 24-bit addressing mode can access the first 16 MB of the virtual storage. MVS allows program to switch from one mode to another during execution in order to access data or call modules running in the other mode. Thus new programs can take advantage of 31 bit addressing and still be compatible with ones written for 24 bit addresses.

All MVS programs have an addressing mode (AMODE) attribute that indicates which addressing mode is to take effect when a module is given control. The AMODE attribute is assigned to MVS program module by the programmer as input to the assembler or linkage-editor. The default is 24 bit addressing mode.

MVS modules have a residence mode (RMODE) attribute that indicates whether they must be loaded below the 16 MB address line or can be loaded anywhere in the virtual storage. RMODE = 24 modules require residency below 16 MB. RMODE = ANY allows the OS to load a module anywhere in virtual storage

A program that must be directly addressable by 24 bit callers must reside below the 16 MB line. A program that doesn't have 24 bit callers, or whose 24 bit callers call it indirectly, can reside anywhere. The RMODE attribute is assigned as input to the assembler or the linkage-editor, or by default. The default is 24 bit residence mode.

PAGING

To allow the parts of a program in virtual storage to move between real and auxiliary storage, MVS breaks real storage, virtual storage and auxiliary storage into blocks. The terminology the system uses is as follows

- A block of real storage is a Frame
- A block of virtual storage is a Page
- A block of auxiliary storage is a Slot

A page, a frame and a slot are all of the same size; each consists of 4KB. An active virtual storage page resides in a real storage frame; a virtual storage page that becomes inactive resides in an auxiliary storage slot.

MVS divides virtual storage into 4K sections called pages. Data is transferred between real and DASD storage one page at a time. As a result, real storage is divided into 4K sections called page frames, each of which can hold one page of virtual storage. Similarly, the DASD area used for virtual storage, called a page data set, is divided into 4K page slots, each of which holds one page of virtual storage.

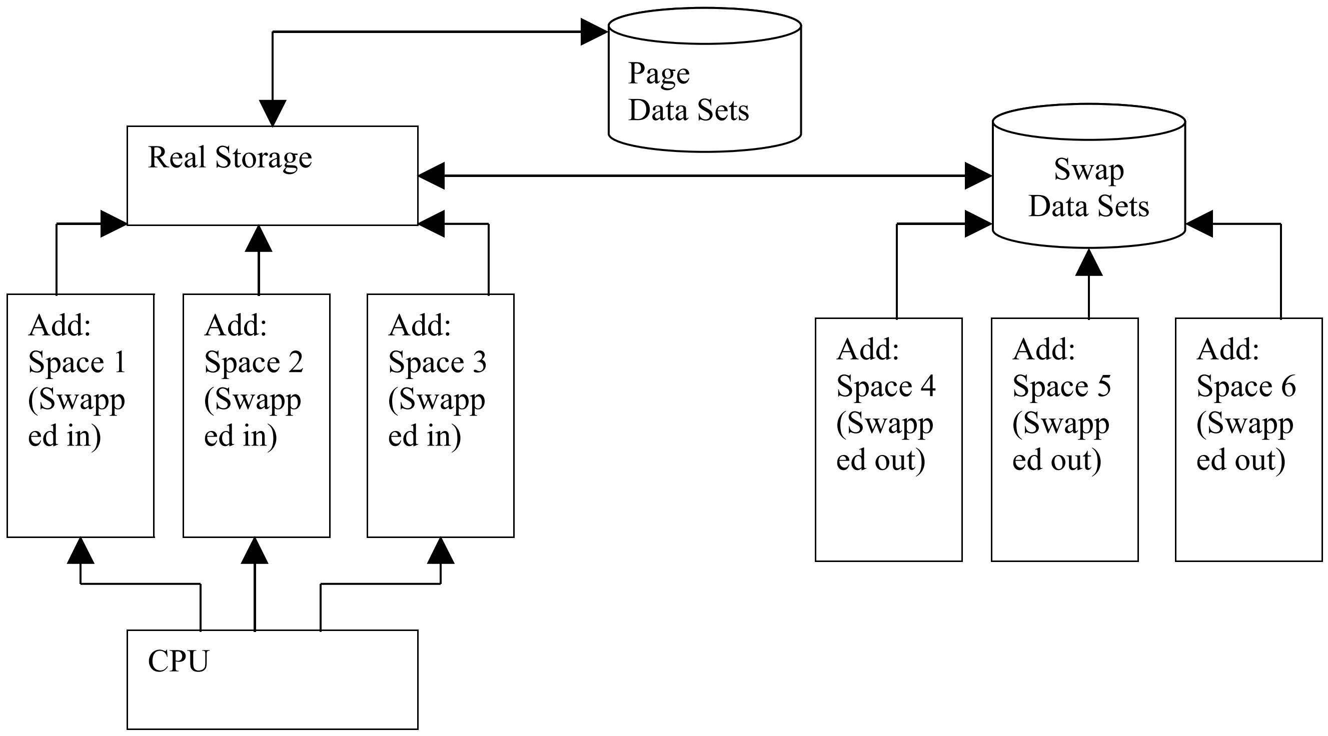
When a program refers to a storage location that is not in real storage, a page fault occurs. When that happens, MVS locates the page that contains the needed data on DASD and transfers it into real storage. That operation is called a page-in. In some cases, the new page can overlay data in a real storage page frame. In other cases, data in a page frame has to be moved to a page data set to make room for the new page. That is called a page-out. Either way, the process of bringing a new page into real storage is called paging.

EXPANDED STORAGE

Most new System/370 processors include a special type of memory known as expanded storage. Expanded storage improves the efficiency of virtual storage operations by acting as a large buffer between real storage and the page data sets. Simply put, when a virtual storage page must be paged out, the processor moves the page's contents to expanded storage. This transfer occurs at CPU speeds rather than at DASD speeds, so the operation is almost instantaneous. Pages are written to the actual page data set only when expanded storage becomes full. The amount of expanded storage on a processor varies depending on the processor model.

SWAPPING

In swapping, address spaces are moved from real storage to auxiliary storage and vice versa. This has the effect of moving an entire address space into or out of real storage. It is one of several methods MVS employs to balance the system workload as well as to ensure that an adequate supply of available real storage frames is maintained. Address spaces that are swapped in are active having pages in real storage frames and pages in auxiliary storage slots. Address spaces that are swapped out are inactive; the address space resides on auxiliary storage and can't execute until it is swapped in. Swapping is performed in response to the recommendations from the System Resource Manager (SRM).


Page Data Sets
Real Storage
Swap Data Sets
Add: Space 1 (Swapped in)
Add: Space 2 (Swapped in)
Add: Space 3 (Swapped in)
Add: Space 4 (Swapped out)
Add: Space 5 (Swapped out)
Add: Space 6 (Swapped out)
CPU

You can think of swapping as the same thing as paging, only at a higher level. Rather than move small 4K pieces of virtual storage in an out of real storage, swapping effectively moves entire address spaces in and out of virtual storage. Since paging occurs only for address spaces that are currently in virtual storage, paging doesn't occur for address spaces that are swapped out

THE MVS SYSTEM

- System Generation

  When an installation purchases the MVS operating system, IBM sends the basic components that make up the MVS on a series of tapes, called distribution libraries. System generation, only a part of the overall process of installing MVS from the distribution libraries, selects and assembles the various components an installation needs to create a working MVS system. To control system generation, often called sysgen, a systems programmer codes special macro instructions that specify how the MVS components from the distribution libraries should be put together.

Interestingly, an installation must already have a working MVS system before it can generate a new one. This is because an existing MVS system is required to execute the sysgen macro instructions. Fortunately, most installations perform sysgen to upgrade to a newer version of MVS or to make changes to their current version. So they can use their current version of MVS to execute the sysgen

For installations that do not already have an MVS system, the system installation process includes setting up a small, limited function MVS system that can execute the sysgen for the complete, full function MVS system

The macro instructions a systems programmer codes for a sysgen fall generally into two categories. The first category of macro instructions defines the system's hardware configuration... They are needed because MVS must know about every I/O device that is attached to the system. As a result, whenever a new I/O device is installed, the system must be generated again. MVS lets you do a smaller, less time consuming type of sysgen called an iogen to change the I/O device configuration.

The second category of macro instructions in a sysgen indicates which options of the OS should be included. Like which Job Entry Subsystem, what optional access methods and so on

The output from sysgen is a series of system libraries that contain, among other things, the executable code that makes up the OS

- System Initialization

  Once an MVS OS has been generated, it can be used to control the operation of the computer system. To begin a system initialization, the system operator uses the system console to start an Initial Program Load or IPL. That causes the computer system to clear its real storage and begin the process of loading MVS into storage from the system libraries.

2. COBOL

REDEFINES CLAUSE

Sometimes, it may be found that two or more storage areas defined in the DATA DIVISION are not in use simultaneously. In such cases only one storage area can serve the purpose of two or more areas if the area is redefined. The REDEFINES clause allows the same area to be referred to by more than one data name with different sizes and pictures.

For example,

```
01     SALES-RECORD.
       05    SALES-TYPE             PIC  X(01).
       05   SALES-BY-UNIT.
             10    QTY                    PIC  9(04).
             10    UNIT-PRICE         PIC  9(08)V99.
       05    TOTAL-SALES  REDEFINES
          SALES-BY-UNIT.
             10    AMOUNT             PIC   9(10)V99.
             10    FILLER             PIC   X(02).
```

This example describes a sales record which may either contain the total amount of sales (TOTAL-SALES) or the QTY & UNIT-PRICE
The purpose of such description may be to have two types of records and their types may be determined from the data item named SALES-TYPE
Depending on some predetermined values of SALES-TYPE the record will be interpreted in one of the two forms.
Note that SALES-BY-UNIT & TOTAL-SALES refer to the same storage space. They really represent two different mappings of this same storage area

Advantage:
Conservation of Storage space

Syntax:
Level-number   data-name-1 REDEFINES data-name-2

Rules
- The level-number of data-name-1 and data-name-2 must be identical.
- Except when the REDEFINES clause is used to 01 level, data-name-1 and data-name-2 must be of same size
- When the 01 level is used, the size of data-name-1 must not exceed that of data-name-2
- Multiple redefinitions are allowed. The entries giving the new descriptions must immediately follow the REDEFINES entry.

- In the case of multiple redefinitions, the data-name-2 must be the data name of the entry that originally defined the area.
- The REDEFINES clause must immediately follow the data-name-1
- Entries giving new descriptions can't have the value clauses (except in the case of condition-names(88))
- The REDEFINES clause must not be used for records (01 level) described in the FILE SECTION. The appearance of multiple 01 entry in the record description is implicitly assumed to be the redefinition of the first 01 level record.
- This clause must not be used for level-number 66 or 88

(COBOL 77 Standard)

According to COBOL - 85, a redefined data item can be equal or smaller in size than that of the data item it redefines.
Note that in COBOL - 74, the two must be equal.

For example

```
      05    DATA-A                 PIC   X(12)
      05    DATA-B  REDEFINES
            DATA-A.
            10    FIELD-A          PIC   9(03)V99.
            10    FIELD-B          PIC   9(02).
```

RENAMES CLAUSE

Sometimes a re-grouping of elementary data items in a record may be necessary so that they can belong to the original as well as to the new group.

```
01    PAY-REC.
      5     FIXED-PAY.
            10    BASIC-PAY                PIC   9(6)V99.
            10    DEARNESS-ALLOWANCE       PIC   9(6)V99.
      5     ADDITIONAL-PAY.
            10    HOUSE-RENT               PIC   9(4)V99.
            10    MNTHLY-INCENTIVE         PIC   9(3)V99.
      5     DEDUCTIONS.
            10    PF-DEDUCT                PIC   9(3)V99.
            10    IT-DEDUCT                PIC   9(4)V99.
            10    OTHER-DEDUCT             PIC   9(3)V99.
66    PAY-OTHER-THAN-BASIC    RENAMES
      DEARNESS-ALLOWANCE      THRU        MNTHLY-INCENTIVE
66    IT-AND-PF-DEDUCTIONS    RENAMES
      PF-DEDUCT               THRU        IT-DEDUCT.
```

In the above example, `PAY-OTHER-THAN-BASIC` will become a new group consisting of `DEARNESS-ALLOWANCE, HOUSE-RENT` and `MNTHLY-INCENTIVE`. Note that the new group overlaps on two original groups, namely, part of `FIXED-PAY` and the entire `ADDITIONAL-PAY`. Such overlapping is allowed provided the elementary items are all contiguous. In a similar way `IT-AND-PF-DEDUCTIONS` has two elementary items `PF-DEDUCT` and `IT-DEDUCT`. This new group is formed out of the original group `DEDUCTIONS`. Alternatively, the same thing can also be done in the original group description by placing like this

```
.
.
.
      5     DEDUCTIONS.
            10    IT-AND-PF-DEDUCTIONS.
                  15    PF-DEDUCT               PIC   9(3)V99.
                  15    IT-DEDUCT               PIC   9(4)V99.
            10    OTHER-DEDUCT            PIC   9(3)V99.
```

Syntax

```
66    data-name-1 RENAMES     data-name-2 THRU        data-name-3
```

Rules

- All RENAMES entries must be written only after the last record description entry
- The RENAMES clause must be used only with the special level number 66
- Data-name-2 and data-name-3 can be the names of elementary items or group items. They however can't be items of levels 01,66,77 or 88
- Neither data-name-2 nor data-name-3 can have an OCCURS clause in its description entry, nor can they be subordinate to an item that has an occurs clause in its data description entry
- Data-name-3, if mentioned, must follow data-name-2, in the record and must not be one of its sub fields.

USAGE CLAUSE

Normally, a computer can store data in more than one internal form. In COBOL, a programmer is allowed to specify the internal form of the data item so as to facilitate the use of the data item more efficiently. Broadly, there are only two general forms of internal representation – computational and display. Only numeric data items can be displayed as computational, and as the name suggests, an item specified as computational can take part in arithmetic operations more efficiently. On the other hand, any data item can be specified as display. This form is suitable for input/output and character manipulations. Whether a data item is computational or display, it can be specified with a usage clause in addition to the PIC clause

USAGE IS DISPLAY

This is the most common form of internal data. Each character of the data is represented in one byte and a data item is stored in a couple of contiguous bytes. The number of bytes required is equal to the size of the data item. One can specify the usage as DISPLAY. However, it is also the default.

COMP AND COMP-3 FIELDS

USAGE IS COMPUTATIONAL / USAGE IS COMP

`USAGE IS COMPUTATIONAL / USAGE IS COMP` stores data in the form in which the computer actually does its computation. Usually this form is binary. Thus defining `WORKING-STORAGE SECTION` entries in binary format is desirable when many repetitive arithmetic computations must be performed

`COBOL 85` permits the `USAGE IS BINARY` clause as well to specifically represent data in binary form

A few notes to remember: while allocating the space, for COMP fields, the compiler will allocate space in multiples of word, that is, it can be half word (2 bytes), full word or word (4 bytes), double word (8 bytes) like that

So if the PIC specified is `9(1) COMP, 9(2) COMP, 9(3) COMP OR 9(4) COMP,` the space allocated by the compiler will be half word (2 bytes). This allocation depends on the length of the maximum number that you can represent with this specification.

That is if you specify `9(4)`, the max number that you can represent is `9999`. This number can be easily represented by 2 bytes. With 2 bytes you can represent up to `65535`

If you specify `9(5) COMP, 9(6) COMP, 9(7) COMP OR 9(8) COMP`, the space allocated by the compiler will be one word (4 bytes). That is with `9(8)`, you can represent a maximum of `99999999`, this is possible with a word, and a word can represent up to `4294967295`

If you specify a `PIC` more than `9(8),` it will allocate a double word (8 bytes).

USAGE IS COMP-1

If the usage of a numeric data item is specified as COMP-1, it will be represented in one word (4 bytes) in the floating point form (single precision floating point form). Such representation is suitable for arithmetic operations. The number is actually represented in hexadecimal. The PIC clause can’t be specified for COMP-1 items

USAGE IS COMP-2

This usage is the same as COMP-1 except that the data is represented internally in two words (8 bytes). The advantage is that this increases the precision of the data (double precision) which means that more significant digits can be available for the item. The PIC clause can’t be specified for COMP-2 items

USAGE IS COMP-3

`COMP-3` enables the computer to store two digits in each storage position, except for the rightmost position, which holds the sign. The sign is stored separately as the rightmost half-a-byte regardless of whether S is specified in the PIC or not.

The hexadecimal number C or F denotes a positive sign and the hexadecimal number D denotes a negative sign.

Suppose if you move `1234567` into a field defined `9(7)`. In `DISPLAY` mode, which is default, this field will use 7 storage positions.
If you define the field with `PIC 9(7) COMP-3`, it will however use only four positions

12	34	56	7+

We can save a significant amount of storage by using the `USAGE-COMP-3`

USAGE IS COMPUTATIONAL / COMP / BINARY
This usage stores data in the form in which the computer actually does its computation. Usually this form is binary. Thus defining `WORKING-STORAGE SECTION` entries in binary format is desirable when many repetitive arithmetic computations must be performed. The item must be an integer (no assumed decimal point is allowed).

`COBOL 85` permits the `USAGE IS BINARY` clause as well to specifically represent data in binary form

A few notes to remember: while allocating the space, for COMP fields, the compiler will allocate space in multiples of word, that is, it can be half word (2 bytes), full word or word (4 bytes), double word (8 bytes) like that

So if the PIC specified is `9(1) COMP, 9(2) COMP, 9(3) COMP OR 9(4) COMP,` the space allocated by the compiler will be half word (2 bytes). This allocation depends on the length of the maximum number that you can represent with this specification.

That is if you specify `9(4)`, the max number that you can represent is `9999`. This number can be easily represented by 2 bytes. With 2 bytes you can represent up to `65535`

If you specify `9(5) COMP, 9(6) COMP, 9(7) COMP OR 9(8) COMP`, the space allocated by the compiler will be one word (4 bytes). That is with `9(8)`, you can represent a maximum of `99999999`, this is possible with a word, and a word can represent up to `4294967295`

If you specify a `PIC` more than `9(8),` it will allocate a double word (8 bytes).
Points to keep in mind while you declare an item as COMP

1) Don't use COMP if the data item is
   a) Properly DISPLAY
   b) Nonnumeric or edited
   c) A field associated with a unit record device or terminal (this means, you can't accept from user, an item that is declared as COMP)
2) Use COMP if the data item is a
   a) A numeric field in a record input from a tape or disk file whether it was already COMP
   b) Numeric, is used in calculations, and is a field in a record to be output to a tape or disk file from which it will later be input and used in further computation
   c) In WORKING-STORAGE, and is numeric and not edited and used in calculations

DIFFERENT FORMS OF EVALUATE STATEMENT

Evaluate is like a case statement and can be used to replace nested Ifs. The difference between EVALUATE and case is that no 'break' is required for EVALUATE i.e. control comes out of the EVALUATE as soon as one match is made.

Different forms of EVALUATE statements are

```
EVALUATE
WHEN A=B AND C=D
         imperative stmt
WHEN (D+X)/Y = 4
         imperative stmt
WHEN OTHER
         imperative stmt
END-EVALUATE
```

```
EVALUATE SQLCODE ALSO FILE-STATUS
     WHEN 100 ALSO '00'
        imperative stmt
     WHEN -305 ALSO '32'
        imperative stmt
     WHEN OTHER
        imperative stmt
     END-EVALUATE
```

```
EVALUATE SQLCODE ALSO A=B
WHEN 100 ALSO TRUE
   imperative stmt
WHEN -305 ALSO FALSE
         imperative stmt
END-EVALUATE
```

```
EVALUATE SQLCODE ALSO TRUE
WHEN 100 ALSO A=B
         imperative stmt
WHEN -305 ALSO (A/C=4)
         imperative stmt
END-EVALUATE
```

During the execution of an EVALUATE statement, the values denoted by the list of subjects (items in the EVALUATE statement) are compared with the values denoted by the list of objects in a WHEN phrase to establish a “match” between the two.

- The value of a subject is compared with the value/range of values of the object in the corresponding ordinal position
- In the case of a single valued (numeric/non numeric) object, the subject – object comparison is done in the usual way
- When a range of values is specified for the object, the subject – object comparison results in TRUE, if the value of the subject falls within the range
- In the case of conditional values, the subject – object comparison results in TRUE, if both evaluate to the same value (that is if both are TRUE or both are FALSE).
- If ANY is specified for an object, the subject-object comparison always results in TRUE
- The list of subjects is said to “match” with the list of object, if all the corresponding subject – object comparisons result in true
- Note that the values of the subjects need not be of the same class. For example one can be numeric and the other can be alpha numeric

After the execution of one of the when clauses, the control is automatically passed on to the next sentence after the EVALUATE statement. There is no need of any extra code.

SEARCH and SEARCH ALL

SEARCH is a serial search

SEARCH ALL is a binary search and the table must be sorted before using SEARCH ALL. (ASCENDING/DESCEDING KEY clause is to be used and the data must be loaded in this order)

Binary Search is search on a sorted array. Compare the item to be searched with the item at the center. If it matches, it stops the search else repeat the process with the left half or the right half depending on where the item lies.

Searching order in the SEARCH ALL can be either ASCENDING or DESCENDING. ASCENDING is default. If you want the search to be done on an array sorted in descending order, then while defining the array, you should give DESCENDING KEY clause. (You must load the table in the specified order).

Performance Considerations for Indexes VS Subscripts

- Using COMP to address a table is 30% slower than using indexes!
- Using COMP-3 to address a table is 300% slower than using indexes!!
- Using DISPLAY data item to address a table is 450% slower than using indexes!!!

Rule of the Thumb for SEARCH and SEARCH ALL

- For a table with less than 50 entries, go for SEARCH (Sequential Search)
- For a table with more than 50 entries go for SEARCH ALL (Binary Search)

Packed Decimal

Using an odd number of digits for PACKED DECIMAL (COMP-3) is 5% to 20% faster than using an even number of digits!

MEANING OF DEVICE NAME IN SELECT CLAUSE

```
SELECT   INFILE  ASSIGN  TO  UT-S-INFILE
OR
SELECT   INFILE  ASSIGN  TO  DA-S-INFILE
```

What they mean actually is

First Part in DDNAME: Device Class
UT stands for utility (Tape or sequential disk)
DA stands for direct access (Disk)

Second Part in DDNAMAE: Method of Organization
S – Sequential (Printer, Terminal, Disk or Tape)
I, R, D – Disk files to be accessed randomly

NEXT SENTENCE AND CONTINUE

There is a big difference between NEXT SENTENCE & CONTINUE. Please refer the test program given below

```
WORKING-STORAGE SECTION.
01  SAMPLE    PIC X(2) VALUE  'AB'.
01  SAMPLE2   PIC X(2) VALUE  'CD'.
PROCEDURE DIVISION.
PARA1.
    PERFORM PARA2.
    STOP RUN.
PARA2.
    IF SAMPLE = 'AB'
       NEXT SENTENCE
    ELSE
       DISPLAY 'SAMPLE IS NOT AB'
    END-IF
    IF SAMPLE2 = 'CD'
       DISPLAY 'SAMPLE 2 IS CD'
    ELSE
       NEXT SENTENCE
    END-IF
    .
```

In this case, we may assume that 'SAMPLE 2 IS CD' will get displayed but since SAMPLE is 'AB', NEXT SENTENCE will get executed and this **will transfer control to the statement following the period** so the program will move to STOP RUN. END-IF won't act here as a statement terminator. SYSOUT will be empty in this case.
It doesn't matter whether NEXT SENTENCE is nested inside an IF statement. SYSOUT will be empty in the following program also.

```
WORKING-STORAGE SECTION.
01  SAMPLE    PIC X(2) VALUE  'AB'.
01  SAMPLE2   PIC X(2) VALUE  'CD'.
01  SAMPLE3   PIC X(2) VALUE  'EF'.
PROCEDURE DIVISION.
PARA1.
```

```
    PERFORM PARA2.
    STOP RUN.
PARA2.
    IF SAMPLE = 'AB'
       IF SAMPLE3 = 'EF'
          NEXT SENTENCE
       END-IF
    ELSE
       DISPLAY 'SAMPLE IS NOT AB'
    END-IF
    IF SAMPLE2 = 'CD'
       DISPLAY 'SAMPLE 2 IS CD'
    ELSE
       NEXT SENTENCE
    END-IF
    .
```

`CONTINUE` will transfer control to the end of the statement that contains 'CONTINUE' (remember, end of the statement may not be a period). So, for the following program `'SAMPLE 2 IS CD'` will be printed in SYSOUT

```
WORKING-STORAGE SECTION.
01  SAMPLE    PIC X(2) VALUE  'AB'.
01  SAMPLE2   PIC X(2) VALUE  'CD'.
PROCEDURE DIVISION.
PARA1.
    PERFORM PARA2.
    STOP RUN.
PARA2.
    IF SAMPLE = 'AB'
       CONTINUE
    ELSE
       DISPLAY 'SAMPLE IS NOT AB'
    END-IF
    IF SAMPLE2 = 'CD'
       DISPLAY 'SAMPLE 2 IS CD'
    ELSE
       CONTINUE
    END-IF
    .
```

Why READ FILE and WRITE RECORD?

You `READ FILE` because you don't know in advance

1) Whether there actually a record to read or not

2) For variable or undefined length files, how long the next record will be if there is one.

You `WRITE RECORD` because you know in advance the answer to both of the above questions.

RETURN CODE OF INTERNAL SORT (IN A COBOL PROGRAM)

The return code or completion code is stored in a SORT-RETURN special register.

`IF SORT-RETURN = 0` (Successful completion of SORT/MERGE)
`IF SORT-RETURN = 16` (Unsuccessful completion of SORT/MERGE)

Static and Dynamic Subroutine CALLs

Keep in mind as you read this that some compilers let you set options that will override the calling mechanisms shown below. Therefore, even if your program is coded to call a program statically, the compiler can convert it to the dynamic form of CALL if you set (or don't set) the correct compiler options. Such compiler options are discussed at the end of this document.

Static CALLs

In COBOL, you normally call a subroutine like this:

```
CALL        'A'           USING        arguments
```

The static form of the CALL statement specifies the name of the subroutine as a literal; e.g., it is in quotes.

This is the static form of a subroutine call. The compiler generates object code for this which will cause the linker to copy the object module a.obj into your executable when it is linked.

So, if you modify "A" and recompile it, you must also relink all of the executables that call "A", because the each of the executables contains its own copy of "A".

Good Things about Static CALLs

- Fewer files are needed to distribute your application because your application can be built into a single EXE file, or perhaps an EXE for your main, and a couple of

DLLs for subordinate modules. This generally makes it simpler to distribute and/or upgrade your end users.

Note: A dynamic-link library (DLL) is a module that contains functions and data that can be used by another module (application or DLL).

- No risk of mixing/matching different versions of your called subroutines, because they are bundled into your main program.

Bad Things about Static CALLs

- You must relink all of the EXE and DLL files in your application that use a statically linked subroutine in order to use the newer version of the subroutine.
- If your application contains DLLs that call a subroutine statically, each DLL will have its own copy of the subroutine, including the storage defined in the subroutine. As a result, your application uses more storage than necessary.
- If your application has multiple DLLs that use the same statically named subroutine, each DLL has its own copy of that subroutine and its storage. Therefore, if you set a value in the subroutine in one of the DLLs, it's local to that DLL. The copy linked to the other DLLs will not know about this. This can be either a Good Thing or a Bad Thing, of course, but it's definitely a trap for the unwary.

Dynamic CALLs

In COBOL, the dynamic form of a subroutine call is coded like this:

```
   SUBROUTINE-A  PIC X(8)          VALUE 'A'.
.
.
.
CALL SUBROUTINE-A USING arguments
```

The dynamic form of the CALL statement specifies the name of the subroutine using a variable; the variable contains the name of the subroutine to be invoked.

The difference is that the name of the subroutine is found in the variable SUBROUTINE-A. **The compiled code will cause the operating system to load the subroutine when it is required instead of incorporating it into the executable**.

Note that you can also load a module dynamically by including it in a DLL and then linking it using the import library for that DLL!

Good Things about Dynamic CALLs

- You don't need to relink your application if you change something in your subroutine; only the subroutine DLL needs to be relinked.
- All executables that call this subroutine will share the same DLL; both the code and data. Since your application only loads one copy of a dynamically called subroutine, it uses less memory.
- Changes in values contained within the dynamically called subroutine are available to all the DLLs that use it, because they all share the same copy of the subroutine.

Bad Things about Dynamic CALLs

- Every dynamically called subroutine must be linked as a DLL (unless you use an import library to expose other entry points in a DLL). Therefore, if you application consists of hundreds of subroutines and they're all called dynamically, you will need to distribute hundreds of DLLs.
- It's possible to mix versions of your DLLs. This can be a problem both with distributing your application and with end-users installing updates improperly.
- If one of your DLLs is missing, you may not know about it until the user exercises some facility that tries to call that DLL. At that point, your application will terminate abnormally unless you handle this situation.
- If you CALL a DLL, CANCEL it, then CALL it again, you incur more I/O because the routine needs to be reloaded if you CANCEL it. This can slow down an application because it requires more disk activity
- If you mix and match static and dynamic calls to the same subroutine, your software might have several different versions in memory at once. Guess how much fun it will be trying to debug THAT mess?

Which is better, Static or Dynamic CALLs?
The answer is, it depends.

Static subroutines are nice, because your application can be built into a single EXE file, or perhaps an EXE for your main, and a couple of DLLs for subordinate modules.

Dynamic subroutines are nice because you can manage memory differently and you can update a portion of your application by shipping a newer DLL instead of the entire application.

You choose static/dynamic linking by choosing either the **DYNAM** or **NODYNAM** link edit option. (Even if you choose NODYNAM, a CALL identifier (as opposed to a CALL literal), will translate to a DYNAMIC call).
A statically called subroutine will not be in its initial state the next time it is called unless you explicitly use INITIAL or you do a CANCEL. A dynamically called routine will always be in its initial state.

EFFICIENT COBOL CODING TECHNIQUES

Data Types

Using the proper data types is an important factor in determining the performance characteristic of an application. Some of these are discussed below

BINARY (COMP OR COMP-4)
When using binary (COMP) data items, the use of the SYNCHRONIZED clause specifies that the binary data items will be properly aligned on half word, full word or double word boundaries. This may enhance the performance of certain operations on some machines. Additionally, using signed data items with eight or fewer digits produces the best code for binary items. The following shows the performance considerations (from most efficient to least efficient) for the number of digits of precision for signed binary data items (using PICTURE S9(n) COMP)

- n is from 1 to 8
  For n from 1 to 4, arithmetic is done half word instructions where possible
  For n from 5 to 8, arithmetic is done in full word instructions where possible

- n is from 10 to 17
  Arithmetic is done in double word format

- n is 9
  Full word values are converted to double word format and then double word arithmetic is used (this is slower than any of the above)

- n is 18
  Double word values are converted to a higher precision format and then arithmetic is done using this higher precision (this is the slowest of all four binary data items)

Note: using 9 digits is slower than using 10 digits.

Performance considerations for BINARY

Using 1 to 8 digits is the fastest
Using 10 to 17 digits is 20% to 30% slower than using 1 to 8 digits
Using 9 digits is 50% slower than using 1 to 8 digits
Using 18 digits is 1150% slower than using 1 to 8 digits

PACKED DECIMAL (COMP-3)

When using PACKED DECIMAL (COMP-3) data items in computations use 15 or fewer digits in the picture specification to avoid the use of library routines for multiplication and division. A call to the library routine is expensive compared to the calculation in-line. Additionally, using a signed data item with an odd number of digits produces more efficient code since this uses an integral multiple of bytes in storage for the data item (Please refer the Tips and Tricks doc on COMP-3).

Performance Considerations for PACKED DECIMAL

Using an odd number of digits is 6% faster than using the next lower even number of digits.

Using the fewest odd number of digits as possible may result in an additional 5% to 15% savings compared to using the next larger number of odd digits.

Comparing Data Types
When selecting your data types, it is important to understand the performance characteristics of them before you use them. Shown below are some performance considerations of doing several ADDs and SUBTRACTs on the various data types of the specified precision.

Performance Considerations for comparing Data Types

Packed decimal (COMP-3) compared to Binary (COMP or COMP-4)

- Using 1 to 5 digits: packed decimal is 150% to 210% slower than binary
- Using 6 to 8 digits: packed decimal is 220% to 260% slower than binary
- Using 9 to 17 digits: packed decimal is 100% to 240% slower than binary
- Using 18 digits: packed decimal is 65% faster than binary

DISPLAY compared to Packed Decimal

- Using 1 to 9 digits: DISPLAY is 60% to 95% slower than packed decimal
- Using 10 to 18 digits: DISPLAY is 100% to 180% slower than packed decimal

DISPLAY compared to Binary

- Using 1 to 5 digits: DISPLAY is 350% to 480% slower than binary
- Using 6 to 8 digits: DISPLAY is 500% to 570% slower than binary
- Using 9 to 17 digits: DISPLAY is 300% to 725% slower than binary
- Using 18 digits: DISPLAY is 10% faster than binary

Data Conversions

Conversion to a common format is necessary for certain types of numeric operations when mixed data types are involved in the computation. This results in additional processing time and storage for these conversions. In order to minimize this overhead, it is recommended that the following guidelines should be followed.

DISPLAY

Avoid using USAGE DISPLAY data items for computations (especially in areas that are heavily used for computation). When a USAGE DISPLAY data item is used, additional overhead is required to convert the data item to the proper type both before and after the computation. In some cases, the conversion is done by a call to a library routine, which can be expensive compared to using the proper data type that does not require any conversion

Performance considerations for DISPLAY

Using 1 to 5 digits is the fastest
Using 6 to 9 digits is 15% slower than using 1 to 5 digits
Using 10 to 13 digits is 50% slower than using 1 to 5 digits
Using 14 to 16 digits is 65% slower than using 1 to 5 digits
Using 17 to 18 digits is 120% slower than using 1 to 5 digits.

INDEXES Vs SUBSCRIPTS

Using indexes to address a table is more efficient than using subscripts since the index already contains the displacement from the start of the table and does not have to be calculated at run time. Subscripts on the other hand contain an occurrence number that must be converted to a displacement value at run time before it can be used. When using subscripts to address a table, use a binary signed data item with eight or fewer digits (for example, using PICTURE S9(8) COMP) for the data item). This will allow full word arithmetic to be used during the calculations. Additionally in some cases, using four or fewer digits for the data item may also offer some added reduction in CPU time since half word arithmetic can be used

Performance Considerations for Indexes Vs Subscripts (PIC S9(8))

Using binary data items (COMP) to address a table is 56% slower than using indexes

Using decimal data items (COMP-3) to address a table is 426% slower than using indexes

Using DISPLAY data items to address a table is 680% slower than using indexes

3. COMPILER OPTIONS

AWO Or NOAWO

The AWO compiler option causes the APPLY WRITE ONLY clause to be in effect for all physical sequential, variable length, blocked files, even if the APPLY WRITE ONLY clause is not specified in the program. With APPLY WRITE ONLY in effect, the file buffer is written to the output device when there is not enough space in the buffer for the next record. Without APPLY WRITE ONLY, the file buffer is written to the output device when there is not enough space in the buffer for the maximum size record. If the application has a large variation in the size of records to be written, using APPLY WRITE ONLY can result in a performance savings since this will generally result in fewer calls to Data Management Services to handle the I/Os.

Performance Consideration using AWO

One program using variable length files and AWO was 10% faster than NOAWO

OPTIMIZE (STD), OPTIMIZE (FULL) OR NOOPTIMIZE

To assist in the optimization of the code, you should use the OPTIMIZE compiler option. With the OPTIMIZE (STD) or OPTIMIZE (FULL) options in effect, you may receive optimizations that include

- Eliminating unnecessary branches
- Simplifying inefficient branches
- Simplifying the code for the out-of-line PERFORM statement , moving the performed paragraphs in-line, where possible
- Simplifying the code for a CALL to a contained (nested) program, moving the called statements in-line, where possible
- Eliminating duplicate computations
- Eliminating constant computations
- Aggregating moves of contiguous, equal sized items into a single move
- Deleting unreachable code

Additionally with the OPTIMIZE (FULL) option in effect, you may also receive these optimizations

- Deleting unreferenced data items and the associated code to initialize their VALUE clauses

NOOPTIMIZE is generally used while a program is being developed when frequent compiles are necessary. NOOPTIMIZE also makes it easier to debug a program since code is not moved. NOOPTIMIZE is required when using the TEST compiler option with a value other than TEST(NONE). OPTIMIZE requires more CPU times for compiles than NOOPTIMIZE, but generally produces more efficient run time code. For production runs, OPTIMIZE is recommended.

Performance Considerations using OPTIMIZE

- On the average, OPTIMIZE(STD) was 4% faster than NOOPTIMIZE, with a range of 17% faster to equivalent.
- On the average, OPTIMIZE(FULL) was equivalent to OPTIMIZE(STD)

SSRANGE Or NOSSRANGE

Using SSRANGE generates additional code, to verify that all subscripts, indexes and reference modification expressions are within the proper bounds. This inline code occurs at every reference to a subscripted or variable length data item, as well as every reference modification expression, and it can result in some degradation at run time. In general if you need to verify the subscripts only a few times in the application instead of at every reference, coding your own checks may be faster than using the SSRANGE option. For performance sensitive applications, NOSSRANGE is recommended.

Performance Considerations using SSRANGE with CHECK(ON)

On the average, SSRANGE was 4% slower than NOSSRANGE

TRUNC – BIN, STD OR OPT

When using the TRUNC(BIN) compiler option, all binary (COMP) sending fields are treated as half word, full word or double word values, depending on the PICTURE clause, and code is generated to truncate all binary receiving fields to the corresponding half word, full word or double word boundary (base 2 truncation). This can add significant amount of degradation since typically some data conversion must be done, which may require the use of some library routines. BIN is usually the slowest of the three sub options for TRUNC.

When using the TRUNC(STD) compiler option, the final or intermediate result of an arithmetic expression, or the sending field in the MOVE statement, is truncated to the number of digits in the PICTURE clause of the binary (COMP) receiving field (base 10 truncation). This too can add significant amount of degradation since typically the number is divided by some power of 10 (depending on the number of digits in the PICTURE clause) and the remainder is used. A divide instruction is one of the most expensive instructions.

With TRUNC(OPT), the compiler assumes that the data conforms to the PICTURE and USAGE specifications and manipulates the result based on the size of the field in storage (half word, full word or double word)

Performance Considerations for TRUNC

On the average, TRUNC(OPT) was 27% faster than TRUNC(BIN)
On the average, TRUNC(STD) was 26% faster than TRUNC(BIN)
On the average, TRUNC(OPT) was 5% faster than TRUNC(STD)

NUMPROC – NOPFD, MIG OR PFD

Using the NUMPROC(PFD) compiler option generates significantly more efficient code for numeric comparisons. It also avoids the generation of extra code that NUMPROC(NOPFD) or NUMPROC(MIG) generates for most references to COMP-3 and DISPLAY numeric data items to ensure a correct sign is being used. With NUMPROC(NOPFD), sign fix-up processing is done for all references to these numeric data items. With NUMPROC(MIG), sign fix-up processing is done only for receiving fields (and not for sending fields) of arithmetic and move statements

With NUMPROC(PFD), the compiler assumes that the data has the correct sign and bypasses the sign fix-up processing

Using NUMPROC(NOPFD) or NUMPROC(MIG) may also inhibit some other types of optimization. However not all external data files contain the proper sign for COMP-3 or DISPLAY signed numeric data, and hence using NUMPROC(PFD) may not be applicable for all application programs. For performance sensitive applications, NUMPROC(PFD) is recommended when possible.

Performance Considerations using NUMPROC

On the average, NUMPROC(PFD) was 1% faster than NUMPROC(NOPFD)
On the average, NUMPROC(PFD) was 1% faster than NUMPROC(MIG)
On the average, NUMPROC(MIG) was equivalent to NUMPROC(NOPFD)

DYNAM Or NODYNAM

The DYNAM compiler option specifies that all subprograms invoked through the CALL literal statement will be loaded dynamically at run time. This allows you to share common subprograms among several different applications, allowing for easier maintenance of these subprograms since the application will not have to be re-link-edited if the subprogram is changed. DYNAM also allows you to control the use of virtual storage by giving you the ability to use a CANCEL statement to free the virtual storage used by a subprogram when the subprogram is no longer needed. However when using the DYNAM option, you pay a performance penalty since the call must go through a library routine, whereas with the

NODYNAM option, the call goes directly to the subprogram. Hence the path length is longer with DYNAM than with NODYNAM.

Performance Considerations Using DYNAM with CALL (Measuring CALL overhead only)

On the average, for a CALL intensive application, the overhead associated with the CALL using DYNAM ranged from 80% slower to 350% slower than NODYNAM

RENT Or NORENT

Using the RENT compiler option causes the compiler to generate some additional code to ensure that the program is reentrant. Reentrant programs can be placed in shared storage like Link Pack Area (LPA) or the Extended Link Pack Area (ELPA) on MVS. Also the RENT option will allow the program to run above the 16 MB line on MVS. Producing the reentrant code may increase the execution time path length slightly.

Performance considerations using RENT
On the average RENT was equivalent to NORENT with a range of equivalent to 1% slower.

Run Time Options that Affect Run Time Performance

Selecting the proper run time options is another factor that affects the performance of a COBOL application.

ALL31

The ALL31 option allows LE to take advantage of knowing that there are no AMODE(24) routines in the application. It specifies that the entire application will run in AMODE(31). This can help to improve the performance for an all AMODE(31) application because Linkage Editor can minimize the amount of mode switching across calls to common run time library routines.

Performance Considerations using ALL31 (Measuring CALL overhead only)

On the average, ALL31(ON) was 1% faster than ALL31(OFF)

CHECK

The CHECK option activates the additional code generated by the SSRANGE compiler option, which requires more CPU time resources for the verification of the subscripts, indexes and reference modification expressions. Using the CHECK(OFF) run time option deactivates this code but still requires some additional CPU time resources at every use of a subscript, index or reference modification expression to determine that this check is not desired during the particular run of the program. This options has an effect only on a program that has been compiled with the SSRANGE compiler option

Performance Considerations using CHECK
On the average, CHECK(ON) with SSRANGE was 2% slower than CHECK(OFF) with SSRANGE

4. JCL

PDSE (PARTITIONED DATA SET EXTENDED)

A PDSE is a Partitioned Data Set Extended. It consists of a directory and zero or more members, just like a PDS. It can be created with JCL, TSO/E and ISPF just like a PDS and can be processed with the same access methods.

The directory can expand automatically as needed, up to the addressing limit of 524,286 members. It also has an index which provides a fast search for member names.

Space from deleted or moved members is automatically reused for new members, so you don't have to compress a PDSE to remove wasted space. Each member of a PDSE can have up to 15,728,639 records

Allocating a PDSE

Batch Allocation

The DSNTYPE JCL Keyword specifies that a data set should be a PDSE or PDS. If DSNTYPE=LIBRARY is specified, the data set is a PDSE. If DSNTYPE=PDS is specified, the data set is a PDS

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEFBR14
//***********************************************************
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DSN=XXXX.JCLTEST.RIJO.PDSE,
//         DISP=(NEW,CATLG,DELETE),
//         DSNTYPE=LIBRARY,
//         UNIT=TESTDA,
//         SPACE=(CYL,(10,10,10)),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//
```

When directory blocks are not specified, DSORG=PO must be included to identify the data set as partitioned

```
//SYSIN    DD DSN=XXXX.JCLTEST.RIJO.PDSE2,
//         DISP=(NEW,CATLG,DELETE),
//         DSNTYPE=LIBRARY,
//         DSORG=PO,
//         UNIT=TESTDA,
//         SPACE=(CYL,(10,10)),
```

TSO/E Allocation

When allocating a PDSE with the TSO/E ALLOCATE command, you can specify DSNTYPE to identify the data set as PDSE or PDS. If a data set exists with the DCB attributes that you need, you can use the LIKE parameter to copy the attributes to the new PDSE

```
ALLOC  DS('XXXX.JCLTEST.RIJO.PDSE2')
LIKE('XXXX.JCLTEST.RIJO.PDSE')
```

The example below shows an example of allocating a PDSE using the LIKE parameter to copy the attributes of an existing PDS. Adding the DSNTYPE(LIBRARY) parameter to the allocation makes the data set PDSE.

```
ALLOC DS('XXXX.JCLTEST.RIJO.PDSE3')
LIKE(XXXX.JCLTEST.RIJO.PDSE')
DSNTYPE(LIBRARY)
```

ISPF Allocation

We can use the 3.2 option

```
Data Set Name  . . . : XXXX.JCLTEST.RIJO.PDSE6

Management class . . . MCTSL            (Blank for default management
class)
Storage class  . . . . SCSMS            (Blank for default storage class)
 Volume serial . . . . TSLB90           (Blank for system default volume)
**
 Device type . . . . .                   (Generic unit or device address)
**
Data class . . . . . . DCPOTSL          (Blank for default data class)
 Space units . . . . . TRACK            (BLKS, TRKS, CYLS, KB, MB, BYTES
                                         or RECORDS)
 Average record unit                    (M, K, or U)
 Primary quantity  . . 1                (In above units)
 Secondary quantity    1                (In above units)
 Directory blocks  . . 0                (Zero for sequential data set)  *
 Record format . . . . FB
 Record length . . . . 80
 Block size  . . . . . 800
 Data set name type  : LIBRARY          (LIBRARY, HFS, PDS, or blank)   *
                                        (YY/MM/DD, YYYY/MM/DD
```

The Data set name type field specifies whether the new data set is to be a PDSE or PDS. Specify the Data set name type field to be LIBRARY to define a PDSE. With a PDSE allocation, the Directory blocks field is optional.

Creating PDSE Members

Member creation for PDSE is functionally same as for a PDS. There is no functional difference between the two formats

Deleting PDSE Members

A member of a PDSE can be deleted through the ISPF library panel or access method services. With a JCL disposition of delete (DISP=(OLD,DELETE)), the entire data set is deleted. This is the same as for a PDS.

IEBCOPY TO CONVERT A PDS TO PDSE

Here is an example of using IEBCOPY to convert a PDS to a PDSE. In this example, IEBCOPY retains the original PDS, creates a PDSE and copies the members of the PDS into the PDSE. The space is explicitly defined for the PDSE. You may use the LIKE parameter to get the DCB information form a PDS or PDSE

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBCOPY
//**********************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.JCLS,DISP=SHR
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.PDSE.JCLS,
//         DISP=(NEW,CATLG,DELETE),
//         DSNTYPE=LIBRARY,
//         DSORG=PO,
//         UNIT=TESTDA,
//         SPACE=(CYL,(10,10)),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN    DD *
       COPY INDD=SYSUT1,OUTDD=SYSUT2
/*
```

When to use a PDSE

A data set should be considered for PDSE allocation or conversion when one or more of these characteristics apply

- Frequent compresses are needed for the data set
- Out of space abends occur often

- A large directory leads to lengthy directory searches (PDS directory searches are sequential while PDSE searches are indexed)
- The directory size is unknown at allocation time
- The directory size will grow considerably
- The data set is shared for output
- Members will be shared and reused many times
- Protection is needed to prevent users from overwriting the directory or changing the DCB attributes for members

IEBCOPY

With IEBCOPY, you are able to perform any of the following

- Make a copy of a PDS or PDSE
- Merge partitioned data sets (except when unloading)
- Create a sequential form of PDS or PDSE for backup or transport
- Reload one or more members from a PDSU (Partitioned Data Set Unloaded – An IEBCOPY unload data set. A sequential file that can be restored by IEBCOPY to create a PDS) into a PDS or PDSE
- Select specific members of a PDS or PDSE to be copied, loaded or unloaded
- Rename selected members of a PDS or PDSE when copied
- Exclude members from a data set to be copied, unloaded or loaded
- Compress a PDS in place
- Convert a PDS to a PDSE or a PDSE to a PDS…

Examples

To copy an entire PDS to another PDS

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//              CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBCOPY
//*****************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.JCLS,DISP=SHR
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.JCLS.BKUP,
//         DISP=(NEW,CATLG,DELETE),
//         UNIT=TESTDA,
//         SPACE=(CYL,(10,10,40)),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN    DD *
  COPY INDD=SYSUT1,OUTDD=SYSUT2
/*
```

To convert a PDS to a PDSE

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBCOPY
//**********************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.JCLS,DISP=SHR
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.PDSE.JCLS,
//         DISP=(NEW,CATLG,DELETE),
//         DSNTYPE=LIBRARY,
//         DSORG=PO,
//         UNIT=TESTDA,
//         SPACE=(CYL,(10,10)),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN    DD *
       COPY INDD=SYSUT1,OUTDD=SYSUT2
/*
```

You will get error if you try the following JCL

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBCOPY
//************************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.WORK.SRTINPUT,DISP=SHR,
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.WORK.SRTINPUT.BKUP,
//         DISP=(NEW,CATLG,DELETE),
//         UNIT=TESTDA,
//         SPACE=(CYL,(1,1)),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN    DD *
  COPY INDD=SYSUT1,OUTDD=SYSUT2
/*
```

Where, `XXXX.JCLTEST.RIJO.WORK.SRTINPUT` and `XXXX.JCLTEST.RIJO.WORK.SRTINPUT.BKUP` are sequential data sets.

The errors that are displayed are

```
E1        8   DSS20026E    DATA SET
                           'XXXX.JCLTEST.RIJO.WORK.SRTINPUT.BKUP'
                           MUST BE A PDS.
E2        8   DSS11046E    DATA SET
                           'XXXX.JCLTEST.RIJO.WORK.SRTINPUT' IS
                           NOT AN UNLOADED DATA SET.
```

To copy an entire PDS to a sequential tape data set (Unloading)

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=J,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBCOPY
//***********************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.JCLS,DISP=SHR
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.JCLS.UNLOAD,
//         DISP=(NEW,CATLG),
//         UNIT=CART,
//         DCB=(LRECL=80,RECFM=FB,BLKSIZE=0)
//SYSIN    DD *
  COPY INDD=SYSUT1,OUTDD=SYSUT2
/*
```

Loading (Restoring Unloaded copy from Tape to DASD)

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=J,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBCOPY
//**********************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.JCLS.UNLOAD,DISP=SHR
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.JCLS.LOADED,
//         DISP=(NEW,CATLG),
//         UNIT=TESTDA,
//         SPACE=(CYL,(10,10,40),RLSE),
//         DCB=(LRECL=80,RECFM=FB,BLKSIZE=0)
//SYSIN    DD *
  COPY INDD=SYSUT1,OUTDD=SYSUT2
/*
```

Compressing data sets

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=J,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBCOPY
//***********************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.JCLS,DISP=SHR
//SYSIN    DD *
  COPY INDD=SYSUT1,OUTDD=SYSUT1
/*
```

Including members of PDS in a COPY command

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=J,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBCOPY
//**********************************************************
*************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.PDSE.JCLS,DISP=SHR
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.JCLS.IEBCOPY,
//         DISP=(NEW,CATLG,DELETE),
//         SPACE=(CYL,(5,5,5),RLSE),
//         DCB=(LRECL=80,RECFM=FB,BLKSIZE=800)
//SYSIN    DD *
  COPY INDD=SYSUT1,OUTDD=SYSUT2
  SELECT MEMBER=(IEBCOPY1,IEBCOPY2,IEBCOPY3,IEBCOPY4,      -
                 IEBCOPY5,IEBCOPY6)
/*
//
```

Excluding members of PDS in a COPY command

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBCOPY
//**********************************************************
*************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.PDSE.JCLS,DISP=SHR
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.JCLS.OTHERS,
//         DISP=(NEW,CATLG,DELETE),
//         SPACE=(CYL,(10,10,40),RLSE),
//         DCB=(LRECL=80,RECFM=FB,BLKSIZE=800)
//SYSIN    DD *
  COPY INDD=SYSUT1,OUTDD=SYSUT2
  EXCLUDE MEMBER=(IEBCOPY1,IEBCOPY2,IEBCOPY3,IEBCOPY4,    -
                 IEBCOPY5,IEBCOPY6)
/*
//
```

Renaming a member while copying

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=J,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBCOPY
//**********************************************************
*******
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.PDSE.JCLS,DISP=SHR
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.JCLS.IEBCOPY,
//         DISP=(NEW,CATLG,DELETE),
//         SPACE=(CYL,(5,5,5),RLSE),
//         DCB=(LRECL=80,RECFM=FB,BLKSIZE=800)
//SYSIN    DD *
  COPY INDD=SYSUT1,OUTDD=SYSUT2
  SELECT MEMBER=(IEBCOPY1,IEBCOPY2,IEBCOPY3,IEBCOPY4,
                IEBCOPY5,(IEBCOPY6,NEWIEBC6))
/*
//
```

Merging Partitioned Data Sets

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBCOPY
//***********************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.JCLS.IEBCOPY,DISP=SHR
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.JCLS.OTHERS,DISP=SHR
//SYSUT3   DD DSN=XXXX.JCLTEST.RIJO.JCLS.ALL,
//         DISP=(NEW,CATLG,DELETE),
//         SPACE=(CYL,(10,10,40),RLSE),
//         DCB=(LRECL=80,RECFM=FB,BLKSIZE=800)
//SYSIN    DD *
  COPY INDD=(SYSUT1,SYSUT2),OUTDD=SYSUT3
/*
//
```

IEBGENER

IEBGENER is a generalized copy utility used to perform the following tasks

- Produce a backup copy of a sequential data set or a member of a PDS or PDSE
- Produce a PDS or PDSE, or a member of either, from a sequential file
- Produce a printed list of either sequential data sets or PDS/PDSE members
- Re block a data set or change its logical record length

Sample IEBGENER JCL

```
//JS10     EXEC PGM=IEBGENER,REGION=1024K
//SYSPRINT DD SYSOUT=*                  *MESSAGES
//SYSUT1   DD DSN=…,DISP=…              *SEQUENTIAL INPUT FILE
//SYSUT2   DD DSN=…,DISP=…              *OUTPUT FILE
//SYSIN    DD *
  CONTROL STATEMENTS
/*
```

Examples

Using IEBGENER to create a data set from in stream data

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBGENER
//*****************************************************************
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT1   DD *
AJITH              TVM              MCA            15000     12345678
AJAY               TVM              MCA            18000     11111111
ANIL               TVM              BTECH          15000     22222222
SAJAN              TVM              MCA            18000     12121212
KIRAN              TVM              MCA            15000     33333333
THOMAS             TVM              BTECH          18000     13131313
PHILIP             TVM              MCA            15000     21212121
MANU               TVM              MCA            18000     34343434
JAYA CHANDRAN      TVM              BTECH          15000     45544554
ARUN               TVM              BTECH          18000     78654321
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.WORK.SRTINPUT,
//         DISP=(NEW,CATLG,DELETE),
//         UNIT=TESTDA,
//         SPACE=(CYL,(1,1),RLSE),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//
```

Using IEBGENER to create back up of a PDS/PDSE member to another PDS/PDSE member

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBGENER
//*************************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.PDSE.JCLS(IEBGEN2),
//         DISP=SHR
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.PDSE.JCLS(IEBGEN3),
//         DISP=SHR,
//         UNIT=TESTDA,
//         SPACE=(CYL,(1,1),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN      DD DUMMY
//SYSUDUMP DD SYSOUT=*
```

Using IEBGENER to create a back up copy of a sequential data set to another sequential data set

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//               CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1      EXEC PGM=IEBGENER
//*****************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1     DD DSN=XXXX.JCLTEST.RIJO.WORK.SRTINPUT,
//           DISP=SHR
//SYSUT2     DD DSN=XXXX.JCLTEST.RIJO.WORK.SRTINBK,
//           DISP=(NEW,CATLG,DELETE),
//           UNIT=TESTDA,
//           SPACE=(CYL,(1,1),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN      DD DUMMY
//SYSUDUMP DD SYSOUT=*
```

Using IEBGENER to create a back up copy of a sequential data set to a member of a PDS/PDSE

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//               CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1      EXEC PGM=IEBGENER
//****************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1     DD DSN=XXXX.JCLTEST.RIJO.WORK.SRTINPUT,
//           DISP=SHR
//SYSUT2     DD DSN=XXXX.JCLTEST.RIJO.WORK(SRTINBK),
//           DISP=SHR,
//           UNIT=TESTDA,
//           SPACE=(CYL,(1,1),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN      DD DUMMY
//SYSUDUMP DD SYSOUT=*
```

Using IEBGENER to create a back up copy of a PDS/PDSE member to a sequential data set

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//               CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1      EXEC PGM=IEBGENER
//*****************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1     DD DSN=XXXX.JCLTEST.RIJO.WORK(SRTINBK),
//           DISP=SHR
```

```
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.WORK.SRTBKUP,
//         DISP=(NEW,CATLG,DELETE),
//         UNIT=TESTDA,
//         SPACE=(CYL,(1,1),RLSE),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN    DD DUMMY
//SYSUDUMP DD SYSOUT=*
```

Using IEBGENER to produce PDSE from a PDS member

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBGENER
//*******************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.WORK(SRTINBK),
//         DISP=SHR
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.PDSE.WORK(SRTINBK),
//         DISP=(NEW,CATLG,DELETE),
//         UNIT=TESTDA,
//         DSORG=PO,
//         DSNTYPE=LIBRARY,
//         SPACE=(CYL,(1,1),RLSE),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN    DD DUMMY
//SYSUDUMP DD SYSOUT=*
```

Using IEBGENER to change the block size of an existing sequential data set to another new sequential data set

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEBGENER
//********************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.WORK.SRTINPUT,
//         DISP=SHR,
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.WORK.SRTINPUT.NEWBS,
//         DISP=(NEW,CATLG,DELETE),
//         UNIT=TESTDA,
//         SPACE=(CYL,(1,1),RLSE),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=32000)
//SYSIN    DD DUMMY
//SYSUDUMP DD SYSOUT=*
```

You will get error if you try the following JCL

```
//RQRIJOJB JOB (RQRIJOJ,SAMPLE),'RIJO JOSEPH',
//             CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//*
```

```
//STEP1    EXEC PGM=IEBGENER
//**********************************************************
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=XXXX.JCLTEST.RIJO.WORK,
//         DISP=SHR
//SYSUT2   DD DSN=XXXX.JCLTEST.RIJO.WORK.BKUP,
//         DISP=(NEW,CATLG,DELETE),
//         UNIT=TESTDA,
//         SPACE=(CYL,(10,10,100),RLSE),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN    DD DUMMY
//SYSUDUMP DD SYSOUT=*
```

Where, `XXXX.JCLTEST.RIJO.WORK and XXXX.JCLTEST.RIJO.WORK.BKUP` are PDS

You will get errors like this.

```
E1        8   DSS20027E    DDNAME 'SYSUT1', DSN
                           'XXXX.JCLTEST.RIJO.WORK' SHOULD HAVE
                           BEEN A PHYSICAL SEQUENTIAL DATA SET.
                           ACTUAL DSORG IS PO.
E2        8   DSS20027E    DDNAME 'SYSUT2', DSN
                           'XXXX.JCLTEST.RIJO.WORK.BKUP' SHOULD
                           HAVE BEEN A PHYSICAL SEQUENTIAL DATA
                           SET. ACTUAL DSORG IS PO.
```

5. DB2

NULL INDICATOR

To retrieve data from a nullable column, use null indicators.

Syntax:
INTO :HOSTVAR:NULLIND
The picture clause that should be used for the null indicator is S9(4) COMP

Meaning:

-1	: the field is null
0	: the field is not null
-2	: the field value is truncated

Inserting a record with a nullable column
To insert a NULL, move -1 to the null indicator
To insert a valid value, move 0 to the null indicator

DBRM:

DataBase Request Module, has the SQL statements extracted from the host language program by the SQL precompile.

PLAN:

A result of the BIND process. It has the executable code for the SQL statements in the DBRM. Plan is marked as invalid if the index used by it is dropped. The next time the plan is invoked, it is recreated.

SYNONYM

Synonym is an alternate name for a table or view. A synonym is accessible only by the creator.

VIEWS

View is nothing but a stored query. Views are not supported by their own, physically separate, distinguishable stored data. Instead, their definition in terms of other tables is stored in the catalog table (SYSVIEWS).

For example

```
CREATE  VIEW  GOOD_SUPPLIERES
        AS  SELECT   S#, STATUS, CITY
            FROM     S
            WHERE   STATUS > 15;
```

When this CREATE VIEW is executed, the sub query following the AS is not executed; instead it is simply saved in the catalog, under the specified view name (GOOD-SUPPLIERS)

GOOD_SUPPLIERS is in effect a "window" into the real table S. Furthermore that window is dynamic: changes to S will be automatically and instantaneously visible through that window. Likewise, changes to GOOD_SUPPLIERS will automatically and instantaneously be applied to the real table S.

Consider a retrieval operation.

```
SELECT   *
FROM     GOOD_SUPPLIERS
WHERE    CITY = 'LONDON' ;
```

The system handles such an operation by converting it into an equivalent operation on the underlying base table(s). In this example, the equivalent operation is

```
SELECT   S#, STATUS, CITY
FROM     S
WHERE    CITY = 'LONDON'
AND      STATUS > 15;
```

ARE THE VIEWS UPDATABLE ?

Not all of them. Some views are updatable e.g. single table view with all the fields or mandatory fields. Examples of non-updatable views are views which are joins, views that contain aggregate functions(such as MIN this is obvious).

HOW TO RUN A DB2 BATCH PROGRAM

Use DSN utility to run a DB2 batch program. An example is shown:

```
DSN SYSTEM(DSP3)
    RUN PROGRAM(EDD470BD)
    PLAN(EDD470BD)
    LIB('EDGS01T.OBJ.LOADLIB')
END
```

Use IKJEFT01 utility program to run this command in a JCL.

Why SELECT * is not preferred in embedded SQL programs?

For three reasons:

- If the table structure is changed (a field is added), the program will have to be modified
- Program might retrieve the columns which it might not use, leading on I/O over head.
- The chance of an index only scan is lost.

SYNONYM and ALIAS

SYNONYM: is dropped when the table or tablespace is dropped. Synonym is available only to the creator.

ALIAS: is retained even if table or tablespace is dropped. ALIAS can be created even if the table does not exist. It is used mainly in distributed environment to hide the location info from programs. Alias is a global object & is available to all.

How does DB2 store NULL physically?

High Values

How do you concatenate the FIRSTNAME and LASTNAME from EMP table to give a complete name?

SELECT FIRSTNAME || ‘ ‘ || LASTNAME FROM EMP;

What is the use of VALUE function?

Avoid handling Nullable fields. It assigns a default of 0 to numeric fields which are NULL

FETCH FIRST n ROWS ONLY

Fetching a limited number of rows: FETCH FIRST *n* ROWS ONLY

In some applications, you execute queries that can return a large number of rows, but you need only a small subset of those rows. Retrieving the entire result table from the query can be inefficient. You can specify the FETCH FIRST *n* ROWS ONLY clause in a SELECT statement to limit the number of rows in the result table of a query to *n* rows.

Example: Suppose that you write an application that requires information on only the 20 employees with the highest salaries. To return only the rows of the employee table for those 20 employees, you can write a query like this:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY
  FROM EMP
  ORDER BY SALARY DESC
  FETCH FIRST 20 ROWS ONLY;
```

OPTIMIZE FOR n ROWS

Minimizing overhead for retrieving few rows: OPTIMIZE FOR *n* ROWS

When an application executes a SELECT statement, DB2 assumes that the application will retrieve all the qualifying rows. This assumption is most appropriate for batch environments. However, for interactive SQL applications, such as SPUFI, it is common for a query to define a very large potential result set but retrieve only the first few rows. The access path that DB2 chooses might not be optimal for those interactive applications.

What OPTIMIZE FOR n ROWS does: The OPTIMIZE FOR *n* ROWS clause lets an application declare its intent to do either of these things:

- Retrieve only a subset of the result set
- Give priority to the retrieval of the first few rows

DB2 uses the OPTIMIZE FOR *n* ROWS clause to choose access paths that minimize the response time for retrieving the first few rows.

Use OPTIMIZE FOR 1 ROW to avoid sorts*:* You can influence the access path most by using OPTIMIZE FOR 1 ROW. OPTIMIZE FOR 1 ROW tells DB2 to select an access path that returns the first qualifying row quickly. This means that whenever possible, DB2 avoids any access path that involves a sort. If you specify a value for *n* that is anything but 1, DB2 chooses an access path based on cost, and you won't necessarily avoid sorts.

How many rows you can retrieve with OPTIMIZE FOR n ROWS*:* The OPTIMIZE FOR *n* ROWS clause does not prevent you from retrieving all the qualifying rows. However, if you use OPTIMIZE FOR *n* ROWS, the total elapsed time to retrieve all the qualifying rows might be significantly greater than if DB2 had optimized for the entire result set.

When OPTIMIZE FOR n ROWS is effective: OPTIMIZE FOR *n* ROWS is effective only on queries that can be performed incrementally. If the query causes DB2 to gather the whole result set before returning the first row, DB2 ignores the OPTIMIZE FOR *n* ROWS clause, as in the following situations:

- The query uses SELECT DISTINCT or a set function distinct, such as COUNT(DISTINCT C1).
- Either GROUP BY or ORDER BY is used, and no index can give the necessary ordering.
- An aggregate function and no GROUP BY clause is used.
- The query uses UNION.

Example: Suppose that you query the employee table regularly to determine the employees with the highest salaries. You might use a query like this:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY
 FROM EMP
 ORDER BY SALARY DESC;
```

An index is defined on column EMPNO, so employee records are ordered by EMPNO. If you have also defined a descending index on column SALARY, that index is likely to be very poorly clustered. To avoid many random, synchronous I/O operations, DB2 would most likely use a table space scan, then sort the rows on SALARY. This technique can cause a delay before the first qualifying rows can be returned to the application.

If you add the OPTIMIZE FOR *n* ROWS clause to the statement, DB2 will probably use the SALARY index directly because you have indicated that you expect to retrieve the salaries of only the 20 most highly paid employees.

Example: The following statement uses that strategy to avoid a costly sort operation:

```
SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
 FROM EMP
 ORDER BY SALARY DESC
 OPTIMIZE FOR 20 ROWS;
```

Effects of using OPTIMIZE FOR n ROWS:

- ❖ The join method could change. Nested loop join is the most likely choice, because it has low overhead cost and appears to be more efficient if you want to retrieve only one row.
- ❖ An index that matches the ORDER BY clause is more likely to be picked. This is because no sort would be needed for the ORDER BY.
- ❖ List prefetch is less likely to be picked.
- ❖ Sequential prefetch is less likely to be requested by DB2 because it infers that you only want to see a small number of rows.
- ❖ In a join query, the table with the columns in the ORDER BY clause is likely to be picked as the outer table if there is an index on that outer table that gives the ordering needed for the ORDER BY clause.

RUNSTATS

RUNSTATS is a DB2 utility to collect statistics about the data values in tables which can be used by the optimizer to decide the access path. These statistics are stored in DB2 catalog tables and is critical in determining access paths for an SQL statement.

IMAGECOPY

IMAGECOPY is a full backup of a DB2 table which can be used in recovery.

DECIMAL(n,m)

For SQL declaration of DECIMAL(n,m), the COBOL equivalent generated is

PIC S9(n-m)V9(m) COMP-3

COALESCE Function in DB2.

The COALESCE function avoids the use of nulls in the result column of OUTER JOIN statements. Consider the following query

```
SELECT EMP.EMPNO, EMP.WORKDEPT, DEPT.DEPTNAME
  FROM EMP FULL OUTER JOIN DEPT
    ON EMP.WORKDEPT = DEPT.DEPTNO;
```

Let us say the result of this query looks as follows:

EMPNO	WORKDEPT	DEPTNAME
3089	S12	SOFTWARE QUALITY
7654	A09	CORPORATE ADMIN
....	...	TRAINING DEPT

It can be seen that the TRAINING DEPT's WORKDEPT is not shown in the table because the query selects WORKDEPT from EMP, not the DEPTNO column from DEPT. This can be rectified using the COALESCE function. The COALESCE function notifies DB2 to look for a value in the listed columns, one from each table in the outer join. If this value is found in either of the table it can be returned as a result. The following example depicts the use of COALESCE

```
SELECT EMP.EMPNO
       COALESCE(EMP.WORKDEPT, DEPT.DEPTNO) AS DEPTNUM,
       DEPT.DEPTNAME
  FROM EMP FULL OUTER JOIN DEPT
    ON EMP.WORKDEPT = DEPT.DEPTNO;
```

The O/P for the above query would be now shown as follows...

EMPNO	DEPTNUM	DEPTNAME
3089	S12	SOFTWARE QUALITY
7654	A09	CORPORATE ADMIN
....	**T01**	TRAINING DEPT

We can see that the COALESCE function determines that the department code is stored in the DEPT.DEPTNO column and returns that value instead of the null because there is no corresponding WORKDEPT number.

QMF (Query Management Facility)

You can access QMF either through TSOA or TSOB. TSOA is test/development region and TSOB is production region. That is if you access DB2 through TSOB, you will be directly accessing production data (that is by default, the system will be pointing to production databases)

- You can set the profile by selecting PF 11 from the QMF main panel. One important setting is the CONFIRM, if you set it to YES, the system will ask for a confirmation before you delete a table/proc etc… Printer is the 4 character Printer ID installed in the Mainframe system and the user can direct the output of a query to the printer using the 4=Print option in the query panel (by PF4). Other options include Upper/Lower case, Space (the name of DB2 DATABASE or TABLESPACE in which tables will be saved by the SAVE DATA command – explained later) etc..

- **Draw function** enables the user to list all the fields in a table by entering the table name in the command area (in the query panel) and by selecting the Draw option (PF 6). For example, if you enter the table name ABCD.CUSTOMER_INFO in the command line and press the PF 6 option, it will list a SELECT statement with all the fields in the table without a where clause. This option is useful when you don't know the fields in the table. If you want to delete a line from the listing thus obtained, you can do so by selecting the line and by pressing PF 11. To insert a new line (for example to add where clause) you can place the cursor at the bottom of the query and press PF 10.

- **Save Query As** - You can save a query for future reference by the option SAVE QUERY AS <query name>. Another option with saving a query is SAVE QUERY AS <query name> (SHARE = ALL). If you use this option, the query will be available to other users also. To do this, after typing the query, issue the save query as command. To View the query you have just saved, you can issue the command LIST QUERIES (OWNER = <your id>).

- Once you have saved the query, you can retrieve using LIST QUERIES (OWNER = <your id>) from the QMF main panel. This will list all the queries under your id.

    - Use DIS option to display a query
    - Run option to run the query
    - ERASE option to erase the query from the system

- **Forms** to format the output of a query

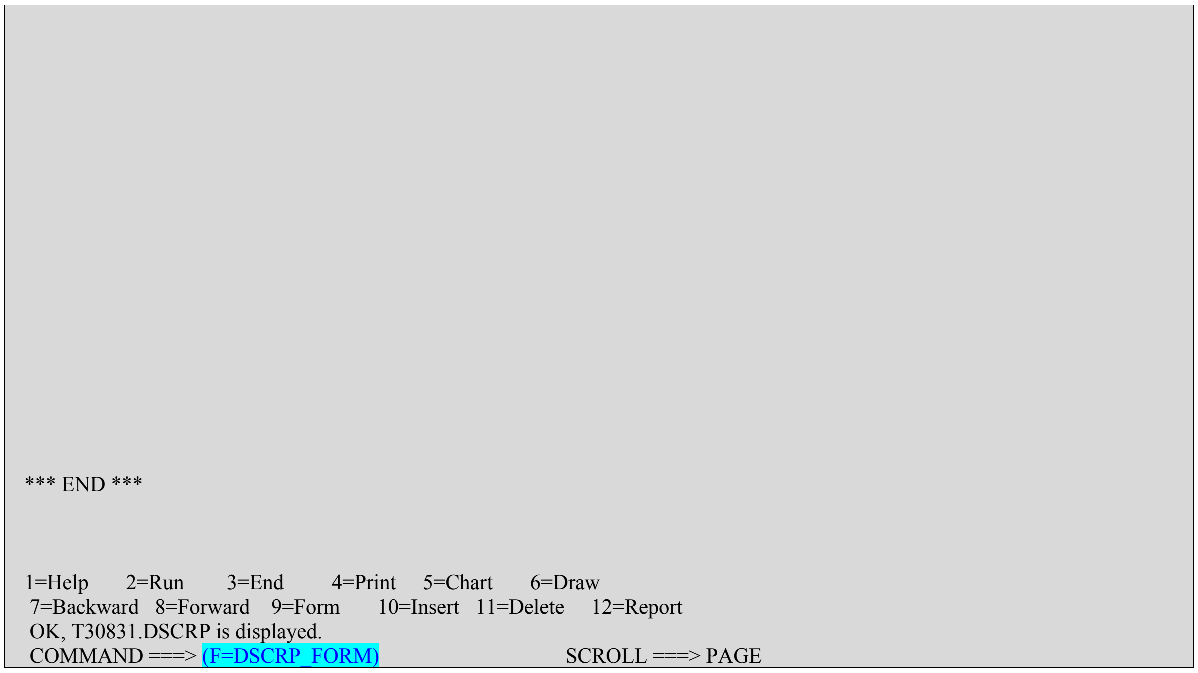
  We can use the FORM option to format the output of a QMF Query. Press PF 9 after a query has run. This will show the default form for that query. Where you can change the form and use this while running your query. A sample form is given below

```
FORM.MAIN

 COLUMNS:            Total Width of Report Columns: 346
  NUM COLUMN HEADING                        USAGE   INDENT WIDTH EDIT  SEQ
  --- ---------------------------------------- ------- ------ ----- ----- ---
   1 N_EMPL_INIT_TEST                          2     4    C    1
   2 N_EMPL_LST_RSLV                           2     14   C    2
   3 I_DSCRP                              2     10   L    3
   4 C_DSCRP_PRTY_TEST                          2     5    C    4
   5 N_TEST_CASE_NO                           2     7    C    5
   6 D_EST_DLVRY                            2     10   TD   6
   7 T_DSCRP_ABBR_DESC                          2     40   C    7
   8 T_DSCRP_DESC                           2     240  C    8
    *** END ***

 PAGE:    HEADING  ===>
        FOOTING  ===>
 FINAL:   TEXT     ===>
 BREAK1:  NEW PAGE FOR BREAK? ===> NO
        FOOTING  ===>
 BREAK2:  NEW PAGE FOR BREAK? ===> NO
        FOOTING  ===>
 OPTIONS: OUTLINE? ===> YES             DEFAULT BREAK TEXT? ===> YES

 1=Help     2=Check    3=End        4=Show        5=Chart       6=Query
 7=Backward 8=Forward  9=           10=Insert      11=Delete      12=Report
 OK, FORM is displayed.
 COMMAND ===> SAVE FORM AS DSCRP_FORM                    SCROLL ===> PAGE
```

Here also you can use the SHARE=Y option to make it available for all the users. Later you can use the LIST ALL option to get this form and modify it according to your requirement. The modified form can be saved by using the SAVE option.

While selecting the query to run, you can specify the form option in the command line to get your display formatted according to your form (as shown below)

```
SQL QUERY         T30831.DSCRP                          LINE    1

 SELECT N_EMPL_INIT_TEST,
 N_EMPL_LST_RSLV, I_DSCRP, C_DSCRP_PRTY_TEST,
 N_TEST_CASE_NO,D_EST_DLVRY,T_DSCRP_ABBR_DESC, T_DSCRP_DESC
 FROM DB27.TCSS_DSCRP
```

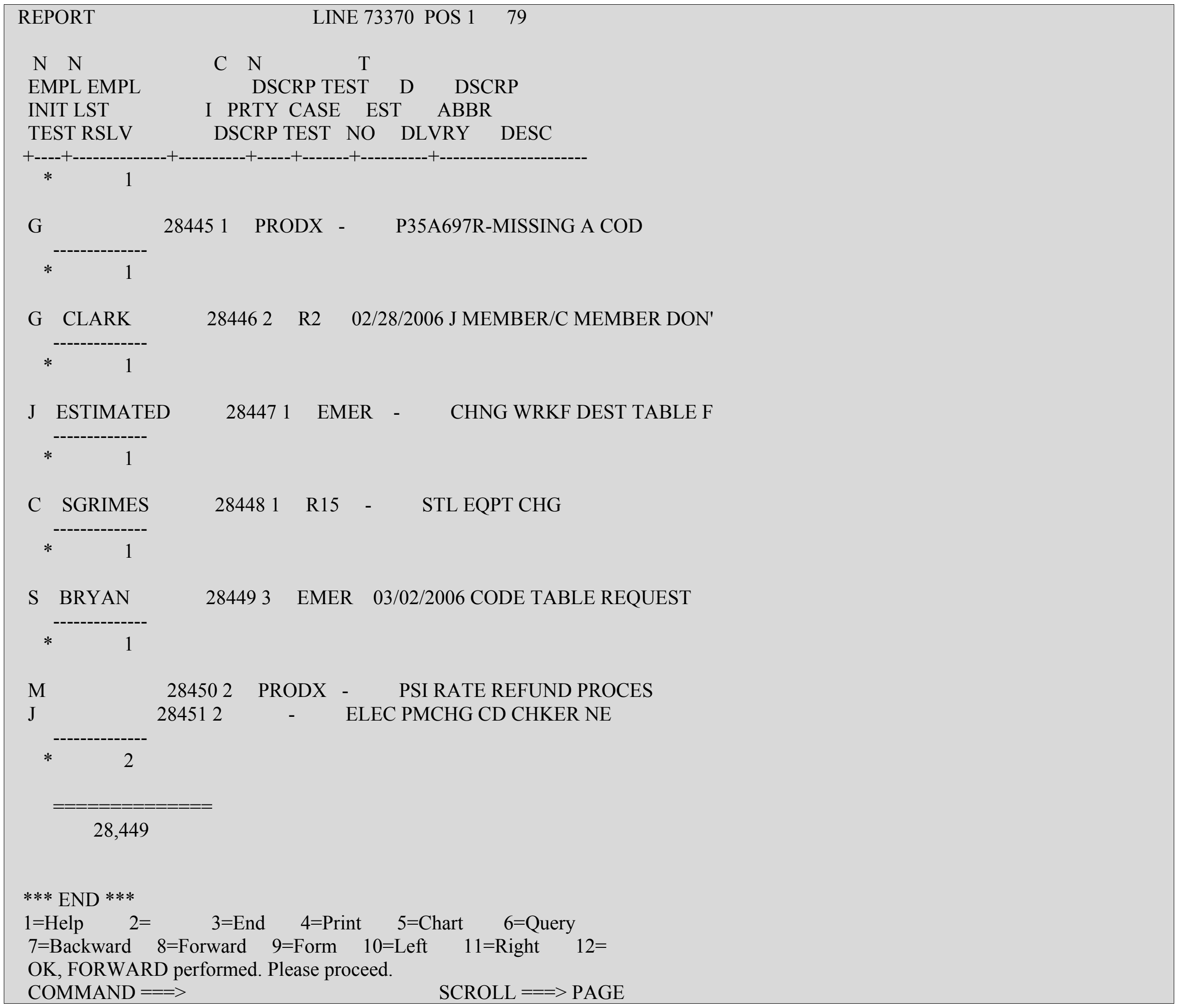


```
*** END ***



1=Help      2=Run       3=End        4=Print     5=Chart      6=Draw
 7=Backward   8=Forward    9=Form       10=Insert   11=Delete     12=Report
 OK, T30831.DSCRP is displayed.
 COMMAND ===> (F=DSCRP_FORM)                              SCROLL ===> PAGE
```

You can change the width, indent etc… with Forms

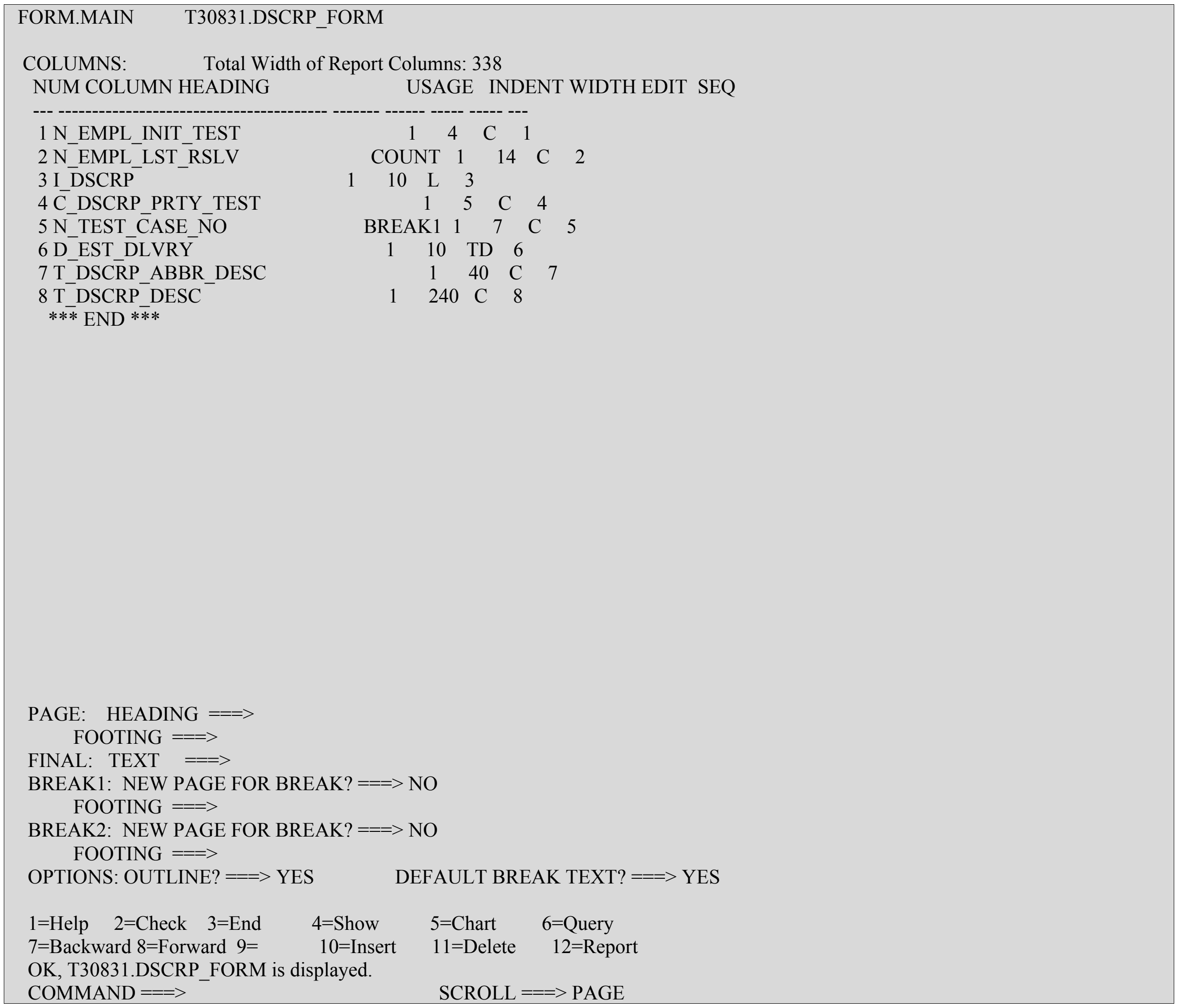
More on FORMS

See the output display in QMF



```
REPORT                              LINE 73370  POS 1      79

  N    N                      C    N                T
 EMPL EMPL                      DSCRP TEST      D        DSCRP
 INIT LST                   I   PRTY  CASE     EST       ABBR
 TEST RSLV                  DSCRP TEST   NO     DLVRY      DESC
+----+--------------+----------+-----+-------+----------+----------------------
    *           1

 G                    28445 1     PRODX   -         P35A697R-MISSING A COD
     --------------
    *           1

 G    CLARK             28446 2     R2      02/28/2006 J MEMBER/C MEMBER DON'
     --------------
    *           1

 J    ESTIMATED          28447 1     EMER    -         CHNG WRKF DEST TABLE F
     --------------
    *           1

 C    SGRIMES            28448 1     R15     -         STL EQPT CHG
     --------------
    *           1

 S    BRYAN              28449 3     EMER    03/02/2006 CODE TABLE REQUEST
     --------------
    *           1

 M                    28450 2     PRODX   -         PSI RATE REFUND PROCES
 J                   28451 2             -         ELEC PMCHG CD CHKER NE
     --------------
    *           2

     ==============
          28,449


*** END ***
1=Help        2=          3=End      4=Print      5=Chart       6=Query
 7=Backward    8=Forward     9=Form     10=Left      11=Right       12=
 OK, FORWARD performed. Please proceed.
 COMMAND ===>                                          SCROLL ===> PAGE
```

Here, there is a break on the N_TEST_CASE_NO and a count on N_EMPL_LST_RSLV. This can be obtained by specifying a form shown in the next page



```
FORM.MAIN         T30831.DSCRP_FORM

 COLUMNS:             Total Width of Report Columns: 338
  NUM COLUMN HEADING                        USAGE   INDENT WIDTH EDIT  SEQ
  --- ---------------------------------------- ------- ------ ----- ----- ---
   1 N_EMPL_INIT_TEST                           1     4    C    1
   2 N_EMPL_LST_RSLV                    COUNT  1     14   C    2
   3 I_DSCRP                         1     10   L    3
   4 C_DSCRP_PRTY_TEST                          1     5    C    4
   5 N_TEST_CASE_NO                   BREAK1  1     7    C    5
   6 D_EST_DLVRY                         1     10   TD   6
   7 T_DSCRP_ABBR_DESC                          1     40   C    7
   8 T_DSCRP_DESC                        1     240  C    8
    *** END ***

 PAGE:    HEADING  ===>
        FOOTING  ===>
 FINAL:   TEXT     ===>
 BREAK1:  NEW PAGE FOR BREAK? ===> NO
        FOOTING  ===>
 BREAK2:  NEW PAGE FOR BREAK? ===> NO
        FOOTING  ===>
 OPTIONS: OUTLINE? ===> YES              DEFAULT BREAK TEXT? ===> YES

 1=Help     2=Check    3=End        4=Show        5=Chart        6=Query
 7=Backward 8=Forward  9=           10=Insert      11=Delete       12=Report
 OK, T30831.DSCRP_FORM is displayed.
 COMMAND ===>                                          SCROLL ===> PAGE
```

You can also change the width of a field to format the display. SUM, AVERAGE also can be used to format the output for numeric items.

To list all the forms, you can use LIST FORMS or LIST FORMS (OWNER = <your id>). You can use ERASE option to delete it.

Saving the output of a Query

The output of a query can be saved as a table name and you can use the same in another query instead of a table.
This can be done by using the option SAVE DATA AS <table name>. To view all the tables that you have saved, you can use the option LIST TABLES option from the QMF main panel.

Additional Information

- You can use wild card characters while using the LIST option. For example you can use LIST QUERIES (NAME = %WEST%). This will list all queries with 'WEST' in their name.

- From the QMF main panel, you can issue PF 7 to retrieve the previous commands. Another way by which you can retrieve the previous commands is by the use of a Question mark (?) in the command line. You can use more than one question mark at the same time to skip that many commands.

- You can't use (SHARE = Y) option with SAVE DATA AS.

- You can use the reset option to clear out the queries/results in the QMF query panel

- You can use PF 6 to refresh the LIST Object panel

- Even if you have logged on to TSOA, you can access the production version of the data by using the CONNECT TO <db name> command in the query panel. By default, TSOA will point to the test region databases, you can use the CONNECT to make it point to the production version.

- In the QMF query panel, you can issue the EDIT command in the command line to open up a TSO/ISPF like editor. Where you can use all the line commands to edit the queries.

- If a query is running for a long time and to cancel the same you can use Shift + Esc key twice.

- You can give a comment with the query by prefixing it with –

- You can use the short for O instead of OWNER in LIST command

6. IMS

DEFINING THE IMS DATABASE

In the IMS, this takes the form of a Data Base Description (DBD) step. In this step, one defines a Data Base Description (DBD), which is processed, and the resulting information is maintained in an IMS library

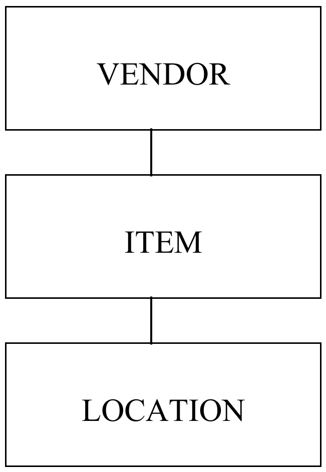
A DBD describes the complete structure of a database. An installation must create one DBD for each DL/I database

Although each database has a single physical structure that's defined by a DBD, the application programs that process it can have different views of it. These views, also called application data structures, specify the databases (one or more) a program can access, the data elements the program can "see" in those databases, and the processing the program can do. This information is specified in a PSB (Program Specification Block)

The DBA codes the assembler language statements necessary to define a DBD or PSB, then assembles and links them, storing the resulting load module in a PDS. This process is called control block generation; generating a DBD is usually called a DBDGEN, and generating a PSB is called PSBGEN.

Let us consider an Inventory Database

The structure of the database is



DBD

```
DBD         NAME = INDBD
*
SEGM        NAME = INVENSEG, PARENT=0, BYTES = 63
FIELD       NAME = (INVENCOD, SEQ), BYTES = 3, START = 1, TYPE = C
FIELD       NAME = INVENNAM, BYTES = 30, START = 4, TYPE = C
FIELD       NAME = INVENADR, BYTES = 30, START = 34, TYPE = C

*
SEGM        NAME = INITMSEG, PARENT=INVENSEG, BYTES = 44
FIELD       NAME = (INITMNUM, SEQ), BYTES = 5, START = 1, TYPE = C
FIELD       NAME = INITMDES, BYTES = 35, START = 6, TYPE = C
FIELD       NAME = INITMPRC, BYTES = 4, START = 41, TYPE = P

*
SEGM        NAME = INLOCSEG, PARENT=INITMSEG, BYTES = 44
FIELD       NAME = (INLOCLOC, SEQ), BYTES = 3, START = 1, TYPE = C
FIELD       NAME = INLOCONH, BYTES = 4, START = 4, TYPE = P
FIELD       NAME = INLOCROP, BYTES = 4, START = 8, TYPE = P
FIELD       NAME = INLOCONO, BYTES = 4, START = 12, TYPE = P
FIELD       NAME = INLOCDAT, BYTES = 6, START = 16, TYPE = C

*
```

PSB

The first macro in the PSBGEN job stream is PCB (Program Communication Block). The PCB describes one database. A PSBGEN job contains one PCB macro for each database the application program can access

A sample PSB for the above DBD is given below

```
PCB         TYPE = DB, DBDNAME = INDBD, KEYLEN = 11, PROCOPT = LS
SENSEG      NAME = INVENSEG
SENSEG      NAME = INITMSEG, PARENT = INVENSEG
SENSEG      NAME = INLOCSEG, PARENT = INITMSEG
```

The KEYLEN parameter specifies the length of the longest concatenated key the program can process in the database

PROCOPT specifies the program’s processing options. They indicate what processing the program is allowed to perform on the database

LS indicate that the program can perform only load operations

The DBA can code the PROCOPT parameter on the SENSEG macro to control access to the database more selectively than is possible at the database level

```
PCB          TYPE = DB, DBDNAME = INDBD, KEYLEN = 11
SENSEG       NAME = INVENSEG, PROCOPT = K
SENSEG       NAME = INITMSEG, PARENT = INVENSEG, PROCOPT = K
SENSEG       NAME = INLOCSEG, PARENT = INITMSEG, PROCOPT = G
```

This means that the user may not access the data within this segment but may use it only for traversal of the hierarchy. A PROCOPT of K indicates that key sensitivity only. A GN call with no SSAs can access only data sensitive segments. If a key sensitive segment is designated for retrieval in an SSA, the segment is not moved to the user's I/O area. The key is placed at the appropriate offset in the key feedback area of the PCB.

If there is a difference in the processing options specified on the PCB and SENSEG statements and the options are compatible, SENSEG PROCOPT overrides the PCB PROCOPT

IMS DL/I DATA BASE TYPES

During installation, the database administrator (DBA) chooses the type of database to use for the IMS-DL/I databases. The DBA decides which type of database to use based on how most of the programs that use an IMS-DL/I database will access the data in the database. The following is a list of database types that the DBA can use to define an IMS-DL/I database

Data Entry Data Base (DEDB)

It is a direct-access database that consists of one or more areas, with each area containing both root segments and dependent segments. The database is accessed using VSAM improved control interval processing (ICIP).

Main Storage Data Base (MSDB)

It is a root-segment database, residing in main storage, which can be accessed to a field level.

Hierarchical Direct Access Method (HDAM)

It is one of DL/I's two direct-access methods. A direct-access method allows DL/I to locate any database record, regardless of the record sequence in the database, by using a randomizing routine or an index. HDAM provides direct access to data through a randomizing routine. Sequentially accessing an HDAM database, DL/I retrieves data in the order that the data are physically stored in the database.

Hierarchical Indexed Direct Access Method (HIDAM)

It is one of DL/I's two direct-access methods. HIDAM provides direct access to data through an index.

Hierarchical Sequential Access Method (HSAM)

It is one of DL/I's sequential-access methods. In a sequential-access database, segments are stored in a hierarchical sequence, one segment after another. HSAM provides sequential access to root segments and dependent segments. You can access data in HSAM databases, but you cannot update any of the data.

Hierarchical Indexed Sequential Access Method (HISAM)

You can processes data sequentially, but it has an index that enables you to directly access records in the database.

Generalized Sequential Access Method (GSAM)

It allows IMS/ESA batch application programs to access a sequential OS/390 data set record that is defined as a database record. This database record is handled as one unit, with no segments, fields, or hierarchical structure. Any records to be added are inserted at the end of the database. GSAM does not allow you to update or delete records in the database.

Simple Hierarchical Sequential Access Method (SHSAM)

It is an HSAM database that contains only one segment type, a root segment. Only two types of calls are valid with SHSAM databases: Get calls to read a database and Insert calls to load a database. You must reload a database in order to update it.

Simple Hierarchical Indexed Sequential Access Method (SHISAM)

It is a HISAM database with only one segment type, a root segment.

PROCOPT

It is the parameter keyword for the processing options on sensitive segments declared in the PCB that you can use in an associated application program.

G Get Function

I Insert Function

R Replace Function, Includes G.

D Delete Function, Includes G.

A All, includes the above four functions. PROCOPT = A is the default setting

P Required if command code D is to be used

O If the O option is used for a PCB, IMS doesn't check the ownership of the segments returned. Therefore, the read without integrity program might get a segment that has been updated by another program. If the updating program abends and backs out, the read without integrity program will have a segment that doesn't exist in the database and never did.

N Reduces the number of abends that read only application programs are subject to. Read only application programs can reference data being updated by another application program. When this happens, an invalid pointer to the data might exist. If an invalid pointer is detected, the read only application program abends. By specifying N, you avoid this. A GG status code is returned to the program instead. The program must determine whether to terminate processing, continue processing by reading a different segment, or access the data using a different path.

T T is the same as the N operand, except that T causes DL/I to automatically retry the operation. If the retry fails, a GG status code is returned to the application program.

E Enables exclusive use of the database or segment by online programs

L Load function for database loading (except HIDAM)

GS Get Segments in ascending sequence only (HSAM only). If you specify GS for HSAM databases, they will be read using the Queued Sequential Access Method (QSAM) instead of the Basic Sequential Access Method (BSAM) in a DL/I IMS region.

LS Segments loaded in ascending sequence only (HIDAM, HDAM). This load option is required for HIDAM. Because you must specify LS for HIDAM databases, the index for the root segment sequence field will be created at the time the database is loaded

H Specifies high speed sequential processing for the application program using a particular PSB. The restrictions for using PROCOPT = H are
It can be used for DEDBs only
It is allowed on the PCB level and not on the segment level.
It must be used with other Fast Path Processing options

A maximum of four PROCOPT options can be specified, including H
It can only be specified for BMP s
Only one PROCOPT = H PCB per database is allowed

If you don't specify the PROCOPT operand, it defaults to PROCOPT = A. The replace and delete functions also imply the get function

In a non-terminal related or fixed terminal related MSDB, only the processing options G and R are valid
In a dynamic terminal-related MSDB, the processing options G, I, R, D, A or any combination of G, I, R and D are valid
In a DEDB, the processing options G, I, R, D, A, P, N, T, O and H are valid

7. TSO ISPF

HRECALL

To recall a dataset that has been migrated without freezing your screen, type HRECALL next to the dataset name in 3.4

TSO SEND/TRANSMIT

To send short messages, use TSO SEND ‘message’ U(userid). If the person is not logged on but you want to send a message which he/she can read when he logs on , say TSO SEND ‘message’ U(userid) LOGON. To send Large information, use TSO TRANSMIT.

INFORMATION ABOUT YOUR DATASET

To know the percentage free space and other statistics of your data set, go to option 3.1 and select I. 3.1 provides other options also.

SAVING A CERTAIN SEQUENCE OF COMMANDS

The commands you use regularly can be saved as a function key. For example if you often change a JCL, save it , submit it and then go to the job spool, you could save a PF key as ‘SAVE;SUB,=IOF’. To do this , type KEYS on command line and make the necessary entries.

TAPE DATASET.

You can’t Rename/Delete a tape dataset. Only you can uncatalog it.

SB37 WHEN TRIED TO SAVE A FILE.

Your PDS is out of space. Start another session, open this PDS in 3.4 and type Z next to it, it will compress your PDS. All X37 abends are out of space situations.

NO SPACE IN DIRECTORY

If you get the above error message while creating a member in a PDS, which means your PDS can no longer accommodate more members in it. You need to increase the directory blocks. Each directory block can hold at least 4 members.

CREATE AND REPLACE COMMANDS (ISPF).

Suppose that you are editing your component in View mode and you want to save the changes. In such a case, you can use CREATE or REPLACE. The said example is just an instance and you can use CREATE/REPLACE for a variety of needs

CREATE

CREATE creates a member of a PDS or a sequential dataset from the data you are editing
Simplified syntax is
CREATE {member} {range}

Where member is the name of the new member and range is two labels that specify the group of lines, from beginning to end, which are added to the new member.

Note: CREATE adds a member to a PDS only if a member of the same name doesn't already exist.

You have to specify a range of lines from the data you are editing. If you want to use the entire data you are editing, you can use the system defined labels .ZF and .ZL (stands for the first and last line) , or you can label the lines according to your needs or you can use CC/MM to select the lines. But please note the difference between CC/MM.

You can specify the member in the command line or you can give CREATE .ZF .ZL and press enter, you will get the edit entry panel and you can specify the DSN.

If you want to create a member in the same PDS in which the component you are editing resides, you can give

CREATE .ZF .ZL NEWCMPNM

If it is in another PDS , you can give a command like

CREATE .ZF .ZL 'IPHT.XXX.YYY(ABC)'

Or

CREATE .ZF .ZL (or the range that you wish), and press enter; specify the DSN in the edit entry panel

REPLACE

The REPLACE primary command replaces a sequential data set or a member of a PDS with the data you are editing. If the member you want to replace doesn't exist, the editor creates it.

Simplified Syntax is

REPLACE {member} {range}

The rules for REPLACE are same as that of CREATE

CUT AND PASTE.

The CUT primary command saves lines to one of eleven named clipboards for later retrieval by the PASTE command. The lines can be appended to the lines already saved by a previous CUT command or can replace existing lines in a clipboard.

The syntax of the CUT command is

```
CUT {line pointer range} {DEFAULT | Clipboard name}
    {REPLACE | APPEND} {DISPLAY}
```

Where,

Line pointer range is

Two line pointers that specify the range of lines in the current member those are to be added to or replace data in the clipboard. A line pointer can be a label or relative line number. You must specify both a starting and ending line pointer. If you do not specify a range of lines, all lines in the edit session are copied to the clipboard.

E.g.: CUT .a .b

Clipboard name is

The name of the clipboard to use. If you omit this parameter, the ISPF default clipboard (named DEFAULT) is used. You can define up to ten additional clipboards. The size of the clipboards and number of clipboards might be limited by installation defaults.

```
E.g.: CUT CUT1
      PASTE CUT1

      CUT CUT2
      PASTE CUT2
```

REPLACE | APPEND is

Specify REPLACE to replace existing data in the clipboard.

Specify APPEND to add the data to the clipboard. You can select REPLACE or APPEND as the default by entering the EDITSET command on the editor command line. The default action depends on the setting specified in the panel displayed by the EDITSET.

DISPLAY is

Show a list of existing clipboards. From this list you can browse, edit, clear, or rename the clipboards.

E.g.: CUT DISPLAY.

Clipboard manager will pop up and will give the options to view, edit …

ISPF EDITOR COMMANDS

To search for a value say ‘0980312’ stored in COMP-3 format, you can use this method

F X ‘0980312’

To do a case sensitive search for a given search string, use this method.

Just enclose the search string within quotes and have the letter C in front of the string itself.

E.g. F C ‘FindMe’ will find all the ‘FindMe’ strings only if the case matches

To specify the direction of search i.e., how to search for a string in the backward/ forward direction (with respect to your current cursor position)?

Direction	Keyword	Example
Backward	PREV	F ‘job’ PREV finds for string ‘job’ in backward direction
Forward	No need to mention any keyword. It’s default direction	F ‘job’ finds for string ‘job’ in forward direction.
From the beginning of the file	FIRST	F ‘job’ FIRST finds the first instance of sting ‘job’ in the file.
Count of all instances in file	ALL	F ‘job’ ALL finds the first instance of sting ‘job’ in the file and gives *total-no-of-instances* of sting ‘job’ in the file.

To search for a string that is not prefix/suffix of another word. That is to find for an instance of a search string that is whole word by itself. To do this, you can use the following method.

Use the keyword ‘word’ at the end of the find command. That is
F job WORD will find for the whole word job

To go to a particular line number, we can use the following command

L ‘search string’. This is the Label command.

AUTOCOMPLETE FEATURE

We can have auto complete feature in ISPF like that we have while browsing the internet. Try the following.

1) Go to ISPF 3.4
2) Enter KEYS in the command line and press enter. The KEYS window will pop up.
3) Set any key to AUTOTYPE, save and exit. For example set F4 as AUTOTYPE.
4) Now type any dataset partially and press the assigned PF key.
5) You can use the assigned PF key to get the next match.

NOTE: It will not retrieve GDGs and VSAM files

NRETRIEV

This is a way of making the system remember previously typed dataset names.

1) Enter KEYS in the command line, the KEYS window will popup
2) Set any key to NRETRIEV and exit.
3) Now when you are in 3.4 or which ever screen you have set the KEYS option, press the assigned PF keys.
4) Pressing the assigned PF key will bring up one by one the last 30 accessed datasets from the REFLIST.

COBOL COMPILER

To know which version of COBOL compiler you are using, look at the first line of your compilation listing.

COMPARING 2 FILES OF DIFFERENT LAYOUTS

If you want to compare 2 files of different layouts you can use this form of 3.13

Suppose you want to compare the 2978:2979 (start_column:stop_column) of the new file with the 1565:1566 (start_column:stop_column - 2 Chars) of the old file, then here is the option that you may use.

Select 3.13
Give new and old DSNs

Point to the Options in the menu bar and press enter
Select Edit Statements (1) option
Give the options like this

CMPCOLMN 2978:2979
CMPCOLMO 1565:1566

The resulting JCL will look like this

```
//RQ123JOB JOB (00124,'SUPERC'),RJOSEPH,
//              CLASS=A,MSGCLASS=Z,REGION=0M,NOTIFY=&SYSUID
//SUPERC  EXEC PGM=ISRSUPC,
//             PARM=(CHNGL,WORDCMP,
//             '',
//             '')
//NEWDD  DD DSN=ISPS.RJOSEPH.RIJO.WORK(COMP1),
//          DISP=SHR
//OLDDD  DD DSN=ISPS.RJOSEPH.RIJO.WORK(COMP2),
//          DISP=SHR
//OUTDD  DD SYSOUT= *
//SYSIN  DD *
CMPCOLMN 2978:2979
CMPCOLMO 1565:1566
/*
```

Note: While comparing using CMPCOLM/ CMPCOLMN / CMPCOLMO, you can’t give file compare; you can give either line compare or word compare

You can either submit the Comparison operation as online or batch.

COUNTING THE NUMBER OF RECORDS IN A DASD/TAPE DATASET

TAPE DATASET

Suppose that the LRECL is 100 bytes, the block size is 1000 bytes then, if we know the block count, we can find the approximate number of records in the tape dataset by the following calculation

Number of records = (Block size * Block count) / (Record size).

In the above case, if we have only 10 blocks, then

Number of records = (1000* 10)/100
= 100

This is an approximate value because, **the last block may not be full.**

Note: To get the block count and all the details regarding the tape data set, you can use the V1.8 option (file aid)
Select V1.8/Select option 1 and give password as user.
Give DSN in the option 'Inquire/update TMC record by Data Set Name' without quotes/Press enter.
This would work for data sets that are not archived.

DASD DATASET

Suppose the allocation for a DASD data set is (TRK(10,2)).

We have the following information

1 Cylinder	= 15 Tracks
1 Track	= 56,664 bytes

In the above case, the maximum number of records that can be stored is

The total space allocation in the above case is 10+15*2 = 40 Tracks.

That is 40 * 56,664 bytes

Suppose that the LRECL is 80 bytes.

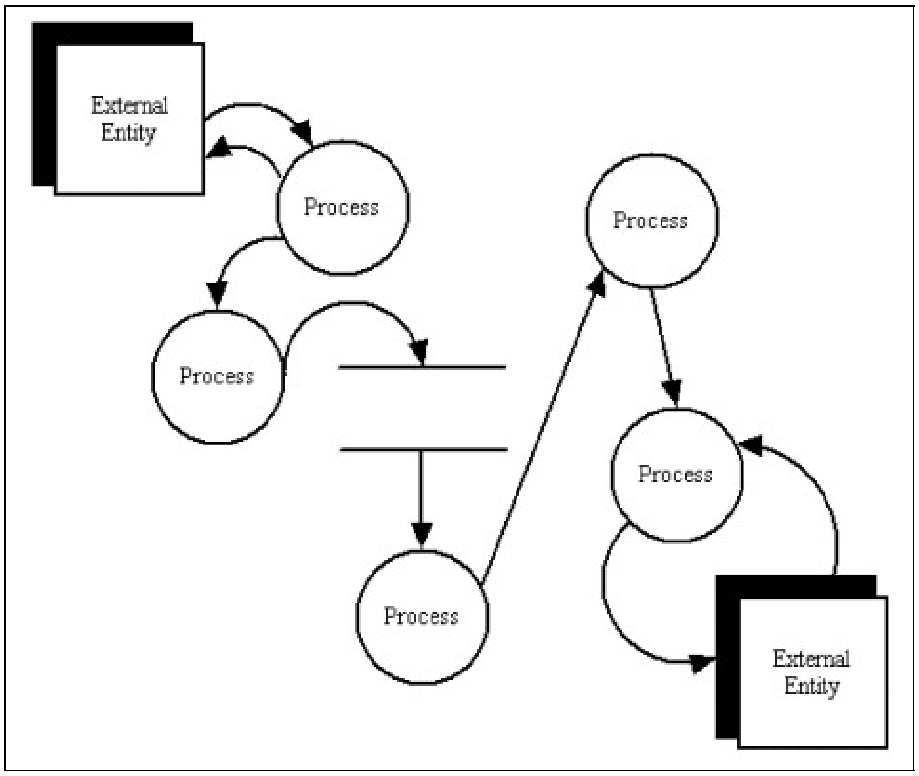
Then the maximum number of records that can be stored is

= (40 * 56664)/80
= 28332

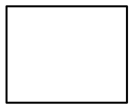
8. Data Flow Diagrams

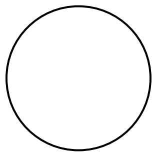
What are Data Flow Diagrams

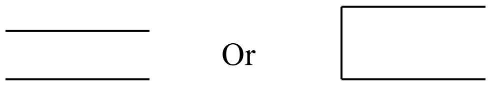
What are Data Flow Diagrams?
Data flow diagrams illustrate how data is processed by a system in terms of inputs and outputs.

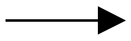


Basic Building Blocks


Represents External Entity

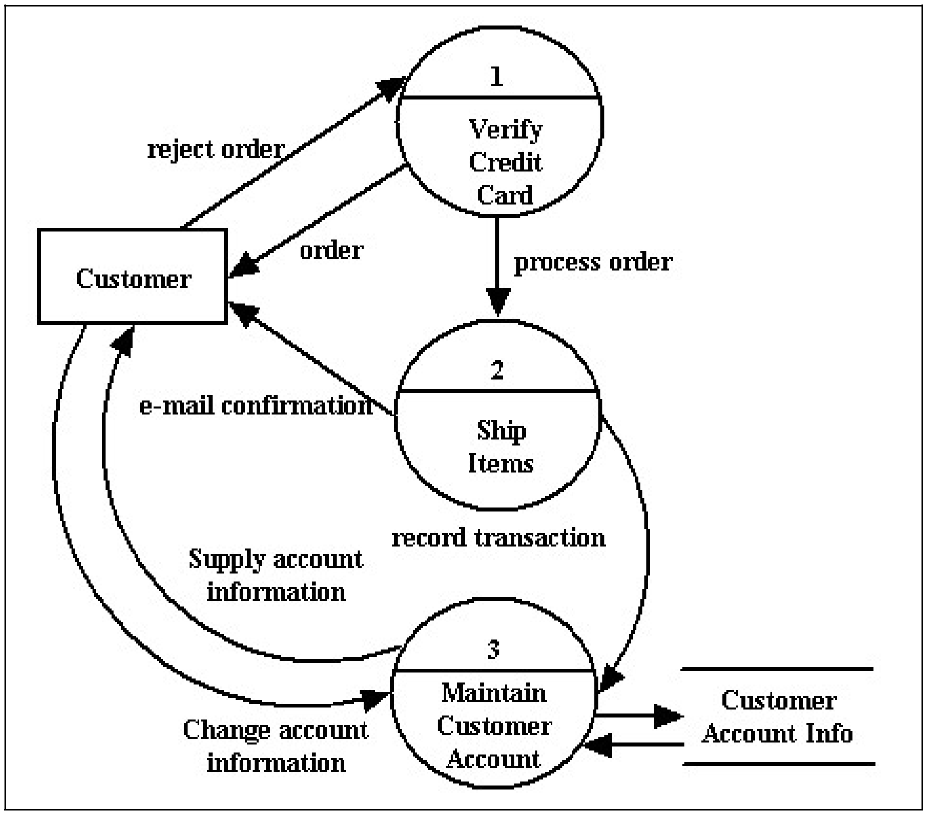

Represents Process


Or
Represents Source or Destination of data


Labeled arrow represents the data flow

DFD Levels

DFD levels
The first level DFD shows the main processes within the system. Each of these processes can be broken into further processes until you reach pseudo code.

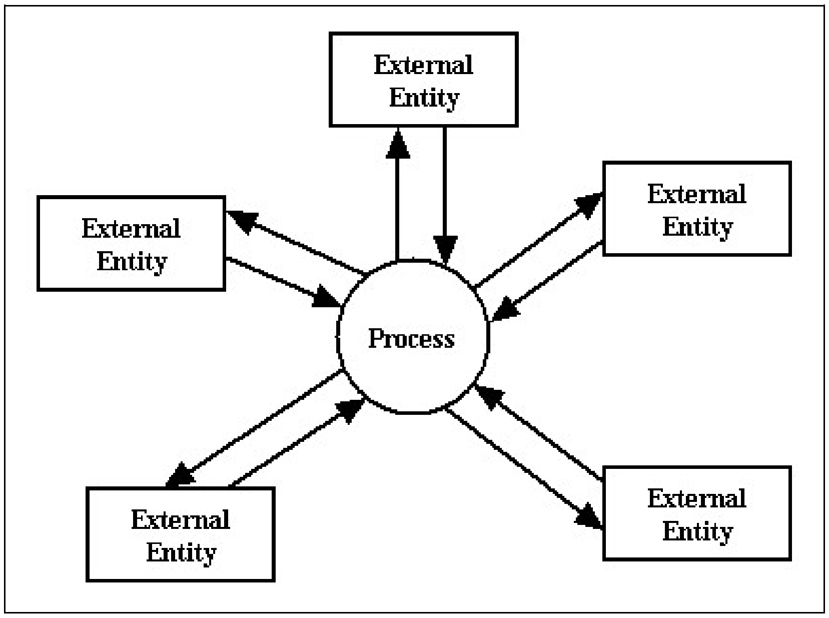


An example first-level data flow diagram

The numbering inside the circle may represent process identification number. Usually the process name is given inside the circle

Context Diagrams

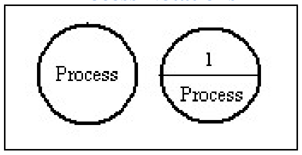
Context Diagrams
A context diagram is a top level (also known as Level 0) data flow diagram. It only contains one process node (process 0) that generalizes the function of the entire system in relationship to external entities.



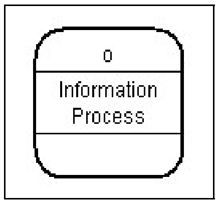
Data Flow Diagram Notations

Data Flow Diagram Notations
You can use two different types of notations on your data flow diagrams: *Yourdon & Coad or Gane & Sarson.*

Process Notations



Yourdon and Coad Process Notations

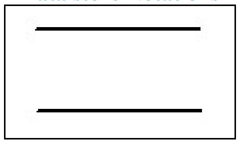


Gane and Sarson Process Notation

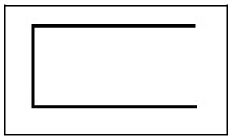
Process

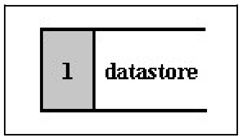
A process transforms incoming data flow into outgoing data flow.

Data store Notations



Yourdon and Coad Data store Notation



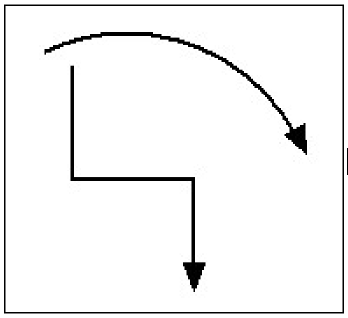


Gane and Sarson Data store Notations

Data Store

Data stores are repositories of data in the system. They are sometimes also referred to as files.

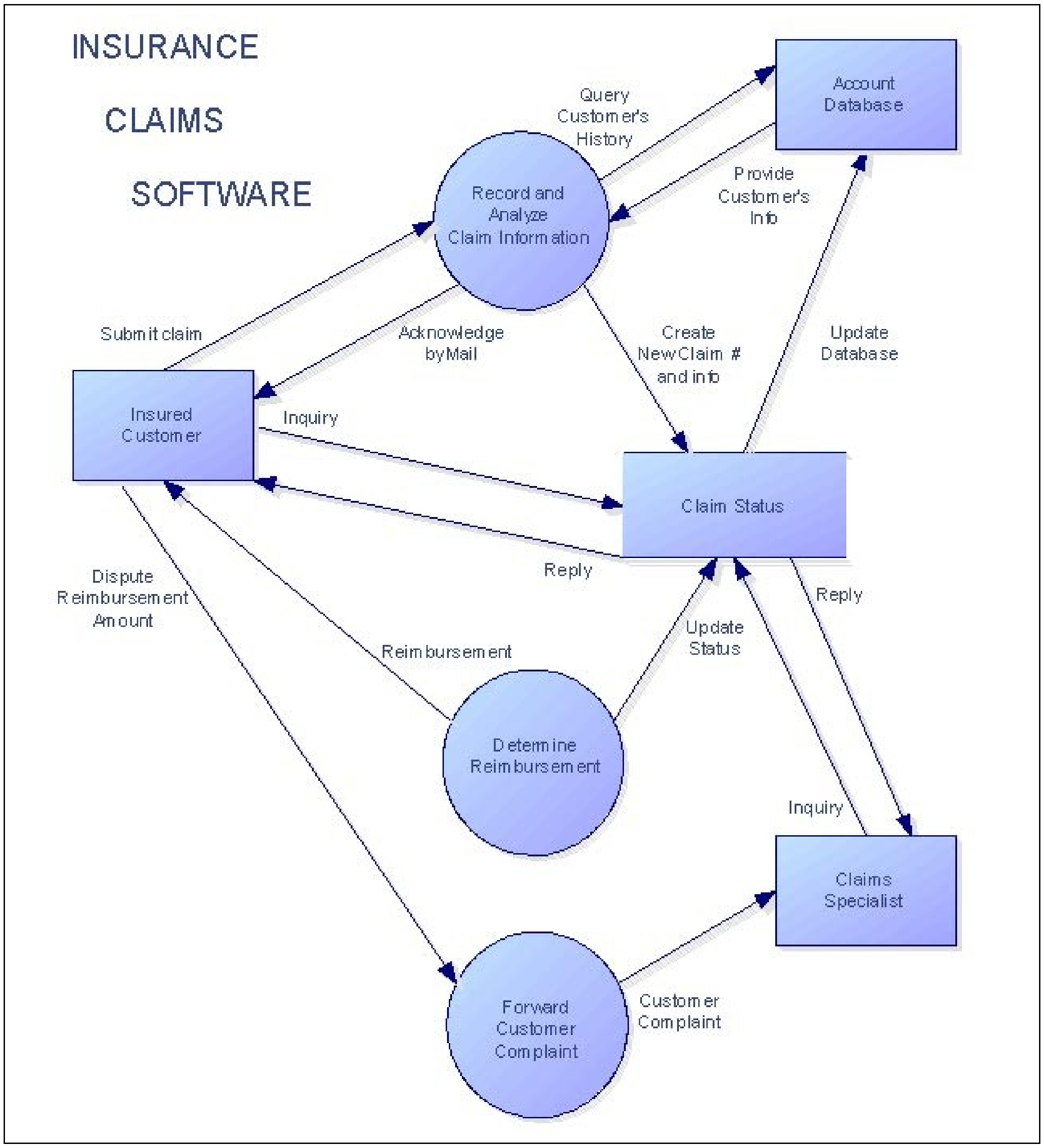
Dataflow Notations



Dataflow

Data flows are pipelines through which packets of information flow. Label the arrows with the name of the data that moves through it.

An Example


INSURANCE
CLAIMS
SOFTWARE
Account Database
Query Customer's History
Provide Customer's Info
Record and Analyze Claim Information
Submit claim
Acknowledge by Mail
Create New Claim # and info
Update Database
Insured Customer
Inquiry
Claim Status
Reply
Dispute Reimbursement Amount
Reimbursement
Update Status
Reply
Determine Reimbursement
Inquiry
Claims Specialist
Forward Customer Complaint
Customer Complaint

9. COMMON ABENDS

JCL error

File attributes don't match; For example, I have given RECFM=VB, RECLEN is same as that specified in FD section. Why do I get this error? For variable record format files you should add 4 bytes to record length in DCB.

S322

Timed out, try changing job class

S806

Load module not found. Check library specified in joblib

S913

Insufficient authority. Check if you have required access to dataset

S878

Region size is not enough. Increase the value you have specified in REGION parameter of JOB statement or in EXEC step. Or you can give REGION=0M; in this case, the system will allocate the maximum size available.

S522

Job cancelled by either user or operator.

S0C4

Storage related problem. Check your linkage section, table definition, and FD section.

S013

A file open error.

S722

The Sysout or spool is full. You program is writing too many things to Sysout. Increase job's Sysout limit by specifying 'LINES=(150,WARNING)' option in job statement and then retry. This will increase your Sysout limit to '150' *thousand* lines.

S0C7

Invalid character in COMP/COMP-3 numeric field – check all COMP/COM-3 numeric fields and arithmetic operations.

S0C7 Abend

S0C7 abend occurs when an invalid character is present in COMP/COMP-3 /numeric field and you tried to manipulate that field by some arithmetic expressions.

Let us look into the internal details.

The representation scheme that we use is EBCDIC. This is an 8 byte extension to the ASCII representation. The following is the EBCDIC representation.

Dec	**Hex**	**ASCII**		**EBCDIC**	
0	00	NUL	Null	NUL	Null
1	01	SOH	Start of Heading (CC)	SOH	Start of Heading
2	02	STX	Start of Text (CC)	STX	Start of Text
3	03	ETX	End of Text (CC)	ETX	End of Text
4	04	EOT	End of Transmission (CC)	PF	Punch Off
5	05	ENQ	Enquiry (CC)	HT	Horizontal Tab
6	06	ACK	Acknowledge (CC)	LC	Lower Case
7	07	BEL	Bell	DEL	Delete
8	08	BS	Backspace (FE)		
9	09	HT	Horizontal Tabulation (FE)		
10	0A	LF	Line Feed (FE)	SMM	Start of Manual Message
11	0B	VT	Vertical Tabulation (FE)	VT	Vertical Tab
12	0C	FF	Form Feed (FE)	FF	Form Feed
13	0D	CR	Carriage Return (FE)	CR	Carriage Return
14	0E	SO	Shift Out	SO	Shift Out
15	0F	SI	Shift In	SI	Shift In
16	10	DLE	Data Link Escape (CC)	DLE	Data Link Escape
17	11	DC1	Device Control 1	DC1	Device Control 1
18	12	DC2	Device Control 2	DC2	Device Control 2
19	13	DC3	Device Control 3	TM	Tape Mark
20	14	DC4	Device Control 4	RES	Restore
21	15	NA	Negative Acknowledge (CC)	NL	New Line

		K			
22	16	SYN	Synchronous Idle (CC)	BS	Backspace
23	17	ETB	End of Transmission Block (CC)	IL	Idle
24	18	CAN	Cancel	CAN	Cancel
25	19	EM	End of Medium	EM	End of Medium
26	1A	SUB	Substitute	CC	Cursor Control
27	1B	ESC	Escape	CU1	Customer Use 1
28	1C	FS	File Separator (IS)	IFS	Interchange File Separator
29	1D	GS	Group Separator (IS)	IGS	Interchange Group Separator
30	1E	RS	Record Separator (IS)	IRS	Interchange Record Separator
31	1F	US	Unit Separator (IS)	IUS	Interchange Unit Separator
32	20	SP	Space	DS	Digit Select
33	21	!	Exclamation Point	SOS	Start of Significance
34	22	"	Quotation Mark	FS	Field Separator
35	23	#	Number Sign, Octothorpe, "pound"		
36	24	$	Dollar Sign	BYP	Bypass
37	25	%	Percent	LF	Line Feed
38	26	&	Ampersand	ETB	End of Transmission Block
39	27	'	Apostrophe, Prime	ESC	Escape
40	28	(	Left Parenthesis		
41	29	)	Right Parenthesis		
42	2A	*	Asterisk, "star"	SM	Set Mode
43	2B	+	Plus Sign	CU2	Customer Use 2
44	2C	,	Comma		
45	2D	-	Hyphen, Minus Sign	ENQ	Enquiry

46	2E	.	Period, Decimal Point, "dot"	AC K	Acknowledge
47	2F	/	Slash, Virgule	BE L	Bell
48	30	0	0		
49	31	1	1		
50	32	2	2	SY N	Synchronous Idle
51	33	3	3		
52	34	4	4	PN	Punch On
53	35	5	5	RS	Reader Stop
54	36	6	6	UC	Upper Case
55	37	7	7	EO T	End of Transmission
56	38	8	8		
57	39	9	9		
58	3A	:	Colon		
59	3B	;	Semicolon	CU 3	Customer Use 3
60	3C	<	Less-than Sign	DC 4	Device Control 4
61	3D	=	Equal Sign	NA K	Negative Acknowledge
62	3E	>	Greater-than Sign		
63	3F	?	Question Mark	SU B	Substitute
64	40	@	At Sign	SP	Space
65	41	A	A		
66	42	B	B		
67	43	C	C		
68	44	D	D		
69	45	E	E		
70	46	F	F		
71	47	G	G		
72	48	H	H		
73	49	I	I		
74	4A	J	J	¢	Cent Sign
75	4B	K	K	.	Period, Decimal Point,

					"dot"
76	4C	L	L	<	Less-than Sign
77	4D	M	M	(	Left Parenthesis
78	4E	N	N	+	Plus Sign
79	4F	O	O	|	Logical OR
80	50	P	P	&	Ampersand
81	51	Q	Q		
82	52	R	R		
83	53	S	S		
84	54	T	T		
85	55	U	U		
86	56	V	V		
87	57	W	W		
88	58	X	X		
89	59	Y	Y		
90	5A	Z	Z	!	Exclamation Point
91	5B	[	Opening Bracket	$	Dollar Sign
92	5C	\	Reverse Slant	*	Asterisk, "star"
93	5D	]	Closing Bracket	)	Right Parenthesis
94	5E	^	Circumflex, Caret	;	Semicolon
95	5F	_	Underline, Underscore	¬	Logical NOT
96	60	`	Grave Accent	-	Hyphen, Minus Sign
97	61	a	a	/	Slash, Virgule
98	62	b	b		
99	63	c	c		
100	64	d	d		
101	65	e	e		
102	66	f	f		
103	67	g	g		
104	68	h	h		
105	69	i	i		
106	6A	j	j		
107	6B	k	k	,	Comma
108	6C	l	l	%	Percent
109	6D	m	m	_	Underline, Underscore

110	6E	n	n	>	Greater-than Sign
111	6F	o	o	?	Question Mark
112	70	p	p		
113	71	q	q		
114	72	r	r		
115	73	s	s		
116	74	t	t		
117	75	u	u		
118	76	v	v		
119	77	w	w		
120	78	x	x		
121	79	y	y		
122	7A	z	z	:	Colon
123	7B	{	Opening Brace	#	Number Sign, Octothorp, "pound"
124	7C	|	Vertical Line	@	At Sign
125	7D	}	Closing Brace	'	Apostrophe, Prime
126	7E	~	Tilde	=	Equal Sign
127	7F	DEL	Delete	"	Quotation Mark
128	80		Reserved		
129	81		Reserved	a	a
130	82		Reserved	b	b
131	83		Reserved	c	c
132	84	IND	Index (FE)	d	d
133	85	NEL	Next Line (FE)	e	e
134	86	SSA	Start of Selected Area	f	f
135	87	ESA	End of Selected Area	g	g
136	88	HTS	Horizontal Tabulation Set (FE)	h	h
137	89	HTJ	Horizontal Tabulation with Justification (FE)	i	i
138	8A	VTS	Vertical Tabulation Set (FE)		

139	8B	PL D	Partial Line Down (FE)		
140	8C	PL U	Partial Line Up (FE)		
141	8D	RI	Reverse Index (FE)		
142	8E	SS 2	Single Shift Two (1)		
143	8F	SS 3	Single Shift Three (1)		
144	90	DC S	Device Control String (2)		
145	91	PU 1	Private Use One	j	j
146	92	PU 2	Private Use Two	k	k
147	93	ST S	Set Transmit State	l	l
148	94	CC H	Cancel Character	m	m
149	95	MW	Message Waiting	n	n
150	96	SP A	Start of Protected Area	o	o
151	97	EP A	End of Protected Area	p	p
152	98		Reserved	q	q
153	99		Reserved	r	r
154	9A		Reserved		
155	9B	CS I	Control Sequence Introducer (1)		
156	9C	ST	String Terminator (2)		
157	9D	OS C	Operating System Command (2)		
158	9E	PM	Privacy Message (2)		
159	9F	AP C	Application Program Command (2)		
160	A0				
161	A1				
162	A2			s	s
163	A3			t	t
164	A4			u	u

165	A5			v	v
166	A6			w	w
167	A7			x	x
168	A8			y	y
169	A9			z	z
170	AA				
171	AB				
172	AC				
173	AD				
174	AE				
175	AF				
176	B0				
177	B1				
178	B2				
179	B3				
180	B4				
181	B5				
182	B6				
183	B7				
184	B8				
185	B9			`	Grave Accent
186	BA				
187	BB				
188	BC				
189	BD				
190	BE				
191	BF				
192	C0				
193	C1			A	A
194	C2			B	B
195	C3			C	C
196	C4			D	D
197	C5			E	E
198	C6			F	F
199	C7			G	G

200	C8			H	H
201	C9			I	I
202	CA				
203	CB				
204	CC				
205	CD				
206	CE				
207	CF				
208	D0				
209	D1			J	J
210	D2			K	K
211	D3			L	L
212	D4			M	M
213	D5			N	N
214	D6			O	O
215	D7			P	P
216	D8			Q	Q
217	D9			R	R
218	DA				
219	DB				
220	DC				
221	DD				
222	DE				
223	DF				
224	E0				
225	E1				
226	E2			S	S
227	E3			T	T
228	E4			U	U
229	E5			V	V
230	E6			W	W
231	E7			X	X
232	E8			Y	Y
233	E9			Z	Z
234	EA				

235	EB				
236	EC				
237	ED				
238	EE				
239	EF				
240	F0			0	0
241	F1			1	1
242	F2			2	2
243	F3			3	3
244	F4			4	4
245	F5			5	5
246	F6			6	6
247	F7			7	7
248	F8			8	8
249	F9			9	9
250	FA				
251	FB				
252	FC				
253	FD				
254	FE				
255	FF				

Look into the HEX column, where numeric 1 is represented as F1 where F is actually the sequence used to represent the series. For calculation, the system will take only the lower order nibbles. In this case, it is 1. So, S0C7 will occur only when an invalid value is present in the lower order nibble. Where invalid value means not simply alphabetic characters stored in numeric field. This is because, if you try to move 1A to a numeric field, the COBOL compiler will do an automatic data conversion. 1A will be represented internally as

1A

FC

11

Here the lower order nibbles contain numeric 11 so if you move 1A to a numeric filed and try to manipulate that filed, it won't cause a S0C7 , because it contains 11. Suppose you are moving 9* to a numeric field. That will be represented internally as

9*

F5

9C

Now, the lower order nibbles contain an invalid character C, if you try to manipulate this field, it would cause a S0C7 abend.

Look at the following program and its output

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.

 01  SAMPLE  PIC X(2) VALUE  '9*'.
 01  SAMPLE1 PIC 9(2) COMP-3.
 01  SAMPLE2 PIC 9(2).
 PROCEDURE DIVISION.
     MOVE SAMPLE  TO SAMPLE1
        DISPLAY SAMPLE1 ' IS SAMPLE 1:COMP-3:BEFORE COMPUTATION'
     MOVE SAMPLE1 TO SAMPLE2
        DISPLAY SAMPLE2 ' IS SAMPLE 2:9(2):MOVED FROM COMP-3'
*********THIS LINE WILL CAUSE A S0C7 ABEND************
     ADD 1 TO SAMPLE1
******************************************************
      DISPLAY SAMPLE1 ' IS SAMPLE 1:COMP-3:AFTER COMPUTATION'
      DISPLAY SAMPLE ' IS SAMPLE: PIC X'
      STOP RUN
```

This program will cause a S0C7 abend.
Its output is

```
9Ü IS SAMPLE 1:COMP-3:BEFORE COMPUTATION
9Ü IS SAMPLE 2:9(2):MOVED FROM COMP-3
```

Note that

```
DISPLAY SAMPLE1 ' IS SAMPLE 1:COMP-3:AFTER COMPUTATION'      and
DISPLAY SAMPLE ' IS SAMPLE: PIC X'
```

Are not executed because just before that the program abended.

This is a possible cause of S0C7 error. For more information on the causes of this error, see the description of S0C7 in QW.

--END--