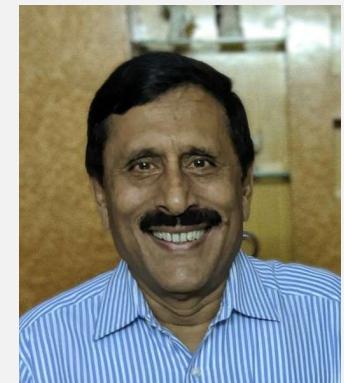


Mainframe Training

Ramana Reddy



Mainframe Concepts

Mainframe Overview

Mainframe System

- High processing speed and throughput
 - 100 to 750 MIPS - Million Instructions Per Second
- Capacity to store and handle huge amounts of Data consistently
 - Typically 50 GB
- Ability to run multiple applications simultaneously
- Support large number of users
 - 100 or more
- High Availability
 - 99.999% Uptime, means 10 minutes down-time in 365 days.
- 31-bit addressing capability
 - Means up to 2 GB of data being accessed in the main memory.

Operating System

MVS (Multiple Virtual Storage)

- MVS is a operating system in Mainframe environment
- The *Virtual Storage* in MVS refers to the use of virtual memory in the operating system. Virtual storage or memory allows a program to have access to the maximum amount of memory in a system even though this memory is actually being shared among more than one application program.
- The operating system translates the program's *virtual* address into the real physical memory address where the data is actually located.
- The *Multiple* in MVS indicates that a separate virtual memory is maintained for each of multiple task partitions.

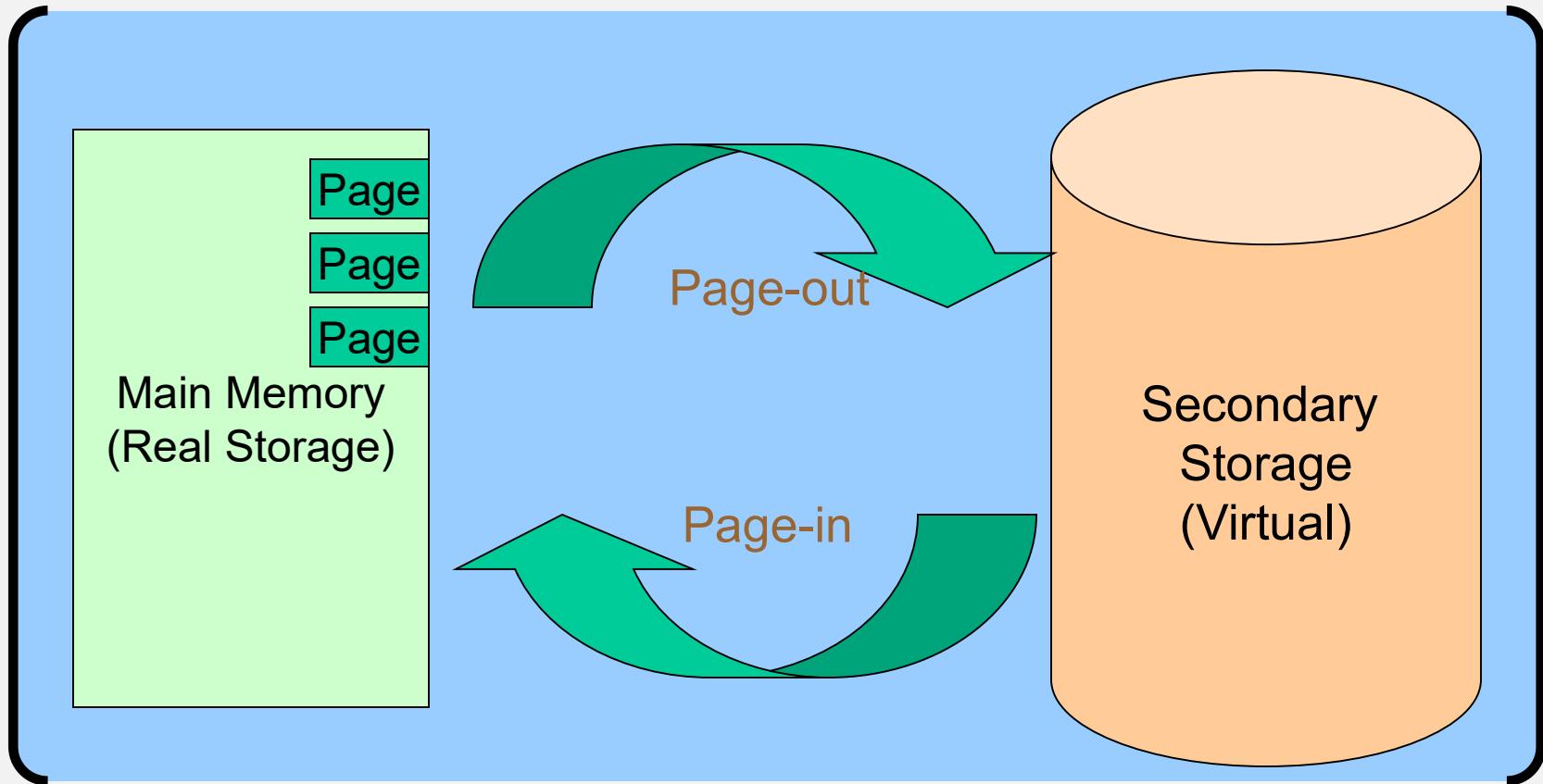
Virtual Storage & Paging

- To manage central storage effectively, operating system writes inactive pages from central storage to DASD
- DASD used by the operating system to contain inactive pages is called auxiliary storage
- Auxiliary storage is divided into separate areas called SLOTS, each the same size as a virtual storage page (4K bytes) and each accessible by a unique address
- When a reference is made to instructions or data that are not currently in central storage, a PAGE FAULT occurs to bring the appropriate page into central storage from auxiliary storage.

Swapping

- MVS periodically transfers entire address space in and out of virtual storage
- Critical pages of address space are written to swap data set
- Swapping is similar to paging only at higher level
- Paging does not occur for address spaces that are swapped out

Swapping and Paging



Secondary Storage

- Direct access storage devices (DASD) is a secondary storage device.
 - This is similar to the hard disk in PC
- DASD Volumes
 - VOL1 label (Similar to Disk drives in PC)
 - ✓ Identifies the DASD device
 - ✓ Stored in 3rd record on track 0 Cylinder 0
 - ✓ Contains the address of VTOC
- VTOC: Volume Table of Contents (similar to FAT in PC)
 - Contains the Data Set Control Blocks (DSCB)
 - Contains Dataset (File) name and its location

Data Management

- Two different data management environments coexist under MVS - *VSAM* & *non-VSAM*
- Data stored in the form of *Data sets*
- Data set is a collection of related data that is managed as a unit
- Within data set data is organized into smaller units called *Records*
- MVS identifies data sets with special records called labels

Sub Systems

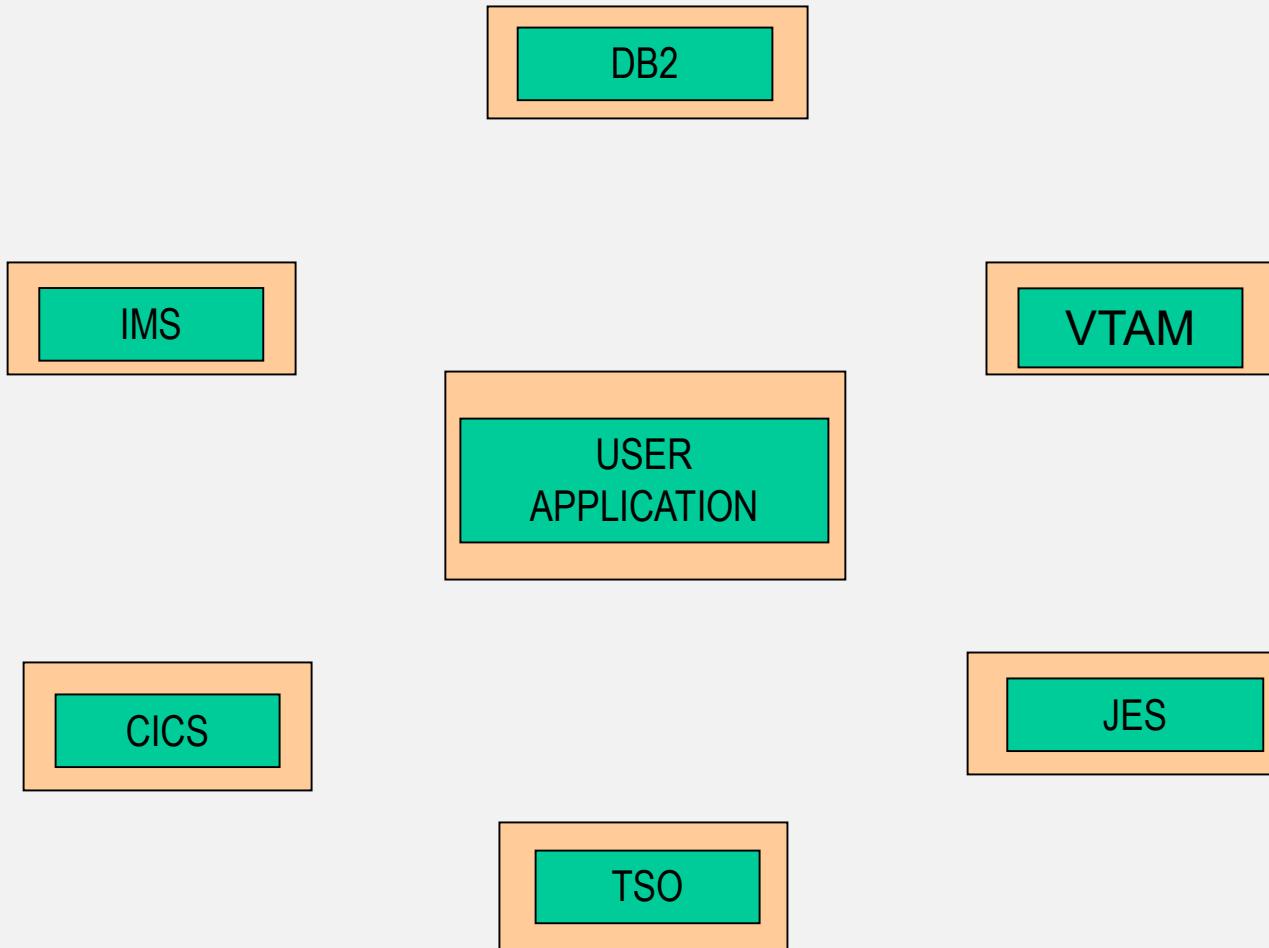
Software products to perform specialized functions

- Don't form part of Operating System OS/390
- Run in a separate address space like any other application program
- Use the operating system to gain access to external resources like DASD, Printer
- All subsystems run as separate jobs which is handled by the JES subsystem
- JES is a special subsystem under MVS. Without JES no subsystem can run

Subsystems interaction within MVS

- All subsystems run within the 2 GB area
- Within the 2GB Systems area each subsystem is allocated a separate area

Sub Systems



Sub Systems

- Commonly Available Subsystems
 - TSO (Time Sharing Option)
 - JES (Job Scheduler)
 - CICS (Transaction Processor)
 - IMS (Transaction Processor with Hierarchical Database)
 - DB2 (Relational Database)
 - SMS (Storage Management Subsystem)

Sub Systems

TSO

- Time Sharing Option
- Allows users to interactively share computer time and resources
- Allows line mode command entry
- ISPF is a software that runs as a part of TSO and provides dialog management service. Dialogs allow a TSO user to issue commands directly or indirectly from panels (screens).

Sub Systems

JES

- Job Entry Subsystem (JES) is a subsystem of the mainframe operating systems that manages *jobs* (units of work) that the system does.
- Each job is described to the operating system by system administrators or other users in job control language
- The operating system then sends the job to the JES program. The JES program receives the job, performs the job based on priority, and then purges the job from the system.
- There are two versions, JES2 and JES3.
 - JES3 allows central control of the processing of jobs using a common work queue.
- Both OS/390 and MVS provide an interactive menu for initiating and managing jobs.

Sub Systems

Batch Processing

- Job
 - A set of tasks to be performed in a pre-determined sequence to satisfy some objective
 - Turn around time is not fixed
 - All the tasks to be performed are represented as JCL statements
 - JCL statements identify the programs that must be executed and the resources required
 - After the JCL statements are built the JCL is submitted to the system

Sub Systems

- Job Execution (cont.)
 1. The JCL will be converted into machine understandable format and moved to Execution Queue
 2. From the Execution Queue the job will taken for execution based on priority
 3. All required data will be taken from the SPOOL
 4. After the job is completed the job will be moved to the output queue from where it will be printed

CICS (Customer Information Control System)

- CICS is an online transaction processing System
- IBM Product of the 70's (Intro in 1969)
- CICS acts like an Operating System
 - Manages its own storage
 - Provides its own file and D/B management
 - Has its own TASK manager
 - Runs in its own ADDRESS SPACE

Sub Systems

IMS (Information Management System)

- IMS Stands for Information Management System.
- IMS consists of two parts
 - DL/I – Hierarchical Database
 - DC – Data Communications
- IMS is used as a Database manager and Transaction Manager

DB2 (DataBase2)

- DB2 Stands for DataBase2
- DB2 is the Relational DBMS on MVS
- Does not have a separate Transaction Manager unlike CICS/IMS
- Works with CICS/IMS in On-line environment and with application programs in a batch mode

File System

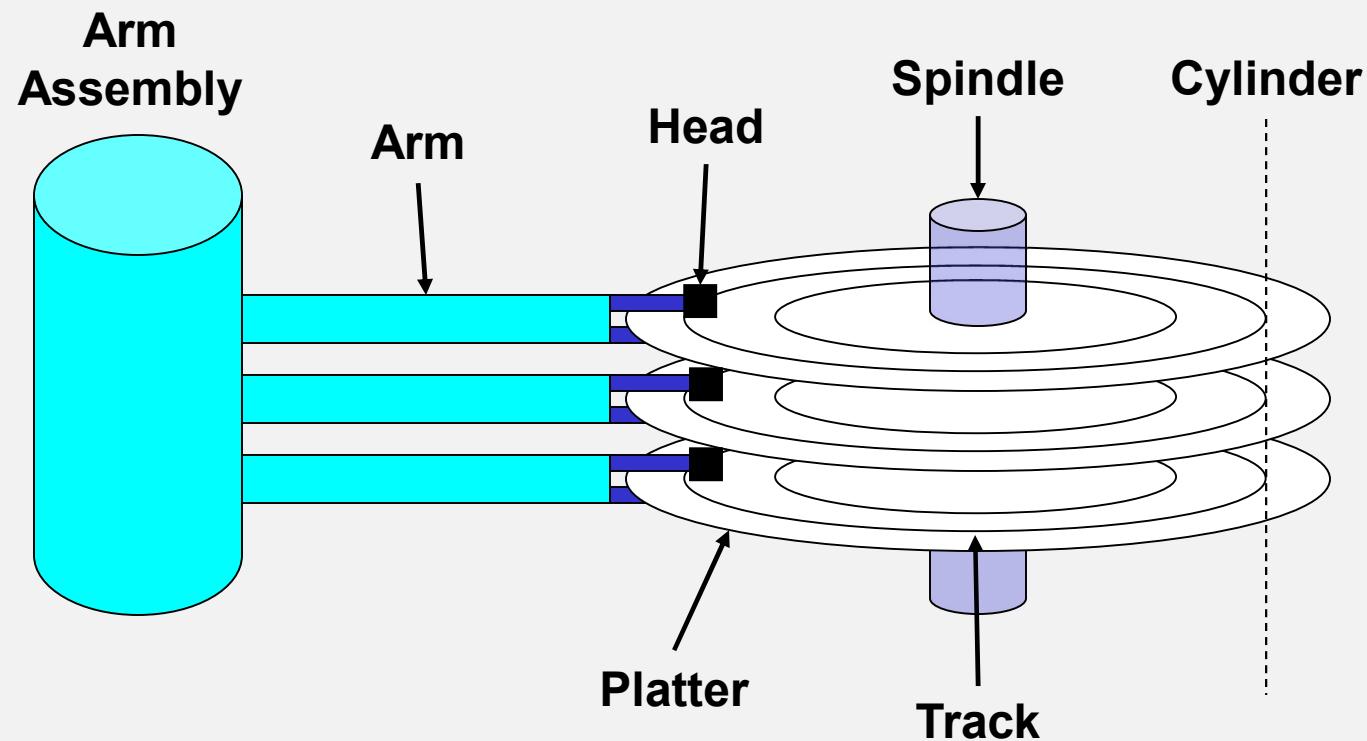
DASD

- Direct access storage devices (DASD) is a secondary storage. This is similar to hard disk in PC

Data Set

- It is a collection of records
- Terminology used in Mainframe for “File”
 - Example
 - ✓ Source Program
 - ✓ Data Record

DASD



File System

Data Set Type

- PS (Physical Sequential)
 - Simple file
- PDS (Partitioned Dataset)
 - This data set consist members.
 - This can be compared to a folder with list of files (No Sub folders)
- VSAM (Virtual Storage Access Method)
 - These data sets are used to store business data.
 - These are indexed for faster access.

File System

Dataset Characteristics

Data Set Organization (DSORG) Sequential Partitioned VSAM	Record Format (RECFM) <u>Fixed</u> <u>Variable</u> <u>Undefined</u> <u>Fixed Blocked</u> <u>Variable Blocked</u>	
Record Length (LRECL)	Block Size (BLKSIZE)	Directory Blocks (DIR)

File System

Dataset Characteristics (cont.)

- Record Format
 - Fixed
 - ✓ This means that all the records stored in these datasets have same length (RECFM=F).
 - Variable
 - ✓ These datasets can store records that are different in length (RECFM=V).
 - Blocked – Blocking is the process of grouping records into blocks before they are written on a volume.
 - ✓ A block consists of one or more logical records. Each block is written between consecutive inter-block gaps (IBG).
 - Blocking can be done for both Fixed Length records (RECFM=FB) and Variable length records (RECFM=VB).

File System

Dataset Characteristics (cont.)

- Record Format (cont.)



IBG =Inter Block Gap
m =Block Size
n =Record Length

File System

Data set characteristic (cont.)

- Record Length
 - This is the record length of the data set
 - For a Variable Record file, this is the *average length* of the record.
 - If the record format for a PDS is fixed, all the members in the PDS have the same record length.
- Block Size
 - The Block Size decides the number of records taken into buffer for each I-O operation.
 - For a Fixed length dataset, this is always specified as an integral multiple of the record size

File System

Dataset Space allocation

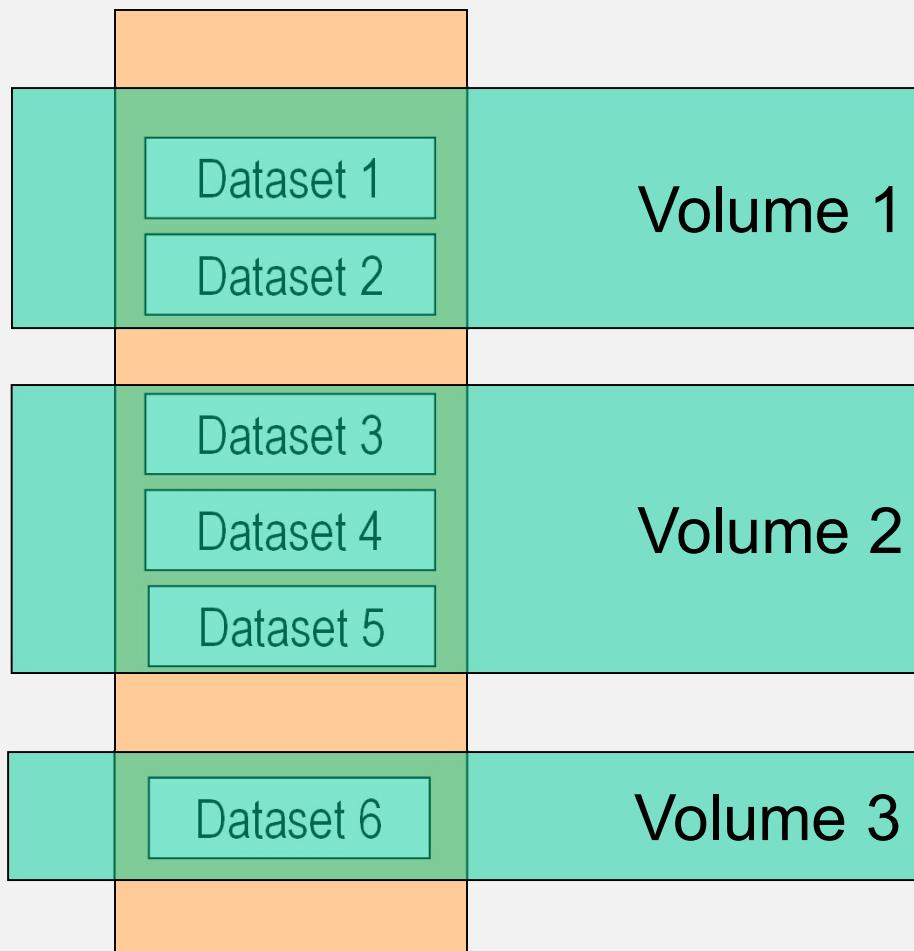
- To start with primary extent is allocated
- When the primary extent is full, one secondary extent is allocated
- Secondary extent is allocated repeatedly based on the space requirement
- Max allocation – 16 extents
- Directory space is required for PDS
- DASD Space Allocation Methods
 - ✓ Block Allocation
 - ✓ Record length Allocation
 - ✓ Track Allocation
 - ✓ Cylinder Allocation

Catalogs

- Catalogs help to locate a file without vol-ser
- Master catalog
 - MVS system has one master catalog
 - Contains
 - entries to identify system data sets
 - entries about user catalogs
- user catalogs
 - MVS system can have multiple user catalogs
 - Contain
 - entries to identify user data sets

File System

Catalog



Catalog Maintains
Dataset names
Irrespective of where
Dataset resides

File System

Data Set Naming Conventions

- Cataloged Data Set should have unique name across installation
- Data set has “Segmented Naming Convention”
 - Each segment represent a level of qualification
 - Each qualifier is can have MAX 8 characters
 - Example:
CTS1T. MAINFRAM. PDS. DATASET
 - The above data set has 4 segment name
- Maximum length of a DS name is 44 characters

TSO / ISPF

TSO Introduction

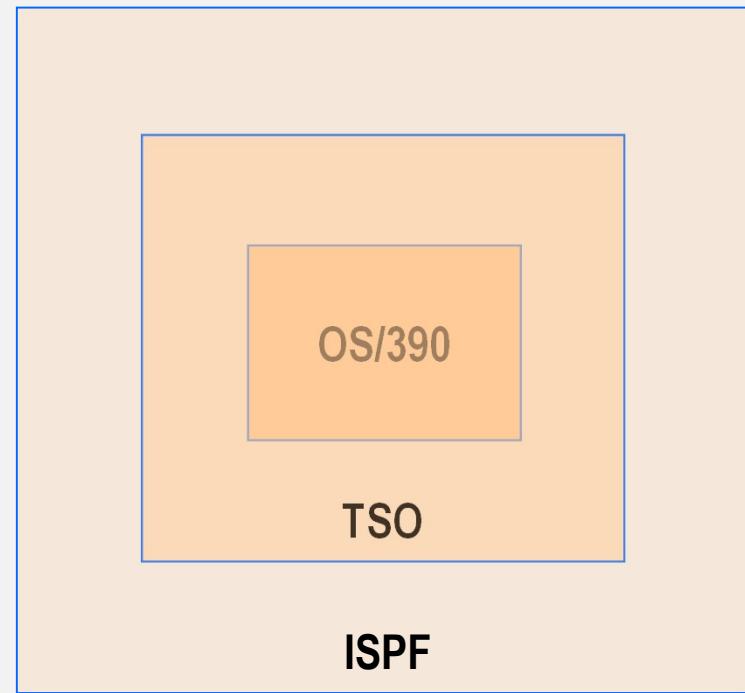
Introduction

Time Sharing Option, generally known as TSO, is a subsystem that runs on the MVS operating system on a S390 machine. This subsystem allows users to interactively work with the system.

TSO Introduction

TSO Environment

- The Interactive System Productivity Facility (ISPF) and its Program Development Facility (ISPF/PDF) work together with TSO/E to provide panels with which users can interact.
- ISPF provides the management service that displays panels and enables a user to navigate through the panels.
- ISPF/PDF is a dialog of ISPF that helps maintain libraries of information in TSO/E and allows a user to manage the library through facilities such as browse, edit, and utilities.



TSO Introduction

Advantages of using TSO/ISPF

- It supports multiple concurrent users to use the System
- ISPF has many menus and options which can be easily used by not so very experienced user.
- TSO is like a SHELL in UNIX environment where as ISPF Is like a Windows environment which is more user friendly.
- Same activities that can be performed in ISPF can also be performed by TSO but the user will have to issue series of commands one by one and have to remember all it's parameters and sub parameters.

TSO Introduction

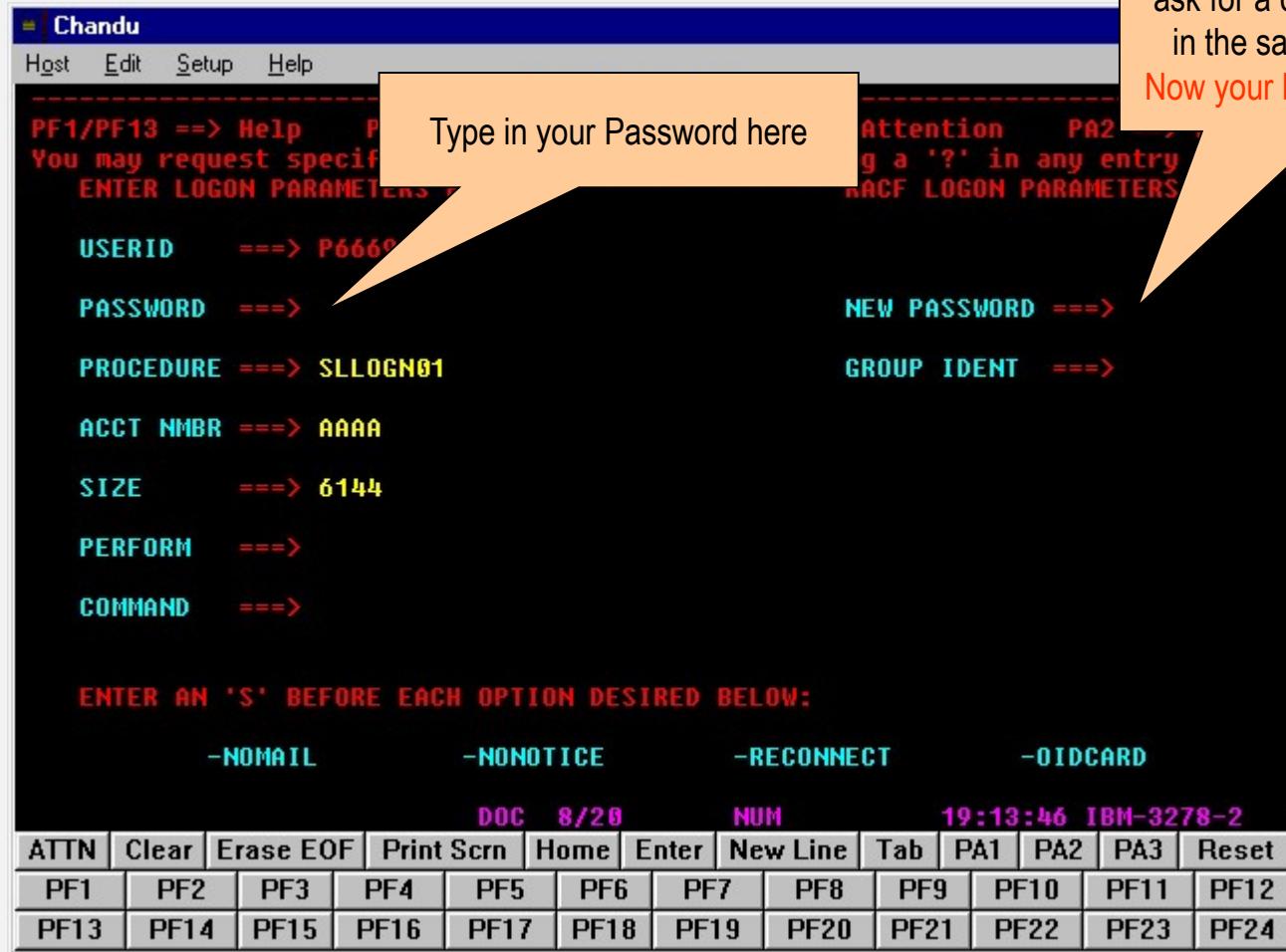
Background Vs Foreground Processing

- During Foreground Processing the Session is locked and no further processing can be done by the user
- During the Background processing the user can trigger the Processing and he can perform other tasks till the Process is complete. The user is notified when the background job is completed
- When background jobs are executed, the user can logoff from the system and his jobs will continue execution in the background

TSO Introduction

Logging on Procedures

- The Authentication screen has the following details.



If You want to change your password then Type in your New password here. The system will ask for a confirmation then type in the same password again.
Now your Password is changed!

ISPF

- Provides an On-line environment to interact with Operating System.
- Provides Menu driven interface as opposed to TSO which provides line mode only.
- Provides facilities for Editing and browsing datasets
- ISPF uses panels to display and receive information and allows customization of the ISPF environment.
- Provides Program Function keys to avoid manual typing of commands.
- Provides for easy management of datasets through the Dataset menu.

ISPF Panels

Action Bars

- Perform actions without ending present action

```
File Edit Confirm Menu Utilities Reserved Test Help
-----
EDIT      CHERYL.CNTL(IEFB| _ 1. Library | s 0000
Command ==>           | 2. Data Set | oll ==
Pull Down
***** *****
00001 //TSOCLW1 JOB .... | *.*. Data Set List |
00002 //STEP1 EXEC PGM=IE| 5. Reset Statistics |
00003 //                   | .... |
-----|
```

- Point and Shoot fields
 - Placing the cursor on a *point and shoot* field, and pressing the ENTER key performs the action specified by the field.
 - If you've also placed a command in the command line, that command will be performed before the point and shoot field action.

ISPF Panel

- Action Bar choices
 - Actions on pull down menus: Place the cursor anywhere on the line of your selection and press ENTER.
 - Selection titles on menu panels. Menu panels that have action bars are set up with title fields that are point and shoot fields.
 - ✓ **Example:** placing the cursor on the word 'Edit' on the Primary Option menu and pressing Enter takes you into Option 2, Edit.
- Member list items.
 - Place the cursor in the area in front of a member list item and press ENTER to select the item.
 - ✓ **Example:** if you are in edit and have a member list displayed, you can tab to the desired member and press Enter to select it without having to type an 'S' first.

ISPF Panels

Selection Fields (single period, single underscore, underscored field)

- A single period (.)
 - Member lists that use a single period in the selection field recognize only a single selection.
 - **Example:** Within the Edit function on this screen, You can select only one member to edit. The below Figure shows an example of this kind of panel.

```
ISREPO01 TLG1063.ISPF.ISPPROF                                     Row 00001 of
00019
Command ===>
      Scroll ===> PAGE
      Name      Prompt          Size     Created        Changed       ID
. CSQOPROF
. CTSTBL
. DGIPROF
```

ISPF Panels

- A single underscore (_)
 - Selection fields marked by a single underscore prompt you to use a slash (/) to select the choice. You may use any non-blank character.
 - **Example:** the “Panel display CUA mode” field on the ISPF Settings panel has a single underscore for the selection field as shown in Figure below.

ISPISMMN	ISPF Settings	0 Members processed
Command ==>		more: +
Options		Print Graphics
Enter "/" to select option		Family printer type 2
_ Command line at bottom		Device name
/ Panel display CUA mode		Aspect ratio . . . 0
/ Long message in pop-up		
_ Tab to action bar choices		
_ Tab to point-and-shoot fields		
/ Restore TEST/TRACE options		General
_ Session Manager mode		Input field pad . . N
/ Jump from leader dots		Command delimiter . ;
_ Edit PRINTDS Command		
_ Always show split line		
_ Enable EURO sign		

ISPF Panels

- An underscored field (_____)
 - Member lists or text fields that use underscores in the selection field recognize multiple selections.
 - **Example:** from the Display Data Set List Option panel, you may select multiple members for print, rename, delete, edit, browse, or view processing.

ISRUDSM DSLIST TLG1063.ISPF.ISPPROF					Row 00001 of 00019	
Command ===>					Scroll ===> PAGE	
Name	Prompt	Size	Created	Changed	ID	
CSQOPROF						
CTSTBL						
DGIPROF						
DSQEPROF						
ICEPPROF						

ISPF Panels

Function Keys

- Function Keys are short cut keys similar to the Keys used in Windows.
- These keys are used to trigger certain actions without typing in their respective commands or to trigger certain series of commands.

```
PF Key Definitions and Labels

Number of PF Keys . . . 12          More:      +
Enter "/" to select . .
                                         Terminal type . . 3278
                                         (Enable EURO sign)

PF1 . . . HELP
PF2 . . . SPLIT
PF3 . . . END
PF4 . . . RETURN
PF5 . . . RFIND
PF6 . . . RCHANGE
PF7 . . . UP
PF8 . . . DOWN
PF9 . . . SWAP
PF10 . . LEFT
PF11 . . RIGHT
PF12 . . RETRIEVE

PF1 label . . .           PF2 label . . .           PF3 label . .
PF4 label . . .           PF5 label . . .           PF6 label . .
PF7 label . . .           PF8 label . . .           PF9 label . .

Command ===>
F1=Help      F2=Split      F3=Exit      F7=Backward  F8=Forward  F9=Swap
F12=Cancel
```

ISPF Panels

Splitting Screens

- Splitting screens helps the user to view two different screens and view both at the same time. The SPLIT Command is used to do the same.

```
----- ISPF/PDF PRIMARY OPTION MENU -----  
OPTION ==>  
  
                                USERID - U105762  
0 ISPF PARMS - Specify terminal and user parameters DATE - 04/02/11  
1 BROWSE - Display source data or output listings TIME - 09:22  
2 EDIT - Create or change source data  
3 UTILITIES - Perform utility functions  
F1=HELP      F2=SPLIT      F3=END      F4=RETURN      F5=RFIND      F6=RCHANGE  
F7=UP        F8=DOWN       F9=SWAP      F10=LEFT       F11=RIGHT     F12=RETRIEVE  
  
-----  
Menu RefList RefMode Utilities Help  
      Data Set List Utility  
      More: +  
blank Display data set list          P Print data set list  
V Display VTOC information         PV Print VTOC information  
  
Enter one or both of the parameters below:  
Dsname Level . . . TRG*024 TEST.JCL  
Option ==>  
F1=Help   F2=Split   F3=Exit   F7=Backward F8=Forward F9=Swap F10=Actions F12=Cancel
```

ISPF Panels

- Only Two screens can be viewed at the same time.
- Switching to other screens when working from one screen is possible using SWAP command.
- The horizontal position of the split can be decided by the user
 - The Current horizontal position of the Cursor during split decides the position of the New Screen.
- Activating either of the screens can be done by Placing the cursor in the screen and pressing ENTER Key.
- There can be 8 screens simultaneously open(Use the START Command).
The swapping between screens can be done by pressing the Screen number and F9 in the command Prompt.
- SWAP LIST command can be used to list all the currently opened screens.

ISPF Panels

- The following screen is thrown when the SWAP LIST Command is executed.

```
Menu RefList RefMode Utilities Help
s Eoooooooooooooo ISPF Task List ooooooooooooooN oooooooooooooo
e          Active ISPF Logical Sessions
O e
e . Start a new screen
e . Start a new application
e Application Name
e
E e
e ID  Name      Panelid  Applid   type
e . 1           ISR@PRIM  ISP      3270
e . 2  DSLIST   ISRUDLP   ISP      3270
D e . 3  DSLIST   ISRUDLP   ISP      3270
e . 4- DSLIST   ISRUDLP   ISP      3270
e . 5* DSLIST   ISRUDLP   ISP      3270
e .
e .
e .
W e
e
DoooooooooooooooREXX exec, or
"=" to execute the previous command.
```

This is the name of the Screen you have opened in **Session 3**. These names are assigned by the system hence the same name is assigned for the similar Panel. This can be confusing to the User when he wants to have List of Source Dataset in one Panel and List of JCL in another Panel.

ISPF Panels

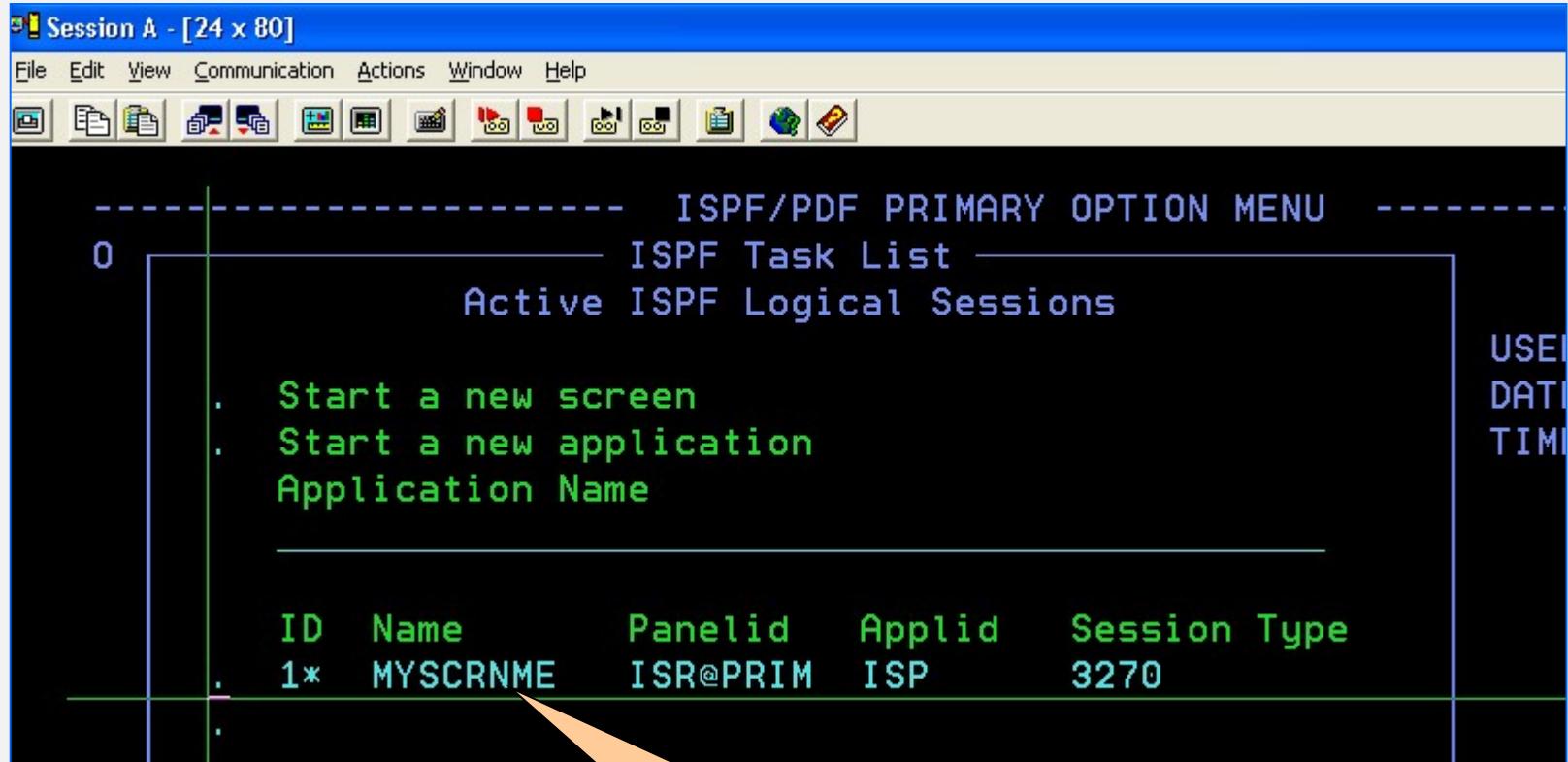
- SCRNAME command is used to assign a Screen a name which User prefers instead of a number.

```
----- ISPF/PDF PRIMARY OPTION MENU -----  
O Eooooooooooooo ISPF Task List ooooooooooooooN  
e          Active ISPF Logical Sessions      e  
e  
e . Start a new screen                      e  
e . Start a new application                  e  
e Application Name                          e  
e  
e  
e  
e ID   Name       Panelid     Applid    Session Type  e  
e . 1*           ISR@PRIM    ISP        3270  
e .  
  
----- ISPF/PDF P  
OPTION ==> SCRNAME MYSCRNME  
  
0 ISPF PARMS - Specify termina  
1 BROWSE      - Display source
```

No Screen Name was associated with this Screen.

This will assign the MYSCRNME to the Screen.

ISPF Panels

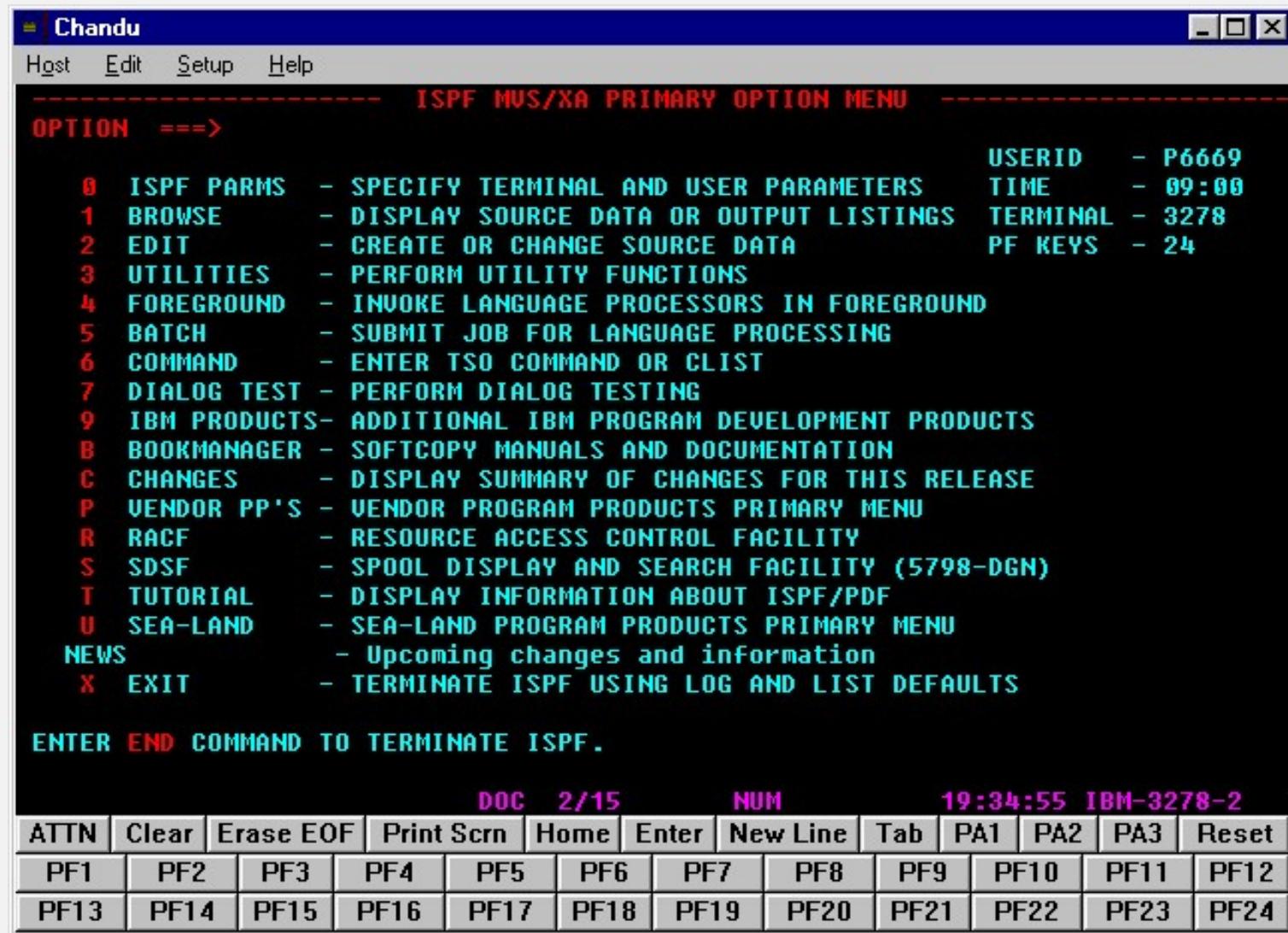


Now the Screen was assigned
"MYSCRNME" name.

Primary Option Menu

- This is the First Screen of the ISPF the user looks into after he/she logs on.
- This screen contains various options to be used for selecting each of the Program.
- The installation Specific Programs also find an entry in this menu.
- Many related programs are grouped together and they find a single entry in this menu.
- Upon selection of these options it may throw up different screens with more options to choose the individual program.

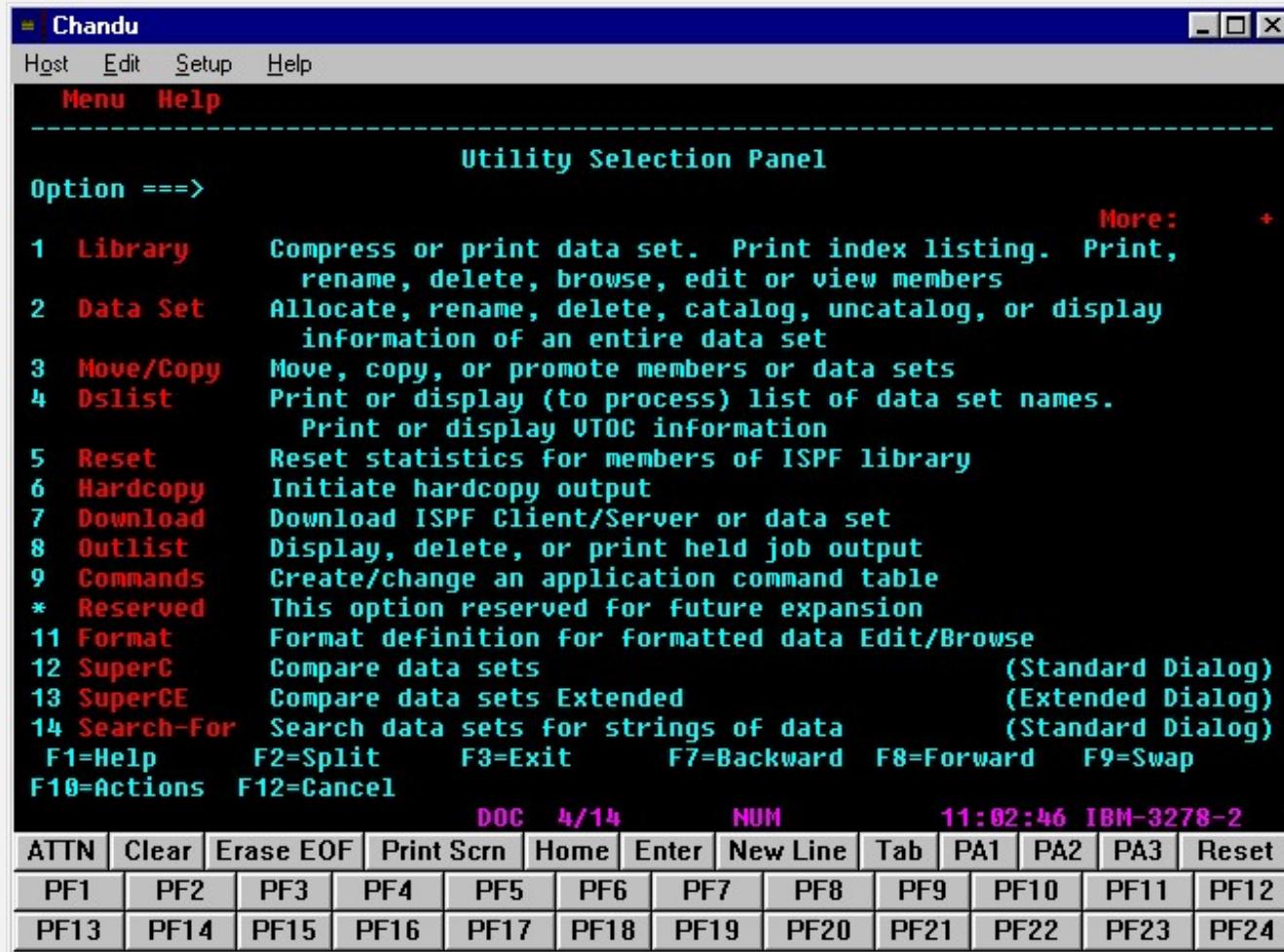
Primary Option Menu



Primary Option Menu - Dataset Utility Functions

The Utilities Option - Option 3

Various utilities that can be performed on a dataset are grouped in this panel.



Primary Option Menu - Dataset Utility Functions

- 1. The Utilities Option - Option 3**
- 2. Library Utility - Option 3.1**
- 3. Data Set Utility - Option 3.2**
- 4. Move/Copy Utility - 3.3**
- 5. DSLIST - Option 3.4**
- 6. Other Utilities**

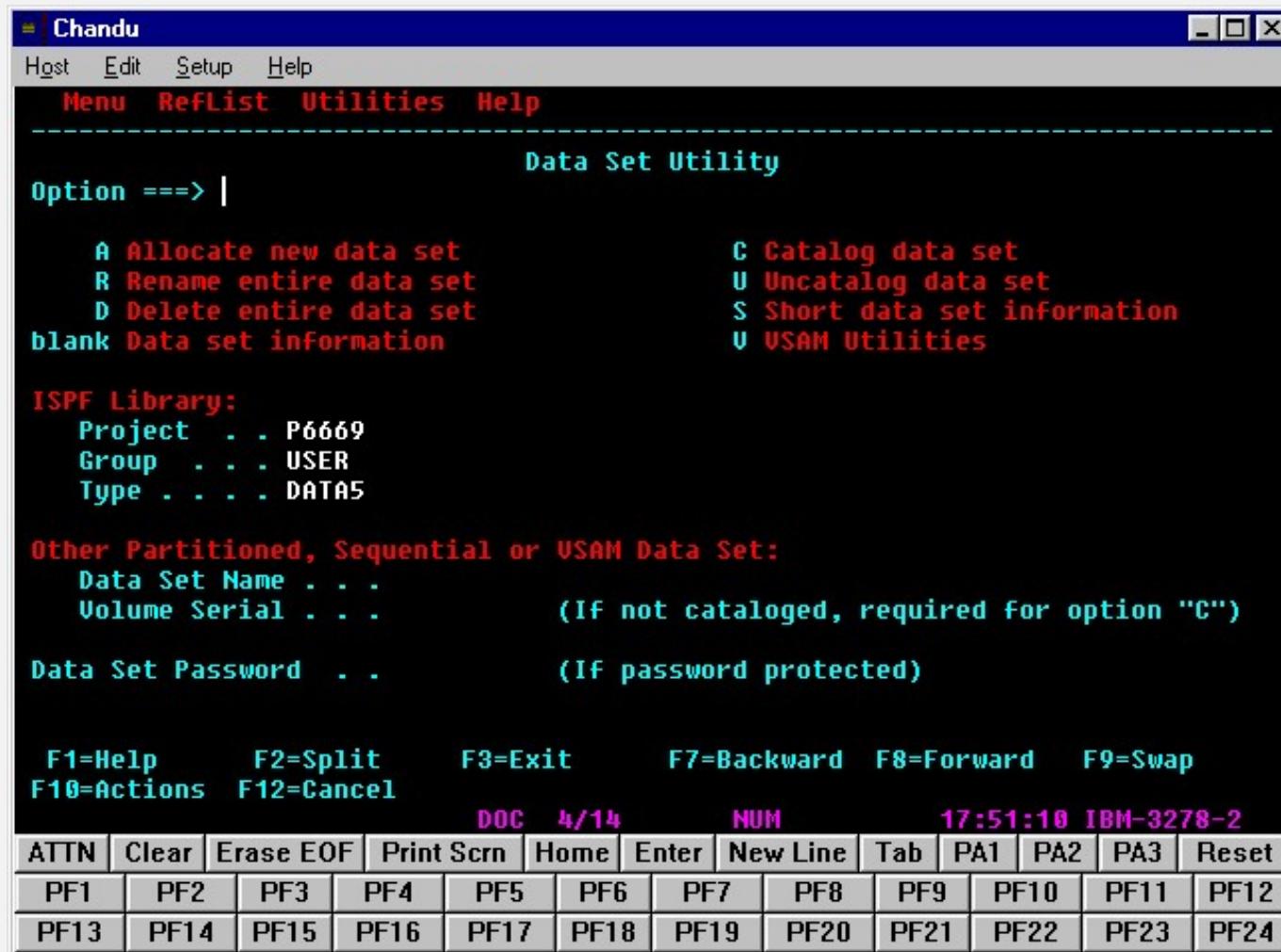
OPTION - 3.2

OPTION - 3.2

- This panel is used to perform the following activities on a dataset
 - Allocate new data set
 - Rename entire data set
 - Delete entire data set
 - Catalog data set
 - Uncatalog data set
 - Short data set information
 - VSAM Utilities
 - View Dataset Information

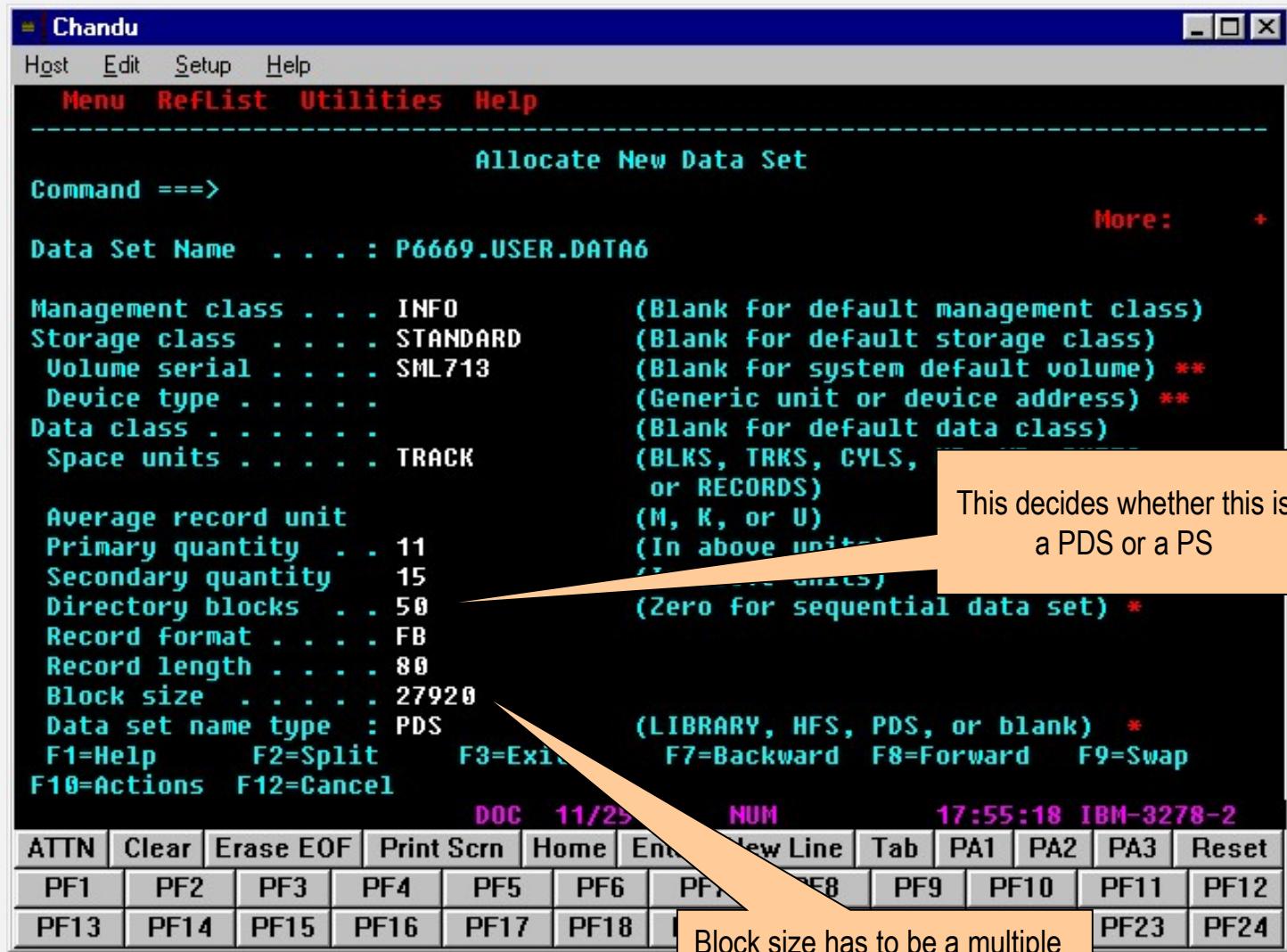
OPTION - 3.2

This is the panel used to allocate any Dataset (Either PS or a PDS)



OPTION - 3.2

This is the panel where allocation parameters are entered



OPTION - 3.3

- This is the Panel Used to Move or Copy a dataset.
 - The option C is used to Copy the dataset
 - The option M is used to Move the dataset
 - The CP/MP is used to Copy or Move along with Print

OPTION - 3.4

- This Panel is used to list the datasets. (Most widely used screen in the ISPF Panel).
 - The dataset matching the search criteria will be listed in this Panel.
 - The Search string can handle Wild search such as use of “*” and “%”
 - ✓ “*” is used for group of any Characters of any Length
 - ✓ “%” is used for matching any one character

OPTION - 3.4

```
Menu RefList RefMode Utilities Help  
ssssssssssssssssssssssssssssssssssssssssssss  
Data Set List Utility
```

```
Option ===>
```

```
blank Display data set list  
V Display VTOC information
```

```
Enter one or both of the parameters below:
```

```
Dsname Level . . . TCHN046.TEST.JCL  
Volume serial . . .
```

```
Data set list options
```

Initial View . . . 1	1. Volume	Enter "/" to select option
	2. Space	/ Confirm Data Set Delete
	3. Attrib	/ Confirm Member Delete
	4. Total	/ Include Additional Qualifiers

```
When the data set list is displayed, enter either:
```

```
"/" on the data set list command field for the command prompt pop-up,  
an ISPF line command, the name of a TSO command, CLIST, or REXX exec, or  
"=" to execute the previous command.
```

The option is used to decide which attribute of the dataset to be displayed when it is listed.

OPTION - 3.4

- When the Initial View Option is chosen as 1 the following is displayed for the below search criteria.

- Enter one or both of the parameters below:

Dsname Level . . . U105762.**.TEST*
Volume serial . . .

Data set list options

Initial View . . . 1	1. Volume	Enter "/" to select option
	2. Space	/ Confirm Data Set Delete
	3. Attrib	/ Confirm Member Delete
	4. Total	/ Include Additional Qualifiers

- DSLIST - Data Sets Matching U105762.**.TEST* Row 1 of 4

Command ===> Scroll ===> PAGE

Command - Enter "/" to select action	Message	Volume
U105762.A.B.TESTA		PUB005
U105762.A.B.TESTAB		PUB003
U105762.TEST.NEW.PDS		PUB005
U105762.TEST.PDS		PUB003

***** End of Data Set list *****

OPTION - 3.4

- When the Initial View Option is chosen as 2 the following is displayed for the below search criteria.

- Enter one or both of the parameters below:

Dsname Level . . . U105762.**.TEST*
Volume serial . . .

Data set list options

Initial View . . . 2	1. Volume	Enter "/" to select option
	2. Space	/ Confirm Data Set Delete
	3. Attrib	/ Confirm Member Delete
	4. Total	/ Include Additional Qualifiers

- DSLIST - Data Sets Matching U105762.**.TEST*

Row 1 of 5

Command ===>

Scroll ===> PAGE

Command - Enter "/" to select action

Tracks %Used XT Device

U105762.A.B.TESTA	15	6	1	3390
U105762.A.B.TESTAB	15	6	1	3390
U105762.TEST.NEW.PDS	15	6	1	3390
U105762.TEST.PDS	15	6	1	3390
U105762.VIMAL.TEST	15	13	1	3390

***** End of Data Set list *****

OPTION - 3.4

- When the Initial View Option is chosen as 3 the following is displayed for the below search criteria.

```
Enter one or both of the parameters below:  
Dsname Level . . . U105762.**.TEST*  
Volume serial . . .  
  
Data set list options  
Initial View . . . 3 1. Volume      Enter "/" to select option  
                      2. Space        / Confirm Data Set Delete  
                      3. Attrib       / Confirm Member Delete  
                      4. Total        / Include Additional Qualifiers  
  
-----  
DSLIST - Data Sets Matching U105762.**.TEST*          Row 1 of 4  
Command ===>                                         Scroll ===> PAGE  
  
Command - Enter "/" to select action  
-----  
U105762.A.B.TESTA                                Dsorg  Recfm  Lrecl  Blksz  
U105762.A.B.TESTAB                               PO     FB      80    8000  
U105762.TEST.NEW.PDS                            PO     FB      80    8000  
U105762.TEST.PDS                                 PO     FB      80    8000  
***** End of Data Set list *****
```

OPTION - 3.4

- When the Initial View Option is chosen as 4 the following is displayed for the below search criteria.

- Enter one or both of the parameters below:

• Dsname Level . . . U105762.**.TEST*

• Volume serial . . .

- Data set list options
 - Initial View . . . 4 1. Volume Enter "/" to select option
 - 2. Space / Confirm Data Set Delete
 - 3. Attrib / Confirm Member Delete
 - 4. Total / Include Additional Qualifiers

- DLIST - Data Sets Matching U105762.**.TEST*

Row 1 of 4

- Command ==>

Scroll ==> PAGE

1

- Command - Enter "/" to select action

Message

Volume

• Tracks % XT Device Dsorg Recfm Lrecl Blksz Created Expires Referred

1

• U105762.A.B.TESTA

PUB005

1

• -----

1

• 15 6 1 3390 PO

*None***

1

• U105762.TEST.NEW.PDS

PUB005

1

• -----

1

• 15 6 1 3390 P

OPTION - 3.4

- The PF Keys PF10 & PF11 are cyclic in Nature in this panel i.e. Upon their repeated pressing the details displayed here are repeated again.

•Command - Enter "/" to select action	Message	Volume
•-----		
• U105762.A.B.TESTA		PUB005
•***** End of Data Set list *****	*****	*****

➤ Pressing PF 11 here will display the following

•Command - Enter "/" to select action	Tracks	%Used	XT	Device
•-----				
• U105762.A.B.TESTA	15	6	1	3390
•***** End of Data Set list *****	*****	*****	*****	*****

OPTION - 3.4 Use of ASTERISK

- Specifying the following Search Criteria will give the entire list of datasets as mentioned in the Previous screen.
 - T*.TEST.JCL
- Specifying **TCHN0*.TEST.JCL** will give the following
 - TCHN046.TEST.JCL
 - TCHN046.TEST.JCL.SORT
 - TCHN050.TEST.JCL
 - TCHN050.TEST.JCL.OBJ
 - TCHN050.TEST.JCL.OUTPUT
 - TCHN077.TEST.JCL
 - TCHN081.TEST.JCL
- Specifying **TCHN046.TEST.JCL** will give the following
 - TCHN046.TEST.JCL
 - TCHN046.TEST.JCL.SORT

Even though the TCHN046.TEST.JCL matches the exact criteria other datasets matching that criteria will also be listed e,g **TCHN046.TEST.JCL.SORT**

OPTION - 3.4 '*' Vs '%'

- Let us say the following datasets are cataloged in the system
 - U105762.A.B.TESTA
 - U105762.A.B.TESTAB
 - U105762.TEST.NEW.PDS
 - U105762.TEST.PDS
- Specifying **U105762.A.B.TEST*** will list the following
 - U105762.A.B.TESTA
 - U105762.A.B.TESTAB
- Specifying **U105762.A.B.TEST%** will list the following
 - U105762.A.B.TESTA
 - Here the "%" is used to denote any character of single length.
 - This explains the difference between an "*" asterisk and a Percentage "%".

OPTION - 3.4 “*” Vs “**”

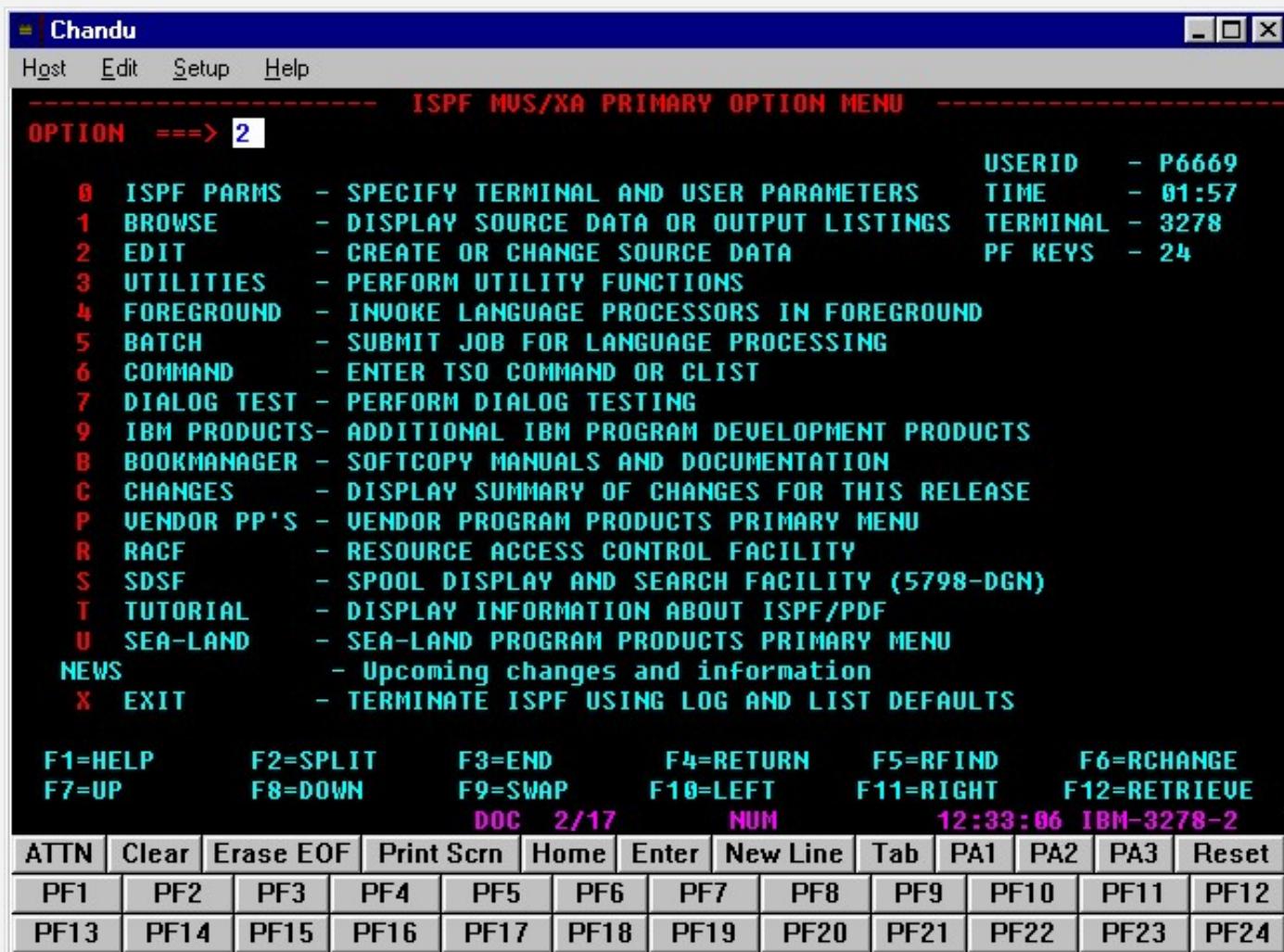
- Double asterisk can be used as a wild card for **None , One or more Nodes**
- Specifying **U105762.*.TEST*** will list nothing.
 - This is because there is no dataset that is cataloged with single Node between U105762 and TEST.
- Specifying **U105762.**.TEST*** will list the following.
 - U105762.A.B.TESTA
 - U105762.A.B.TESTAB
 - U105762.TEST.NEW.PDS
 - U105762.TEST.PDS
 - Here the datasets which have no nodes defined between U105762 and TEST are also listed.
 - This explains the difference between a Single asterisk and a double asterisk.

OPTION - 3.4

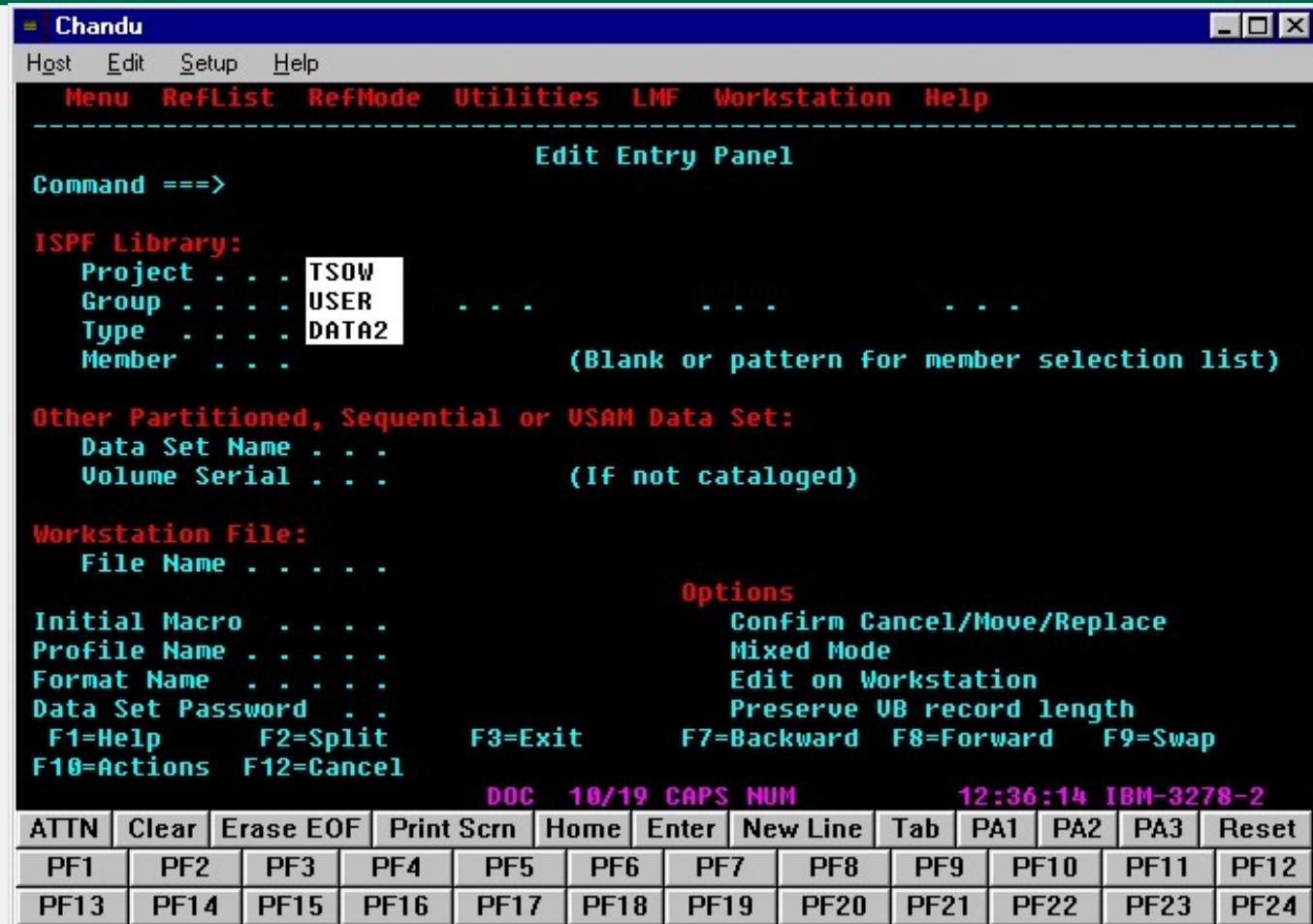
- So far we have seen how to specify the Search string such a way that the desired datasets are listed
- Now let us see what are the actions that can be performed on a dataset that gets listed.
- actions can be categorized as
 - Actions that change the content or the attributes of a dataset
 - Actions that don't.

S. No	Modifying Actions	Non Modifying Actions
1	Edit	View
2	Delete	Browse
3	Rename	Member List
4	Catalog	Print
5	Uncatalog	Print Index
6	Move	Reset
7	Compress	Copy
8	Free	Refadd
9		Info
10		Short Info
11		Exclude
12		Exclude 'NX'
13		Unexclude first 'NXF'
14		Unexclude last 'NXL'

ISPF/PDF EDIT



ISPF/PDF EDIT



This is the panel that is displayed when Option 2 is selected. The name of the file to be edited can be specified in two different areas of the panel.

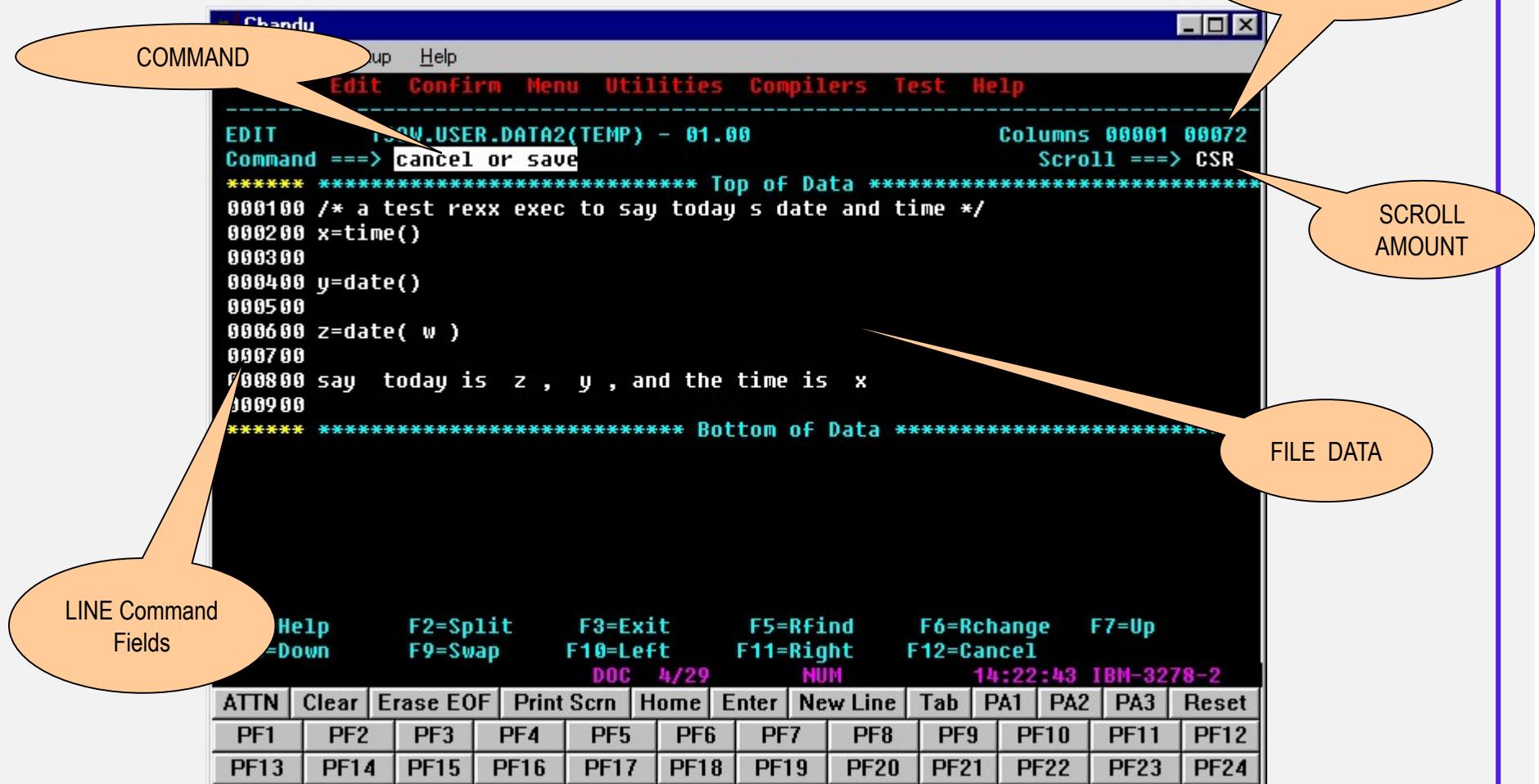
Scrolling

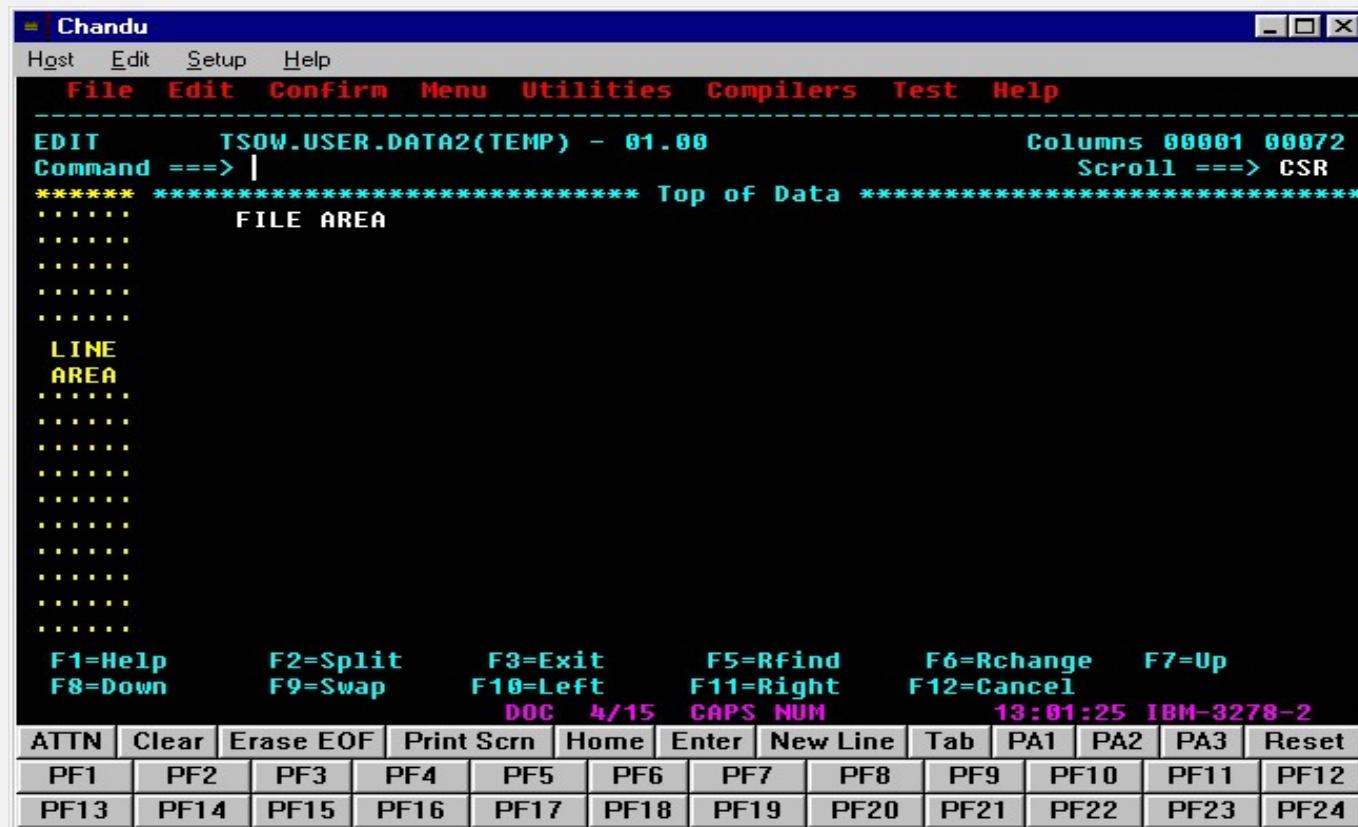
The scroll amount is placed next to SCROLL ==> This field can contain one of the following:

- **A number** from 1 to 9999 for number of lines (up and down) or number of columns (left and right)
 - **PAGE** or P to scroll by page. A "page" is one full screen.
 - **HALF** or H to scroll by half page. A "page" is one full screen.
 - **MAX** or M means scroll to top of data, bottom of data, left margin, or right margin. The previous scroll setting will be replaced after M has been used once.
 - **CSR** or C to scroll based on the current position of the cursor.

Edit Display Screen Format

- The following Screens shows the important regions in a EDIT screen.





For a new dataset entry can now begin and will start in the first column in the File Area.

Inside the Line Area, commands like **C**(Copy), **D**(Delete), **M**(Move), **R**(Repeat) can be specified.....

The screenshot shows a vintage computer terminal window titled "Chandu". The window has a menu bar with "File", "Edit", "Confirm", "Menu", "Utilities", "Compilers", "Test", and "Help". Below the menu bar, the title bar displays "EDIT TSOW.USER.DATA2(TEMP) - 01.00" and "Columns 00001 00072". The status bar at the bottom shows function keys F1 through F12, the date "DOC 4/15", the time "14:16:53", and the model "IBM-3278-2". A large text area contains REXX code:

```
***** **** Top of Data ****
/* a test REXX exec to say today's date and time */
x=time()
y=date()
z=date('W')
say 'today is 'z', 'y', and the time is 'x
```

Once you have the lines in the file that you want, you can press **ENTER**.
This results in

The screenshot shows a vintage computer terminal window titled "Chandu". The window has a menu bar with "File", "Edit", "Confirm", "Menu", "Utilities", "Compilers", "Test", and "Help". Below the menu bar, the title bar displays "EDIT TSOW.USER.DATA2(TEMP) - 01.00". The status bar at the bottom shows "Columns 00001 00072" and "Scroll ==> CSR". The main text area contains a REXX script:

```
***** **** Top of Data ****
000100 /* a test rexx exec to say today s date and time */
000200 x=time()
000300
000400 y=date()
000500
000600 z=date( w )
000700
000800 say today is z , y , and the time is x
000900
***** **** Bottom of Data ****
```

At the bottom of the screen, there is a function key legend and a numeric keypad:

F1=Help	F2=Split	F3=Exit	F5=Rfind	F6=Rchange	F7=Up						
F8=Down	F9=Swap	F10=Left	F11=Right	F12=Cancel							
DOC 4/29		NUM		14:22:43 IBM-3278-2							
ATTN	Clear	Erase EOF	Print Scrn	Home	Enter	New Line	Tab	PA1	PA2	PA3	Reset
PF1	PF2	PF3	PF4	PF5	PF6	PF7	PF8	PF9	PF10	PF11	PF12
PF13	PF14	PF15	PF16	PF17	PF18	PF19	PF20	PF21	PF22	PF23	PF24

If you type in **CANCEL**, the file is not saved. If you type **SAVE**, the file is saved. If you press **PF3** the file will be **SAVED** and returns to previous panel.

Line Area Commands

TYPE	COMMAND	PURPOSE
Copy	C, CC, Cx	Copy one or more lines
Move	M, MM, Mx	Move one or more lines
Insert	I, Ix	Insert one or more lines
Delete	D, DD, Dx	Delete one or more lines
Repeat	R, RR, Rx	Repeat(Duplicate) one or more lines
After, Before, Over	A, B, O	Used with Copy/Move for placement
Exclude	X, XX, Xn	Exclude(protect) lines from commands
Show	S, Sx	Show lines that were excluded
First	F, Fx	Show only the first x lines that were excluded
Last	L, Lx	ditto, but the LAST lines
Columns	COLS	Display the numbered line on the screen
Tabs	TABS	Display the tab positions
Caps	UC, LC	Change to upper/lower case characters

COLS line

The screenshot shows a vintage terminal window titled "Chandu". The menu bar includes "Host", "Edit", "Setup", "Help", "File", "Edit", "Confirm", "Menu", "Utilities", "Compilers", "Test", and "Help". The command line displays "EDIT TSOW.USER.DATA2(TEMP) - 01.00" and "Columns 00001 00072". The data list shows a series of area code entries. A scale line is visible in the middle of the data list, spanning from column 1 to column 7. The bottom of the screen displays function key definitions and a status bar.

ATTN	Clear	Erase EOF	Print Scrn	Home	Enter	New Line	Tab	PA1	PA2	PA3	Reset
PF1	PF2	PF3	PF4	PF5	PF6	PF7	PF8	PF9	PF10	PF11	PF12
PF13	PF14	PF15	PF16	PF17	PF18	PF19	PF20	PF21	PF22	PF23	PF24

The scale line can be shown anywhere using **cols**. To remove it, type in **RESET** on the command line.

Line Commands

- A line command is an edit command that is entered directly on the line to be processed. It is entered by overtyping the sequence number at the beginning of the line .
- Three of the most commonly used line commands are I (insert), D (delete), and R (repeat). Together they provide the most basic line editing functions.
 - Use I to insert one or more lines into the data
 - ✓ I - causes one line to be inserted.
 - ✓ I3 - causes 3 lines (or any number of lines) to be inserted.
 - ✓ An "insert" line is identified by ("") in the sequence field.
 - Use D or DD to delete one or more lines
 - ✓ D- identifies a single line that is to be deleted.
 - ✓ D5- identifies the first of 5 (or any number) of lines to be deleted.
 - ✓ DD- identifies the first and last lines of a block of lines to be deleted.
 - Use R or RR to repeat one or more lines
 - ✓ R - identifies a single line that is to be repeated.
 - ✓ R5 - identifies a line that is to be repeated 5 times.
 - ✓ RR - identifies the first and last lines of a block of lines to be repeated.
 - ✓ RR2 - identifies the first and last lines of a block to be repeated 2 times

Command line commands in PDF/EDIT

TYPE	COMMAND	PURPOSE
LOCATE	L	Locate a line number
FIND	F	Find a character string
CANCEL	CANcel	Don't save changes to the file
RECOVER	RECovery	Turn recovery on/off
SUBMIT	SUB	Submit a batch job for execution
CHANGE	C	Change a value on a line
CAPS	CAPS	Turn uppercase on/off for entire file
NULLS/BLANKS	NULLS	Affects inserting characters
CREATE	CREATE	Create new data from old data
REPLACE	REPL	Replace existing file
COPY	COPY	Copy data from another file
MOVE	MOVE	Physically move data from another source
UNDO	UNDO	Remove changes in reverse order

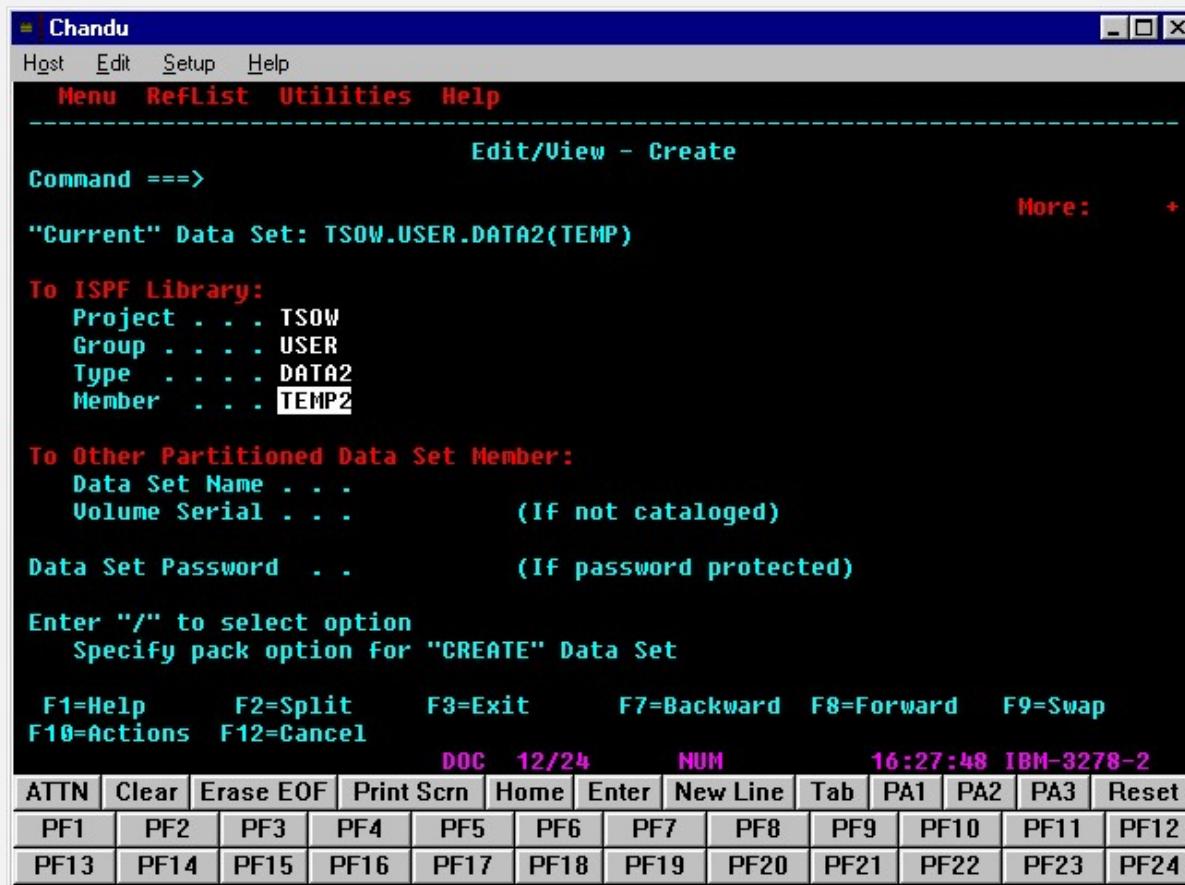
CREATE Command

The **CREATE** command can be used with the **COPY** or **MOVE** line commands to allow you to take some of the file you're looking at, and create a new file with that data. If you use **COPY**, your file remains unchanged; if you use **MOVE**, the lines will disappear from your file

```
Host Edit Setup Help
File Edit Confirm Menu Utilities Compilers Test Help
EDIT      TSO.W.USER.DATA2(TEMP) - 01.01          Columns 00001 00072
Command ==> create                                Scroll ==> CSR
***** **** Top of Data ****
000001 /* -----
000002 /*       An exec to tell where an area code is.... */
000003 /* -----
000004 answer='YES'
000005 areacode.= unknown
000006 AREACODE.205 = ALABAMA
000007 AREA CODE.713 = TE XAS
000008 AREACODE.907 = ALASKA
000009 AREACODE.602 = ARIZONA
000010 AREACOD E.501 = ARKANSAS
cc      AREACODE.715 = dairyland
000012 AREACODE.715 = dairyland
cc      AREACODE.715 = dairyland
000014 AREACODE.715 = dairyland
000015 AREACODE.307 = WYOMING
000016 arg areacode .
000017 AREACODE.205 = ALABAMA
F1=Help    F2=Split   F3=Exit     F5=RFind    F6=Rchange   F7=Up
F8=Down   F9=Swap    F10=Left    F11=Right   F12=Cancel
DOC 4/21    NUM 16:22:00 IBM-3278-2
ATTN Clear Erase EOF Print Scrn Home Enter New Line Tab PA1 PA2 PA3 Reset
PF1 PF2 PF3 PF4 PF5 PF6 PF7 PF8 PF9 PF10 PF11 PF12
PF13 PF14 PF15 PF16 PF17 PF18 PF19 PF20 PF21 PF22 PF23 PF24
```

CREATE Command

We are creating a new file, called **TEMP2**, that will contain the lines specified in our file



REPLACE Command

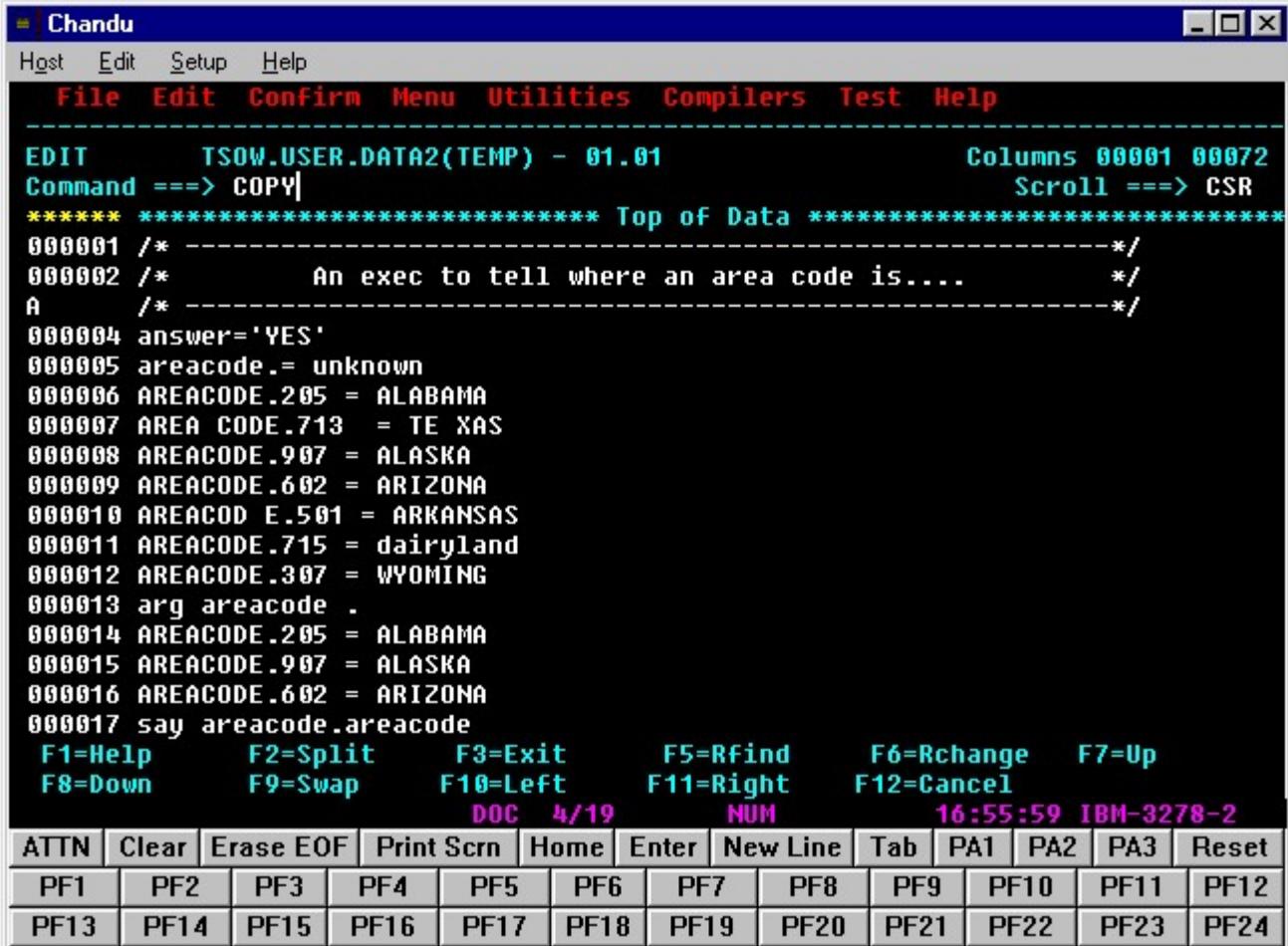
The **REPLACE** command works similar to the **CREATE** command,

It will overwrite data in the existing file with data from our file.

This command can also be used with **MOVE** or **COPY** and the results are the same as when **MOVE** and **COPY** were used with the **CREATE** command.

COPY Command

The **COPY** command used to get data from another file into the one you are editing. You **MUST** specify **BEFORE** or **AFTER** to position the data received

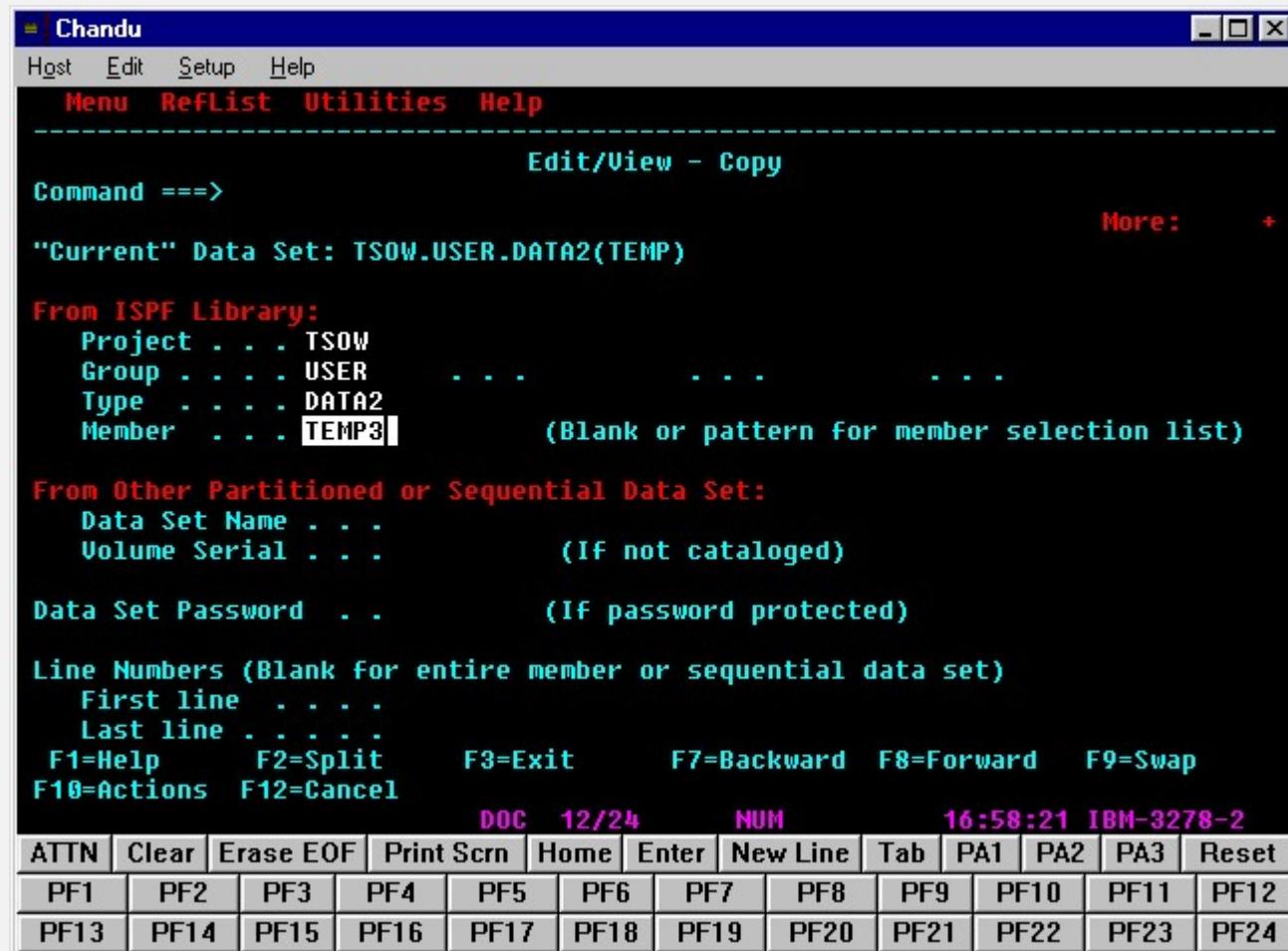


The screenshot shows a window titled "Chudu" with a menu bar including Host, Edit, Setup, Help, File, Edit, Confirm, Menu, Utilities, Compilers, Test, and Help. The main area displays an edit session for file "TSOW.USER.DATA2(TEMP) - 01.01". The command "COPY" has been entered, and the text "Top of Data" is displayed above the current code lines. The code lists various area codes and their corresponding states. The bottom of the screen shows function key definitions and a status bar indicating the document is at page 4/19, the time is 16:55:59, and the machine is an IBM-3278-2.

```
EDIT TSOW.USER.DATA2(TEMP) - 01.01 Columns 00001 00072
Command ==> COPY
***** **** Top of Data ****
000001 /* ----- */
000002 /* An exec to tell where an area code is.... */
A /* ----- */
000004 answer='YES'
000005 areacode.= unknown
000006 AREACODE.205 = ALABAMA
000007 AREA CODE.713 = TE XAS
000008 AREACODE.907 = ALASKA
000009 AREACODE.602 = ARIZONA
000010 AREACOD E.501 = ARKANSAS
000011 AREACODE.715 = dairyland
000012 AREACODE.307 = WYOMING
000013 arg areacode .
000014 AREACODE.205 = ALABAMA
000015 AREACODE.907 = ALASKA
000016 AREACODE.602 = ARIZONA
000017 say areacode.areacode
F1=Help F2=Split F3=Exit F5=Rfind F6=Rchange F7=Up
F8=Down F9=Swap F10=Left F11=Right F12=Cancel
DOC 4/19 NUM 16:55:59 IBM-3278-2
ATTN Clear Erase EOF Print Scrn Home Enter New Line Tab PA1 PA2 PA3 Reset
PF1 PF2 PF3 PF4 PF5 PF6 PF7 PF8 PF9 PF10 PF11 PF12
PF13 PF14 PF15 PF16 PF17 PF18 PF19 PF20 PF21 PF22 PF23 PF24
```

COPY Command

The details of file from where the data is copied entered in this panel



MOVE Command

The **MOVE** command works similar to the **COPY** command,

But after copying lines into your file the original file will be deleted

The screenshot shows a terminal window titled "Chandu". The menu bar includes "Host", "Edit", "Setup", "Help", "File", "Edit", "Confirm", "Menu", "Utilities", "Compilers", "Test", and "Help". The main window displays the following text:

```
EDIT      TSOW.USER.DATA2(TEMP) - 01.01          Member TEMP3 copied
Command ==> MOVE|                                     Scroll ==> CSR
***** ***** Top of Data *****
000001 /* -----
000002 /*     An exec to tell where an area code is....   */
000003 /* -----
B    AREACODE.715 = dairyland
000005 AREACODE.715 = dairyland
000006 AREACODE.715 = dairyland
000007 answer='YES'
000008 areacode.= unknown
000009 AREACODE.205 = ALABAMA
000010 AREA CODE.713 = TE XAS
000011 AREACODE.907 = ALASKA
000012 AREACODE.602 = ARIZONA
000013 AREACOD E.501 = ARKANSAS
000014 AREACODE.715 = dairyland
000015 AREACODE.307 = WYOMING
000016 arg areacode .
000017 AREACODE.205 = ALABAMA
F1=Help      F2=Split      F3=Exit      F5=Rfind      F6=Rchange      F7=Up
F8=Down      F9=Swap       F10=Left      F11=Right      F12=Cancel
DOC 4/19      NUM      17:02:20 IBM-3278-2
```

The bottom of the window features a series of function keys and a numeric keypad.

ATTN	Clear	Erase EOF	Print Scrn	Home	Enter	New Line	Tab	PA1	PA2	PA3	Reset
PF1	PF2	PF3	PF4	PF5	PF6	PF7	PF8	PF9	PF10	PF11	PF12
PF13	PF14	PF15	PF16	PF17	PF18	PF19	PF20	PF21	PF22	PF23	PF24

Displaying Text

- Upper case or Lower case

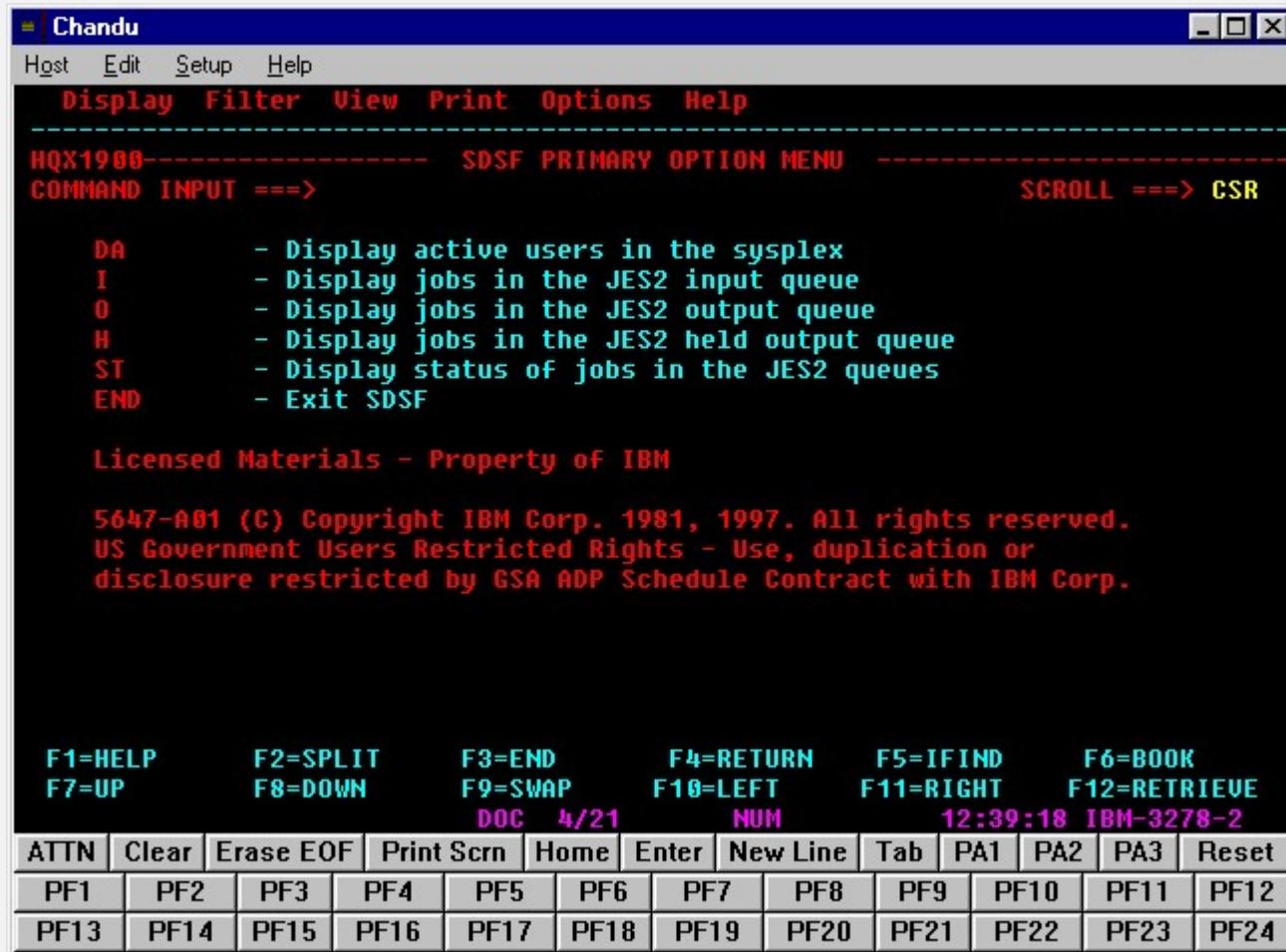
- CAPS OFF lets you enter text in upper and lower case. CAPS ON converts all your text to upper case as you enter it
 - ✓ COMMAND=→ CAPS ON or CAPS OFF
- To convert lines to upper or lower case after you type them in, two sets of commands are provided. You type the command over the line numbers.
 - ✓ UC Convert a single line to Upper Case
 - ✓ UCn Convert n lines to upper case
 - ✓ UCC Convert a block of lines to upper case.
 - ✓ LC Convert a single line to lower case
 - ✓ LCn Convert n lines to lower case
 - ✓ LCC Convert a block of lines lower case

System Display Search Facility - SDSF

- This is a product from IBM to monitor, manage, and control your MVS/JES2 system.
- The following actions can be performed with the SDSF utility.
 - Control job processing (hold, release, cancel, and purge jobs)
 - Monitor jobs while they are running
 - Browse jobs without printing
 - Control job classes
 - Control printers, punches, readers and initiators
 - Issue JES2 and MVS commands that affect jobs

Various Queues in SDSF

SDSF starts of with Primary Menu



Additional Functions

Line Commands

- The following are the most widely used line commands on the Status panel.

H	Hold a running Job
A	Release a Job on Hold
C	Cancel a Job
P	Cancel a job and purge the output
?	Display the output of a job dataset-wise.
S	Display the entire job output
O	Move the job to the Output Queue

Additional Functions

The details of the Job can be viewed by issuing "?" in the line command area

PREFIX=T*	DEST=(ALL)	OWNER=TCH*	SYSNAME=	C	P
NP	JOBNAME	JobID	Owner	Prty	Queue
?	TCHN111	TSU00144	TCHN111	15	EXECUTION
	TCHN113	TSU00145	TCHN113	15	EXECUTION

The below screen shows the details of the Job. Various Minor details such as JESMSGGLG; JESJCL; JESYMSG are also listed. If the details of these individual DD's are required select them and press an enter

SDSF JOB DATA SET DISPLAY - JOB TCHN111 (TSU00144)						
COMMAND INPUT ==>						
<hr/>						
PREFIX=T*	DEST=(ALL)	OWNER=TCH*	SYSNAME=	C	Dest	
NP	DDNAME	StepName	ProcStep	DSID	Owner	
	JESMSGGLG	JES2			2	TCHN111 K
	JESJCL	JES2			3	TCHN111 K
	JESYMSG	JES2			4	TCHN111 K

Additional Functions

- Select the DD whose detailed information is required.

```
Display Filter View Print Options Help
-----
SDSF JOB DATA SET DISPLAY - JOB TCHN111 (TSU00144)
COMMAND INPUT ===>
PREFIX=T* DEST=(ALL) OWNER=TCH* SYSNAME=
NP DDNAME StepName ProcStep DSID Owner C Dest
JESMSGLG JES2 2 TCHN111 K
s JESJCL JES2 3 TCHN111 K
JESYMSG JES2 4 TCHN111 K
```

- The following screen is displayed when the JESJCL is chosen

```
Display Filter View Print Options Help
-----
SDSF OUTPUT DISPLAY TCHN111 TSU00144 DSID      3 LINE 0      COLUMNS 02- 8
COMMAND INPUT ===> SCROLL ===> PR
*****
***** TOP OF DATA *****
1 //TCHN111 JOB 'TRGACCT#'
2 //IKJUSER EXEC IKJUSER
XX*-----
XX* SERVERPAC LOGON PROCEDURE
XX*
XX* THIS PROCEDURE ENABLES USERS TO LOG ON TO TSO/E.
XX* THE CLIST ISPPDF, WHICH RESIDES IN CPAC.CMDPROC,
XX* IS EXECUTED AT FIRST TO INVOKE THE ISPF.
XX*-----
3 XXIKJACCN PROC
4 XXIKJACCN EXEC PGM=IKJEFT01,DYNAMNBR=500,PARM=ISPUSER
5 XXSYSPROC DD DISP=SHR,DSN=CPAC.CMDPROC
```

Log off

- Let us say you have opened four session (Using START Command) and you want to LOGOFF from all the sessions.
- Type =X command from the Command Prompt. This will end the current screen and Perform this until you end all the screens and return to only one screen.
- Finally it will end the ISPF session and will take you to the TSO command Prompt.



➤ ENTER LOGOFF from the command Prompt this will LOGOFF you from the System.

Log & List Datasets Processing

- There are few options that can control your Logoff Procedure. Before you know about the Logoff options you should Know what a Log dataset and List Datasets mean.
- LOG Datasets:
 - All the activities of the user in a Session are logged in a Log dataset. This is generally stored in <USERID>.<SPFLOG#.LIST>
- LIST Datasets:
 - The Save commands in a dataset list or a member list panel will save the details in <USERID>.<SPF#.LIST> dataset.

Job Control Language (JCL)

Job & JCL

- A job is the execution of one or more related programs in sequence
- Each program to be executed by a job is called Job Step
- For example, when the programmer needs to execute two programs in order to sort a file and take a report out of it. Then this job consists of two steps
- JCL is a set of control statements that provide the specifications necessary to process the job

JCL Overview

Job Management

- Operating System (MVS, OS/390, etc) facilitates the following services in order to manage the execution of jobs.
 - Read and store jobs
 - Select jobs for execution based on relative importance of each job
 - Allocate the resources to jobs as they execute
 - Process the printed outputs produced by the jobs.
- Job Entry Subsystem is MVS Component that keeps track of the job that enters the system, presents them to MVS for processing, and sends their spooled output to the correct destination, normally a printer.

JCL Overview

Job Entry into the system

- In order to enter or submit a job into the system, the terminal user issues a SUBMIT command.
- JES2 or JES3 reads the job stream from the DASD file and copy it to a job queue, which is called as the '**JES spool**'.
- JES component called the '**Internal Reader**' processes the input job stream

JCL Overview

- **Processing Job's output**
 - Like jobs, SYSOUT data is assigned an output class that determines how the output will be handled.
 - Common output classes are
 - ✓ A - Standard Printer Output
 - ✓ B - Standard card punch
 - ✓ Z - Held output (Output is held indefinitely so that it can be examined from a TSO terminal)
- **Purging Jobs**
 - Purging the job means that the JES spool space the job used is freed, so that it can be used by other jobs.
 - Any JES control blocks associated with the job are also deleted.
 - Once the job is purged, JES no longer knows its existence.

Role of J C L

- Not used to write computer programs.
- Instead, it is more concerned with input/output telling the operating system everything it needs to know about the requirements.
- It provides the means of communicating between an application program and the operating system and computer hardware

JCL format

Identifier [name] [operation] [parameters] [comment]

Identifier

First two columns must contain //

Exceptions :

```
/* instream data delimiter  
/* comment
```

Name

- Associates name with a JCL statement
- May have 1 to 8 alphanumeric chars and national chars
- Cannot begin with numeric character
- Required for JOB statement
- Optional for EXEC and DD statements
- Must start in column 3

JCL format

Identifier [name] [operation] [parameters] [comment]

Operation

- Specifies a statement function
- Should be separated from name field by at least one blank
- Typically JOB, EXEC, DD

Parameters

- Used to code one or more parameters for the operation
- Begins at least one column after the end of operation field
- Parameters separated with commas
- Spaces not allowed in the list
- Can extend up to column 71

Comment

- Begins in the position after the space that marks end of parameter list

JCL example

The diagram illustrates a JCL (Job Control Language) job card. A light blue rectangular box contains the JCL code. An orange speech bubble labeled "Operation / Statement" points to the first two lines of the code: //KEAN01A JOB and // EXEC. Another orange speech bubble labeled "Parameters" points to the parameters listed on the third line: PGM=IEBGENER. The JCL code itself is as follows:

```
//KEAN01A JOB USER=KEAN01,PASSWORD=ABCDE  
// EXEC PGM=IEBGENER  
//SYSPRINT DD SYSOUT=A  
//SYSUT1 DD DSN=KEAN01.COPYLIB.COBOL(PRTPRG),  
// DISP=SHR  
//SYSUT2 DD SYSOUT=*  
//SYSIN DD DUMMY
```

Parameters

Positional parameters

- MVS interprets the parameter by its position
- If a parameter is omitted, extra comma is coded to account for missing parameter

Key word parameters

- Parameters are coded in any order
- Parameters are identified by a keyword
- Keyword parameters should follow positional parameters

Example:

UNIT=SYSDA,DISP=SHR

Parameters

Sub parameters

- Parameters can have sub parameters
- Sub parameters can be positional or keyword
- Parentheses to be used for more than one sub parameter

Examples:

```
DCB=(DSORG=PO,LRECL=80,BLKSIZE=800)
```

```
DCB=DSORG=PO
```

```
DISP=(SHR,KEEP,KEEP)
```

```
DISP=SHR
```

General Rules

- Lower-case letters not allowed
- NAME field is optional
- NAME must begin in column 3 if used
- must code one or more blanks if name is omitted
- PARAMETERS must end before column 72
- Can continue in the next line starting with //
- PARAMETERS are separated by commas
- Spaces not allowed in parameter list
- Line with only // is null statement which indicates end of job

JOB Statement

- JOB statement is always the first JCL statement coded for a job.
- Three important basic functions of a JOB statement are
 - Identifies the Job name that operating system uses to refer the job.
 - Supplies the Accounting Information that will be used by the MVS to determine who is responsible for the job.
 - Supplies run parameters that influence or limit how the job is processed.
- One job should have only one JOB statement and necessary parameters
- Single JOB can contain many STEPs and a single STEP in turn may use many datasets.

JOB Statement

- Syntax:

//Jobname	JOB	operands	comments
-----------	-----	----------	----------

- Example

```
//OZA092AX JOB ACCT12,'TOM SAWYER',CLASS=A,  
//           MSGCLASS=C,REGION=1000,  
//           ADRSPC=REAL,TYPRUN=SCAN,  
//           TIME=1440,MSGLEVEL=(1,1)
```

- Job name (OZA092AX) and the operation field (JOB) are the minimum required in the JOB statement.

JOB statement - Positional parameters

Job accounting information

Syntax :

(account-number,additional-information)

'account-number,additional-information'

account-number

(,additional-information)

Example :

```
//JOB2 JOB (A123,dept1)
//JOB2 JOB 'A123,dept'
//JOB2 JOB (A123,'DEPT/MIS',124)
//JOB2 JOB A123
//JOB2 JOB (,DEPT1)
```

JOB statement - Positional parameters

Programmer name

- It can be 1 to 20 characters long
- If name has special chars or space it should be enclosed in single quotes
- If it contains single quote, two single quotes to be used
- Hyphen cannot be part of the name

Example :

```
//JOB2 JOB (A123),Ramana
//JOB2 JOB (A123),'O"Sullivan'
//JOB2 JOB (A123),'Ramana Reddy'
//JOB2 JOB ,Ramana
```

JOB Statement – Key word parameters

- **CLASS**

- Specifies Job Class (A-Z)
- Established to group jobs
- Installation dependent
- Example:

```
//JOBA JOB CLASS=N,MSGCLASS=A
```

- **PRTY**

- Specifies Job selection priority
- Priority within the job class

- **MSGCLASS**

- Message Output class (A-Z)
- Installation dependent

JOB Statement – Key word parameters

- **TIME**

- Maximum processor time required by the job.
- TIME = (min,sec) where min <=1440 and sec <60.
- When TIME=1440 is coded the job will not be timed out.
- This Parameter can be coded in EXEC statement also.
- Example:

```
//JOBA JOB PRTY=4 ,TIME=20
```

- **REGION**

- Amount of memory needed for a job.
- A default value may be set by the installation for each PRTY value.
- Can also be coded in an EXEC statement

- **NOTIFY**

- Information regarding completion of the background job's execution is sent to the TSO (Time Sharing Option) user whose identification is given through this Parameter.
- Example:

```
//JOBA JOB REGION=1MB ,NOTIFY=&SYSUID
```

JOB Statement – Key word parameters

- **ADDRSPC**

- Specifies whether real or virtual address space is required.
- ADDRSPC=VIRT is default.
- When ADDRSPC=REAL is specified, then REGION Parameter has to be coded.
- Requesting real storage places heavier demands on system resources.
- This Parameter can also be coded on EXEC statement.
- Example:

```
//JOBA JOB ADDRSPC=REAL
```

JOB Statement – Key word parameters

- **MSGLEVEL**

- Determines the specific amount of output (system messages and JCL statements) that is to be written to the MSGCLASS output device.
- This contains two positional sub Parameters

MSGLEVEL=(JCL statements, JES allocation messages)

0 - Allocation and termination messages in abnormal termination
1 - Allocation and termination messages in all situations

0 - only JOB statement is printed
1 - all JCL expansions of the procedures are printed
2 - only the JCL statements submitted are printed.

EXEC Statement

- A job may have many steps in it (maximum 255).
- Each step would in turn invoke a program or a procedure for task completion.
- Each step execution would certainly require details like the
 - Name of the program or procedure that has to be executed
 - Location of the program or procedure
 - Values that have to be passed to the program
 - Storage requirements
 - Effect on subsequent steps, when it terminates abnormally
 - Priority of the step within the job
- Format :

```
//stepname EXEC PGM=program-name,keyword-parameters
```

EXEC Statement – Positional parameters

- **PGM**
 - Identifies the name of the program
 - Example :

```
//STEP1 EXEC PGM=PAYUPDT
```

EXEC Statement – Key word parameters

- ACCT

- Specifies the accounting information for the job step
- Consists of one or more subparameters depending on installation
- Rarely used in EXEC statement
- Syntax :

```
ACCT=(accounting-information,...)
ACCT=accounting-information
```

- Example :

```
//STEP2 EXEC PGM=pgm1,ACCT=(A123,RR)
//STEP2 EXEC PGM=pgm1,ACCT=(RR,47,'B=3421')
```

EXEC statement - Keyword parameters

- **COND**

- This Parameter determines whether or not to execute the remaining steps in the job based on the condition coded here.
- If the condition is true, the execution of the step is bypassed.
- Syntax

COND=(code,operator)

- Where code value varies from 0 to 4095 and operators are GT, GE, EQ, NE, LT and LE.
- System returns a code to the calling program after each step is executed. The return code of the job step is compared with the coded condition in the COND Parameter and action is taken accordingly.

EXEC statement - Keyword parameters

- **COND**

Some common condition codes

- | | |
|----|----------------------------------|
| 0 | Successful execution |
| 4 | Warning messages |
| 8 | Some error |
| 12 | More Serious error |
| 16 | Fatal error. Job cannot continue |

- **Full syntax**

```
COND=(code,operator)
COND=(code,operator,stepname)
COND=((code,operator),(code,operator))
COND=EVEN
COND=ONLY
```

EVEN indicates that step to be executed even previous step terminates abnormally

ONLY indicates that the step to be executed only if previous step terminates abnormally

EXEC statement - Keyword parameters

- COND

Examples :

```
//STEP3 EXEC PGM=PGM1,COND=(4,GT)
```

Step bypassed if CC is less than 4

```
//STEP3 EXEC PGM=PGM1,COND=((4,GT),(6,EQ))
```

Step bypassed if CC is less than 4 or CC is equal to 6

```
//STEP3 EXEC PGM=PGM1,COND=(4,EQ,STEP1)
```

Step bypassed if CC of STEP1 is equal to 4

JOB & EXEC Common parameters

- Parameters common to both JOB & EXEC statements:
 - REGION
 - ADDRSPC
 - TIME
 - COND
- If the above Parameters are coded in the JOB and the EXEC statements (except TIME), then values specified in the JOB statements will override those specified in the EXEC statements.

DD Statement

- DD stands for Data Definition. DD Statements are helpful in specifying the input and the output datasets (equivalent to files in other languages) that are required for the execution of a particular job step.
- Maximum of 255 DD Statements can be coded in a job step
- In addition to the above, DD Statements are used to specify the following :
 - Space allocated to the datasets
 - Usage of the datasets mentioned
 - Whether the datasets have been created already
 - What is to be done with the datasets if the job completes successfully or abnormally

DD Statement

Format :

//ddname DD positional/keyword parameters

Positional parameters

*

DATA
DUMMY

Keyword parameters

DSN
DISP
UNIT
SPACE
DCB
VOLUME

DD Statement

DDnames reserved for system use

```
//JOBLIB  
//STEPLIB  
//SYSIN  
//SYSOUT  
//SYSABEND  
//SYSCHK  
//SYSUDUMP  
//SYSOUD  
//SYSDBOUT
```

DD Statement – positional parameters

DUMMY

- Specifies non existing dataset
- Used when JCL is to be tested for errors without using data sets
- When program tries to read DUMMY data set, end-of-file is encountered
- When program writes to DUMMY data set, the data is lost

Example :

```
//SYSIN DD DUMMY
```

We can also mention

```
//SYSIN DD DSN=NULLFILE
```

DD Statement – positional parameters

* / DATA

- Specifies instream data
- Instream data should be terminated by /*
- If terminator is omitted, next JCL statement is taken as terminator
- // can be part of instream data with DATA parameter

Examples :

```
//SYSIN DD *
A00101005995392
B00104008923019
/*
```

```
//SYSIN DD DATA
A00101005995392
C002010046901234
/*
```

```
//SYSIN DD DATA,DLM=@$
A00101005995392
B00104008923019
//INVTRAN
C002010046901234
@$
```

DD Statement – key word parameters

DSN or DSNAME

- **Specifies data set name**

SYNTAX :

```
DSN=datasetname
```

Example :

```
DSN=RAMANA.TEST.EMP
```

```
DSN=RAMANA.GROUP.DATA(FILE1)
```

DD Statement – key word parameters

Temporary Data Set

- Created during job execution and deleted after job completion
- Temp datasets identified if DSN parameter is omitted or name starts with &&

Example :

```
//JOBA JOB A123,'Ramana'  
//STEP1 EXEC PGM=PGM1  
//ABC DD DSN=&&ABCD  
//STEP2 EXEC PGM=PGM2  
//DATA2 DD DSN=*.STEP1.ABC
```

Referbacks

- Reference to values of parameters previously specified .
- Legal only within the Job
- Example :

```
//STEP1 EXEC    PGM=MDL1
//DD8    DD      DSN=EMPFILE ,DISP=MOD
//.....<other steps>.....
//STEP9 EXEC    PGM=MDL2
//DD2    DD      DSN=*.STEP1.DD8
```

DD Statement – key word parameters

DISP

- Describes status of the dataset prior to the step execution.
- Indicates what processing can be done with the dataset.
- Implies what has to be done with the dataset, if the job is successful or a failure.
- Syntax :

DISP=(Status before execution, normal disposition, abnormal disposition)

Possible parameter values

Status before execution	Normal Disposition	Abnormal disposition
NEW	DELETE	DELETE
OLD	KEEP	KEEP
SHR	CATLG	CATLG
MOD	UNCATLG	UNCATLG
	PASS	

DD Statement – key word parameters

DISP

Disposition	Meaning
NEW	Specifies that the dataset is created in this job step
OLD	Specifies that the dataset already exists. If OLD is coded, then the user gets exclusive control on the dataset and no one can read/write until the job is over.
MOD	If the dataset already exists, then all the output records will be appended next to the existing records. If the dataset does not already exists, it will be created .
SHR	Specifies that the dataset already exists and other jobs can also access this dataset
DELETE	Specifies that the dataset to be deleted once the step execution is over.
KEEP	Specifies that the dataset should be kept after the end of the job.
PASS	Specifies that the dataset will be used in subsequent job steps and the dataset should be deleted once the job is over.
CATLG	Specifies that the dataset should be kept and catalogued
UNCATLG	The catalog entry for the dataset will be removed but the dataset will remain.

DD Statement – key word parameters

DISP

- If status is not coded, it defaults to NEW
- The normal and abnormal DISP defaults to the status of the dataset
 - ✓ OLD datasets are KEPT
 - ✓ NEW datasets are DELETED
 - ✓ SHR datasets are KEPT
 - ✓ MOD datasets are KEPT
- If only normal DISP is coded, the abnormal DISP defaults to what is coded in the normal DISP.

DD Statement – key word parameters

UNIT

- Specifies input / output device
- can be a number indicating model type
- Name of group to which the device belongs
- Examples :

```
//DD1    DD      DSN=PROJ.PAY.TEST1,DISP=SHR
//                      UNIT=SO5 (Device Address)
//DD2    DD      DSN=PROJ.PAY.TEST2,DISP=SHR
//                      UNIT=3390 (Device Type)
//DD3    DD      DSN=PROJ.PAY.TEST3,DISP=SHR
//                      UNIT=SYSDA (Device Group)
//DD4    DD      DSN=PROJ.PAY.TEST4,DISP=SHR
//                      UNIT=AFF=DD3 (Affinity Parameter)
```

- The last option specifies that the device used by the dataset specified in the prior DD name, should be used for this dataset also, meaning that both the datasets will reside on the same volume.

DD Statement – key word parameters

VOL

Specifies volume serial number

Examples :

```
VOL=SER=T104  
VOL=REF=*.STEP1.DD1
```

DD Statement – key word parameters

SPACE

- Used to allocate storage for new data sets on DASD
- Space can be in tracks, cylinders, blocks
- Directory space also to be mentioned for PDS
- Syntax :

```
SPACE=(CYL,(primary,secondary,directory),RLSE,CONTIG)
SPACE=(TRK,(primary,secondary,directory),RLSE,CONTIG)
SPACE=(blks,(primary,secondary,directory),RLSE,CONTIG)
```

- blks indicates block size (allocation rounded to number of tracks)
- Maximum 15 secondary extents are allocated to data set
- RLSE indicates that the space not used to be released on close
- CONTIG indicates that space should be contiguous

DD Statement – key word parameters

SPACE

- Example:

```
//DD1    DD      DSN=PROJ.PAY.TEST,  
//                      SPACE=(TRK,(20,5,10),RLSE)
```

- In the above example, primarily 20 tracks are allocated to the dataset while creation.
- Once the primary space becomes insufficient, then the space will be extended in terms of 5 tracks (maximum of 15 extensions).
- Here 10 blocks are requested for directory information regarding the members of this PDS.

DD Statement – key word parameters

- **DCB** (DATA CONTROL BLOCK) Parameter

- Supplies information to the system regarding the organization of data in a dataset.
- Omitted if the dataset already exists.
- Useful for output datasets which are created during the job
- Can take DCB parameters from other dataset by mentioning its name
- Syntax

```
DCB=(RECFM=format,LRECL=record length, BLKSIZE=Block size,DSORG=org)
```

```
DCB=datasetname
```

- RECFM=format

- ✓ Used to specify the Record format of the dataset.

- ✓ Format can be

- F Fixed length

- FB Fixed length Blocked

- V Variable length

- VB Variable length Blocked

- U Undefined

DD Statement – key word parameters

➤ LRECL=num

- ✓ Logical Record Length for fixed length records.
- ✓ For variable length records, it would be equal to largest record size +4 bytes. These additional 4 bytes are used to store information in the Record Descriptor Word (RDW)

➤ BLKSIZE=num

- ✓ Used to specify the block size for a dataset.
- ✓ Max. value is 32,760 bytes

➤ DSORG=org

- ✓ Used to specify type of organization
- ✓ Can be PS or PO (PDS)

DD Statement – key word parameters

DCB

Example:

```
//DD1      DD      DSN=PROJ.PAY.TEST,  
              DISP=(NEW,CATLG),  
              SPACE=(100,(10,10)),  
DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
```

- The above example is for creating a new dataset with the record format as Fixed Block with logical record length as 80 and the block size as 800 bytes.

```
//DD1      DD      DSN=PROJ.PAY.TEST,  
              DISP=(NEW,CATLG),  
              SPACE=(100,(10,10)),  
DCB=PROJ.PAY.OLD
```

- This example shows that DCB parameters of PROJ.PAY.OLD are to be used

DD Statement – key word parameters

SYSOUT

- Routes the output to a printer
- Value of this parameter is the output class related to output destination
- Syntax :

```
//ddname DD SYSOUT=class
```

- Example:

```
//ABCD DD SYSOUT=A
```

```
//XYZ DD SYSOUT=*
```

Same as JOB
class

STEPLIB

- A program can be executed through the EXEC statements as follows :

```
//STEP100 EXEC PGM=PROGRAM1
```

- The program that has to be executed has to be loaded onto the memory. All the programs will reside either in the system library or user defined libraries.
- System automatically searches for the program in the system library and if it is not found, then it will search in the user defined libraries.
- Location of the program that is to be executed is defined through a ddname STEPLIB .
- Example:

```
//STEP100      EXEC      PGM=PROGRAM1
//STEPLIB       DD        DSN=USERID.PGMLIST,DISP=SHR
```

JOBLIB

- If there are many steps involved in a job and most of those steps specified are going to execute the programs from the same library, then the library can be mentioned in the JOBLIB ddname with JOB statement.
- If both JOBLIB and STEPLIB are present, then the library specified in the STEPLIB will only be searched for that particular step and the JOBLIB is ignored
- Example:

```
//JOB123      JOB     S123,'AAA'  
//JOBLIB       DD      DSN=TEST.PGMLIB,DISP=SHR  
//STEP100      EXEC    PGM=PROGRAM1  
.....  
//STEP160      EXEC    PGM=PROGRAM2  
//STEPLIB      DD      DSN=USERID.PGMLIST,DISP=SHR
```

In the above example, PROGRAM1 will be searched in TEST.PGMLIB and PROGRAM2 will be searched in USERID.PGMLIST alone.

Concatenated datasets

- Concatenated datasets can be specified using a single DD name
- All of them treated as a single data source

Example :

```
//JOB1      JOB    (A123) ,Ramana
//STEP1     EXEC   PGM=CPYCBL
//DD1       DD     DSN=RAMANA.TEST.FIL1 ,DISP=SHR
//          DD     DSN=RAMANA.TEST.FIL2 ,DISP=SHR
//          DD     DSN=RAMANA.TEST.FIL3 ,DISP=SHR
//          DD     DSN=RAMANA.TEST.FIL4 ,DISP=OLD
//DD2       DD     SYSOUT=A
```

SYSIN ddname

- Relates to standard input
- response for COBOL ACCEPT statement can be given through //SYSIN
- Cannot be specified in procedures
- Example :

```
//STEP2 EXEC    PGM=LOAD MODULE
//DD8 DD          DSN=EMPFILE ,DIP=MOD
//SYSIN DD *
99999
KAMAL
/*
```

SYSOUT ddname

- Relates to standard output
- Output of COBOL DISPLAY statement can be sent to //SYSOUT
- Example :

```
//SYSOUT DD DSN=RAMANA TEMP OUT,DISP=SHR  
//SYSOUT DD SYSOUT=A
```

SYSPRINT ddname

- Specifies that an execution report of the PGM is required.
- It defines the output file containing messages from programs
- Example:

```
//STEP1 EXEC PGM=SORT
//SYSPRINT DD SYSOUT=*
//.....
//
```

DUMPS

- **SYSUDUMP**

- Used to request a formatted dump of the user program's work area and task control blocks if an abend occurs.
- No DCB parameters need be specified on the SYSUDUMP DD and it can be allocated to the SYSOUT CLASS, DASD or TAPE.
- SYNTAX:

```
//SYSUDUMP      DD      SYSOUT=Class
```

- **SYSMDUMP**

- Used to request an unformatted dump of the user address space and task control blocks, as well as the system areas, if an abend occurs.

JCL Procedures

Procedures

- When a group of JCL statements are used regularly by many users, then they can be grouped together and stored as PROCEDURES.
- These procedures can be accessed by JCL statements
- While invoking a procedure, it is possible to override a parameter already specified in the procedure
- Procedures are invoked in two ways in EXEC statements (simply by mentioning name or by using keyword PROC
- Example :

```
//STPNM      EXEC  DOIT  
//STPNM      EXEC  PROC=DOIT
```

Both the above will invoke a procedure DOIT

Types of Procedures

- **In-stream procedures**
 - Procedures coded inside JCL
 - These procedures cannot be accessed by other jobs
- **Cataloged procedures**
 - Procedures stored as a member of PDS
 - Traditionally these procedures are part of system library SYS1.PROCLIB
 - They can be part of private libraries also
 - Names of private libraries to be provided in JCLLIB statement

Instream Procedures

- Start with a key word PROC
- End with a key word PEND
- Procedure name cannot exceed 8 characters and can include alphabets,digits and national characters
- The first character cannot be a digit
- No more than 15 in-stream procedures allowed
- Instream DD statement is not allowed
- JOB statement not allowed
- Example :

```
//PROC1 PROC
//STEP1 EXEC PGM=PGM1
//DD1    DD DSN=RAMANA.TEST.DATA,DISP=SHR
//DD2    DD SYSOUT=*
//STEP2 EXEC PGM=PGM2
//DD1    DD DSN=RAMANA.NEW.DATA,DISP=SHR
//          PEND
```

Instream Procedures

- Example showing invoking Instream procedure

```
//JOB1    JOB    (A123),Ramana
//PROC1   PROC
//STEP1   EXEC PGM=PGM1
//DD1     DD DSN=RAMANA.TEST.DATA,DISP=SHR
//DD2     DD SYSOUT=*
//STEP2   EXEC PGM=PGM2
//DD1     DD DSN=RAMANA.NEW.DATA,DISP=SHR
//          PEND
//STEP3   EXEC PROC1
```

Cataloged Procedures

- Procedure stored as a PDS member
- System library SYS1.PROCLIB is the default location for the procedures
- JCLLIB statement is used to refer procedures in private libraries
- All rules of in-stream procedures apply except --
 - PROC statement is optional
 - PEND statement not allowed

JCLLIB

- A list of PDS names can be provided in this statement
- The libraries are searched in the specified order for the procedure
- Syntax :

```
//name JCLLIB ORDER=(library, library....)
```

- Example :

```
//JOB1      JOB     (A123),Ramana
//PROCLIB   JCLLIB  ORDER=(USR.TEST LIB1,USR.OTHER LIB)
//CALLIT    EXEC    CAT1
```

Search for CAT1 in
USR.TEST LIB1 and
USR.OTHER LIB in that order

Overriding parameters

- At times there may be a need to override one or more Parameters in procedures.
- Example:
 - suppose you wish to use the JCL coded in a procedure which accesses a specific dataset, but you want to access a different dataset.
 - Instead of rewriting the procedure for the new dataset name, you can easily override this DD statement while invoking the procedure
 - The modified contents of the procedure will be active only for the duration of the job in which they exist. The original procedure remains intact
- Done in three different ways
 - Modify parameters of EXEC statement
 - Modify parameters coded in DD statements
 - Add entirely new DD statement to the procedure.

Modifying parameters in EXEC

- Parameters can be modified by coding as
parameter.stepname=value
- If step name is not given, it applies to all steps of the procedure
- To nullify a parameter, code as
parameter.stepname=
- Example

```
//PROC1      PROC
//STEP100     EXEC   PGM=PROGRAM1
//DD1         DD     DSN=PROJ.PAY.TEST1,DISP=SHR
//STEP110     EXEC   PGM=PROGRAM2,COND=(4,LE)
//DD2         DD     DSN=PROJ.PAY.TEST2,DISP=SHR
```

```
//JOB123      JOB    ABC,'XYZ'
//PROCLIB     JCLLIB ORDER=(USR.TEST LIB1,USR.OTHER LIB)
//EXEC1       EXEC   PROC1,COND STEP110=(8,LT)
```

Modifying parameters in DD

- Parameters can be modified in DD statement by coding as

```
//stepname.ddname DD parameter=value
```

- Example :

```
//PROC1          PROC
//STEP100        EXEC   PGM=PROGRAM1
//DD1            DD     DSN=PROJ.PAY.TEST1,DISP=SHR
//STEP110        EXEC   PGM=PROGRAM2,COND=(4,LE)
//DD2            DD     DSN=PROJ.PAY.TEST2,DISP=SHR
```

```
//JOB123         JOB    ABC,'XYZ'
//PROCLIB        JCLLIB ORDER=(USR.TEST LIB1,USR.OTHER LIB)
//EXEC1          EXEC   PROC1
//STEP100.DD1    DD     DISP=OLD
//STEP110.DD2    DD     DISP=OLD
```

Add a new DD statement

- A new DD statement can be added to the procedure
- It is convenient if the procedure uses different dataset each time
- Example :

```
//PROC1      PROC
//STEP100    EXEC   PGM=PROGRAM1
//STEP110    EXEC   PGM=PROGRAM2,COND=(4,LE)
//DD2        DD     DSN=PROJ.PAY.TEST2,DISP=SHR
```

```
//JOB123    JOB    ABC,'XYZ'
//PROCLIB   JCLLIB ORDER=(USR.TEST LIB1,USR.OTHER LIB)
//EXEC1     EXEC   PROC1
//STEP100.DD1 DD     DSN=PROJ.PAY.TEST1,DISP=SHR
```

Modifying concatenated datasets

- Concatenated datasets can be modified by coding DD statements in the same order
- Example :

```
//PROC1      PROC
//STEP100    EXEC   PGM=PROGRAM1
//DD1        DD      DSN=PROJ1.PAY.TEST1,DISP=SHR
//           DD      DSN=PROJ1.PAY.TEST2,DISP=SHR
//           DD      DSN=PROJ1.PAY.TEST3,DISP=SHR
```

```
//JOB123  JOB    ABC,'XYZ'
//PROCLIB   JCLLIB ORDER=(USR.TEST LIB1,USR.OTHER LIB)
//EXEC1    EXEC   PROC1
//STEP100.DD1 DD
//           DD      DSN=PROJ2.PAY.TEST1,DISP=OLD
//           DD
```

Symbolic parameters

- Used to generalize the procedures
- Parameters defined using ‘&’ followed by upto 7 characters in a procedure
- Actual value for these parameters to be provided while invoking the procedure
- Allowed only in operand field
- Parameter names cannot be symbolic
- Symbolic Parameters can be concatenated with each other, each having its original value.
- Concatenated symbolic Parameter cannot exceed 120 characters
- Default values can be provided in the procedure which will be used only when values not supplied during invocation

Symbolic parameters

- Example :

```
//PROC1      PROC
//STEP100     EXEC   PGM=PROGRAM1
//DD1         DD      DSN=&FILE,DISP=&HOW
//DD2         DD      SYSOUT=&CLS
```

```
//JOB123      JOB    (A123),RAMANA
//PROCLIB     JCLLIB ORDER=(USR.TEST LIB1,USR.OTHER LIB)
//EXEC1       EXEC   PROC1,FILE=RAMANA.PROJ.SET,HOW=SHR,CLS=A
```

Symbolic parameters

- Example (with default values) :

```
//PROC1      PROC  CLS=*,FILE=RAMANA.STATUS,HOW=SHR  
//STEP100     EXEC  PGM=PROGRAM1  
//DD1        DD    DSN=&FILE,DISP=&HOW  
//DD2        DD    SYSOUT=&CLS
```

```
//JOB123      JOB    (A123),RAMANA  
//PROCLIB     JCLLIB ORDER=(USR.TESTLIB1,USR.OTHERLIB)  
//EXEC1       EXEC   PROC1,HOW=SHR,CLS=A
```

- In this example, as value not provided for symbolic parameter FILE, it will default to RAMANA.STATUS

Generation Data Group (GDG)

- For certain applications where the master file is updated regularly, say daily, weekly, monthly or quarterly, through transaction files, it is necessary to keep the old master files intact.
- In order to maintain the old files intact wherever necessary without leading to any confusion, MVS provides the feature Generation Data Group (GDG).
- GDG helps maintaining files having chronological and functional relationship
- Example:
 - There is a dataset A which will be updated by certain application on a daily basis and it is required that all the versions of the files (the 30 versions of the dataset that are created) have to be retained.
 - In that case, maintaining a GDG is quite convenient

Generation Data Group (GDG)



Next generation



Current generation



Previous generations

Generation Data Group

- Before using GDG , We need to create
 - GDG Index
 - GDG Model
- IDCAMS (the 'AMS' stands for Access Method Services), utility is used to create GDG index.
- Control statement for creating GDG index

```
DEFINE GDG(NAME(entry name) -  
          LIMIT(number) -  
          EMPTY / NOEMPTY -  
          SCRATCH / NOSCRATCH)
```

EMPTY : all gens to be removed
NOEMPTY : only old gen to be removed
NOEMPTY is default

SCRATCH : Actual deletion
NOSCRATCH : Only uncataloged
SCRATCH is default

Generation Data Group

- Once the index has been created, a model data set must be created
- This model data set contains specifications for the DCB subparameters for all data sets that will belong to that GDG
- Model dataset helps us avoid entering DCB parameters for new member
- Model needs just VTOC entry. No space required
- Example :

```
//MYJOB          JOB      (A123),RAMANA
//STEP1           EXEC     PGM=IDCAMS
//SYSIN           DD      *
      DEFINE GDG(NAME(OZA122.EMP.MAST) -
                  LIMIT(10) -
                  NOEMPTY -
                  SCRATCH)
//STEP2           EXEC     PGM=IEFBR14
//MODEL1          DD      DSN=OZA122.EMP.MODEL,
//                           DISP=(NEW,KEEP,DELETE),
//                           UNIT=SYSDA,
//                           SPACE(TRK,0),
//                           DCB=(LRECL=80,RECFM=FB,BLKSIZE=800,DSORG=PS)
//
```

Generation Data Group

Example for using GDG :

```
//JOB1      JOB    (A123),Ramana
//          EXEC   PGM=UPDT
//TRANS     DD     DSN=OZA122.EMP.TRANS,DISP=OLD
//OLDMAST   DD     DSN=OZA122.EMP.MAST(0),DISP=OLD
//NEWMAST   DD     DSN=OZA122.EMP.MAST(+1),DISP=(NEW,CATLG),
//                  VOL=SER=A102,SPACE=(TRK,(10,5),RLSE),
//                  DCB=OZA122.EMP.MODEL
//ERRLIST   DD     SYSOUT=*
//SYSPRINT  DD     SYSOUT=*
```

Generation Data Group

- Certain rules to be followed while creating a GDG :
 - Max limit of number of datasets which can exist within one GDG is 256.
 - Normal convention that is followed while creating a dataset equally applies for creating a dataset within a GDG, except that the generation number has to be specified.
 - GDG's can reside either on tapes or DASD's.
 - For all new generation datasets the DSN and unit Parameters have to be coded.
 - For all new generation datasets the DISP Parameters must be set to CATLG
 - All DCB parameters are required if model does not exist

IBM Utilities

IBM Utilities

Utility programs or just utilities are generalized programs that can be used for common data processing functions. The following are some of the utility functions :

- IEBGENER
- IEFBR14
- IEBPTPCH
- IEBCOPY
- IEBCOMPR
- IEHMOVE
- SORT

IEBGENER

- One of the most commonly required utility functions for copying or printing data sets.
- Copying and printing a data set are essentially the same thing. To print a file, the file is copied to a SYSOUT data set
- A simple IEBGENER job requires four DD Statement.
 - SYSPRINT Defines an output message files
 - SYSUT1 Defines the Input file
 - SYSUT2 Defines the output file
 - SYSIN Defines a control file
- Also used for editing data while copying with the help of control statements though //SYSIN

IEBGENER

- Example: IEBGENER for copying a sequential file

```
//COPYFILE      EXEC   PGM=IEBGENER
//SYSPRINT      DD     SYSOUT=*
//SYSUT1        DD     DSN=PROJ.PAY.TEST1,DISP=SHR
//SYSUT2        DD     DSN=PROJ.PAY.TEST2,DISP=(NEW,CATLG),
//                      UNIT=SYSDA,VOL=SER=OZAV04,
//                      SPACE=(CYL,(1,1))
//SYSIN         DD     DUMMY
```

- The above example is used for copying a sequential file named PROJ.PAY.TEST1 to another new sequential file named PROJ.PAY.TEST2.
- In the above example, no control statements are supplied.

IEFBR14

- A do nothing utility used to create,delete,catalog or uncatalog data sets
- Examples :

```
//CRTFIL    EXEC PGM=IEFBR14
//STEP1      DD    DSN=OZA122.EMP.MAS,DISP=(NEW,CATLG),
               VOL=SER=A102,SPACE=(TRK,(10,5)),
               DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
```

- The above example allocates new dataset OZA122.EMP.MAS

```
//DLTFIL    EXEC PGM=IEFBR14
//STEP1      DD    DSN=OZA122.EMP.MAS,DISP=(OLD,DELETE)
```

- This example for deleting dataset OZA122.EMP.MAS

IEBCOPY

- To make copies of PDS
- Options
 - Copy data sets
 - Compress data sets
 - Merge two or more PDS
 - Copy PDS to a sequential file (unload)
 - Reload sequential data set into PDS
 - Select or Exclude members while copying
- If INDD refers to an existing PDS and OUTDD allocates new PDS, members copied to OUTDD
- If OUTDD refers to an existing PDS, members merged into OUTDD
- If OUTDD refers to sequential file, IEBCOPY unloads the INDD into sequential file
- If INDD refers to sequential file, IEBCOPY reloads into OUTDD
- If both INDD and OUTDD refer to the same DD statement, IEBCOPY compresses the PDS

IEBCOPY

- Control statements

```
COPY
```

```
OUTDD=ddname,INDD=ddname[,ddname...]
```

```
SELECT MEMBER=(member,member...)
```

```
EXCLUDE MEMBER=(member,member...)
```

- Example :

```
//COMPRESS      EXEC PGM=IEBCOPY
//SYSPRINT       DD SYSOUT=*
//INFILE        DD   DSN=USER1.LIB.P12,DISP=OLD
//OUTFILE       DD   DSN=USER1.LIB.P24,DISP=OLD
//SYSIN          DD      *
COPY OUTDD=OUTFILE,INDD=INFILE
EXCLUDE MEMBER=(PRG1,PRG2)
/*
```

SORT & MERGE Utilities

Introduction

- Data stored in a file in one sequence needs to be reordered to produce a particular kind of output.
- To resequence a file, MVS SORT/MERGE utility (also known as DFSORT) will be used.
- SORT function is to change the sequence of the file as per the request.
- MERGE function assumes that files are already sorted and will combine into a single file, again in a right sequence
- Real time example for SORT functionality
 - Employee file stored in employee number sequence might have to be reordered so that it is in sequence by employee name or department.

SORT & MERGE Utilities

JCL Requirements for Sort / Merge

- SORTIN
 - Specifies the input file to be sorted.
 - If more than one file has to be sorted, it can be concatenated.
- SORTINnn
 - Specifies the input files to be merged.
 - Maximum of 16 files can be merged.
 - nn must be consecutive number starting with 01.
- SORTOUT
 - Defines the output files for a sort or merge operations.
- SORTWKnn
 - Specifies the sort work files. nn should be consecutive number beginning with 01.
 - One or two files are sufficient.
 - Total space allocated to work files must be twice the sum of the input file size.

SORT & MERGE Utilities

SORT Control Statement

- Syntax

```
SORT    FIELDS=(position,length,format,sequence ....),EQUALS | NOEQUALS  
SORT    FIELDS=(position,length,sequence ..),FORMAT=format,EQUALS | NOEQUALS  
MERGE  FIELDS=(position,length,format,sequence ....)  
MERGE  FIELDS=(position,length,sequence ..),FORMAT=format  
INCLUDE COND=(field,comparison,value)  
OMIT    COND=(field,comparison,value)
```

- **FIELDS** Specifies control fields in the input records.
- **Position** Location of the first byte of the control field in the input record
- **Length** Length in bytes of the control field.

SORT & MERGE Utilities

- Format Two character code that identifies the format of the data in the control field.
 - CH Character
 - ZD Zoned Decimal
 - PD Packed Decimal
 - BI Unsigned Binary
 - AC ASCII Character
 - FL Signed Normalized Floating point
 - FI Signed Fixed point Binary
- Sequence Identifies the order of sorting
 - ✓ A Ascending sequence
 - ✓ D Descending sequence
- EQUALS/NOEQUALS
 - Specifies whether or not the order of the records in the input files is preserved in the output files when all the control fields are equal. Valid only for SORT statement.

SORT & MERGE Utilities

- Example
 - Input

Position 3-7	11-13	18-20
12345	AAA	12
12345	ABC	907
10110	322	430
10110	600	800
10110	600	730

- Control Statement

SORT FIELDS=(3,5,A,11,3,D,18,3,A),FORMAT=CH

- Output

10110	600	730
10110	600	800
10110	322	430
12345	ABC	907
12345	AAA	12

SORT & MERGE Utilities

Example (SORT) :

```
//SORT          EXEC PGM=SORT
//SORTIN        DD   DSN=USER1.TRAN.APRIL,DISP=SHR
//              DD   DSN=USER1.TRAN.MAY,DISP=SHR
//              DD   DSN=USER1.TRAN.JUNE,DISP=SHR
//SORTOUT       DD   DSN=USER1.TRAN.FINAL,DISP=(NEW,CATLG),
//                  VOL=SER=A102,SPACE=(TRK,(10,5)),
//                  DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SORTWK01      DD   VOL=SER=A102,SPACE=(CYL,(20,5))
//SORTWK02      DD   VOL=SER=A102,SPACE=(CYL,(20,5))
//SYSOUT        DD   SYSOUT=*
//SYSIN         DD   *
               SORT FIELDS=(41,6,CH,A,29,4,CH,A)
/*
*/
```

SORT & MERGE Utilities

Example (MERGE) :

```
//SORT          EXEC PGM=SORT
//SORTIN01      DD   DSN=USER1.TRAN.APRIL,DISP=SHR
//SORTIN02      DD   DSN=USER1.TRAN.MAY,DISP=SHR
//SORTIN03      DD   DSN=USER1.TRAN.JUNE,DISP=SHR
//SORTOUT       DD   DSN=USER1.TRAN.FINAL,DISP=(NEW,CATLG),
//                           VOL=SER=A102,SPACE=(TRK,(10,5)),
//                           DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSOUT        DD   SYSOUT=*
//SYSIN         DD   *
      MERGE FIELDS=(41,6,CH,A,29,4,CH,A)
/*
//
```

SORT & MERGE Utilities

Control statements examples :

```
SORT FIELDS=(41,6,A,29,4,D),FORMAT=CH  
SORT FIELDS=(41,6,CH,A),EQUALS  
SORT FIELDS=(1,5,CH,D),NOEQUALS
```

```
MERGE FIELDS=(41,6,A,29,4,D),FORMAT=CH
```

```
INCLUDE COND=(1,1,CH,EQ,C'A')
```

```
OMIT COND=(2,5,CH,EQ,C'BATCH')  
OMIT COND=(6,3,ZD,EQ,+89)
```

Common JCL ABEND codes

- **SOC4** - 1. Index exceeds the size of table
 2. Trying to use File Section variables without opening the file
- **S0CB** - Attempting to divide by 0 and not using ON SIZE ERROR
- **S322** - Time out error
- **SB37** - Insufficient disk space
- **SE37** - The error occurred during end of volume processing when an output operation was requested for a dataset. Ran out of directory space in a PDS output exceeded volume count
- **SD37** - A dataset opened for output used all the primary space, and no secondary space was requested. Either specify a larger primary quantity or add a secondary quantity to the request.
- **S806** - Load module not found
- **S913** - Security violation
- **U1056** - Program didn't close a file before ending

VSAM

VSAM

- VSAM stands for Virtual Storage Access Method.
- Access Method Services (AMS) is general purpose utility that provides a variety of services for VSAM files
- IDCAMS is the program used for this purpose
- Type of service (create, delete etc) is specified by control statements through //SYSIN
- The different types of organization for VSAM data sets are:
 - Entry Sequenced Data Set (ESDS)
 - Key Sequenced Data Set (KSDS)
 - Relative Record Data Set (RRDS)
 - Linear Data Set (LDS)

VSAM Building Blocks

- **Logical Record** - unit of information used to store data in a VSAM data set
 - **Control Interval** - VSAM uses CI to contain records.
 - **Control Area** - The CIs in a VSAM data set are grouped together into fixed length contiguous areas of direct storage called Control Areas
-
- CI is unit of data VSAM transfers between virtual and disk storage
 - CI concept is similar to blocking for Non-VSAM files
 - CI usually contains more than one logical record
 - The size of CI affects performance
 - Size of CI must be between 512 bytes to 32 KB (multiples of 512 for size up to 8 KB. Beyond 8 KB it should be in multiples of 2 KB)
 - Size of CA can be as small as a track (min-CA) or as large as a cylinder (max-CA)

VSAM Building Blocks

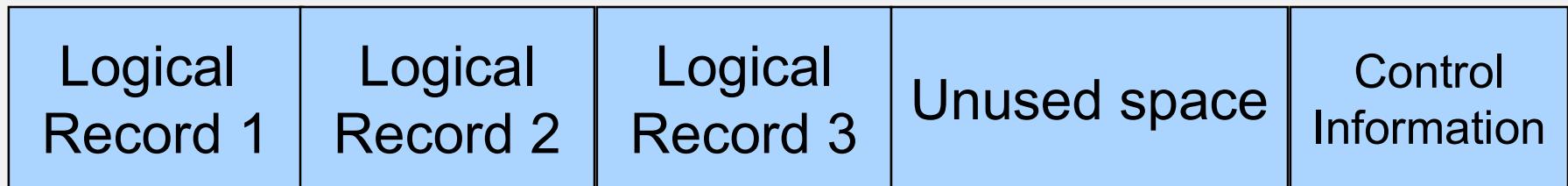
Control Area 1

Control interval 1	Logical Record 1	Logical Record 2	Logical Record 3	Logical Record 4
Control interval 2	Logical Record 5	Logical Record 6	Logical Record 7	Logical Record 8
Control interval 3	Logical Record 9	Logical Record 10	Logical Record 11	Logical Record 12
Control interval 4	Logical Record 13	Logical Record 14	Logical Record 15	Logical Record 16

Control Area 2

Control interval 5	Logical Record 17	Logical Record 18	Logical Record 19	Logical Record 20
Control interval 6	Logical Record 21	Logical Record 22	Logical Record 23	Logical Record 24
Control interval 7	Logical Record 25	Logical Record 26	Logical Record 27	Logical Record 28
Control interval 8	Logical Record 29	Logical Record 30	Logical Record 31	Logical Record 32

Control Interval structure

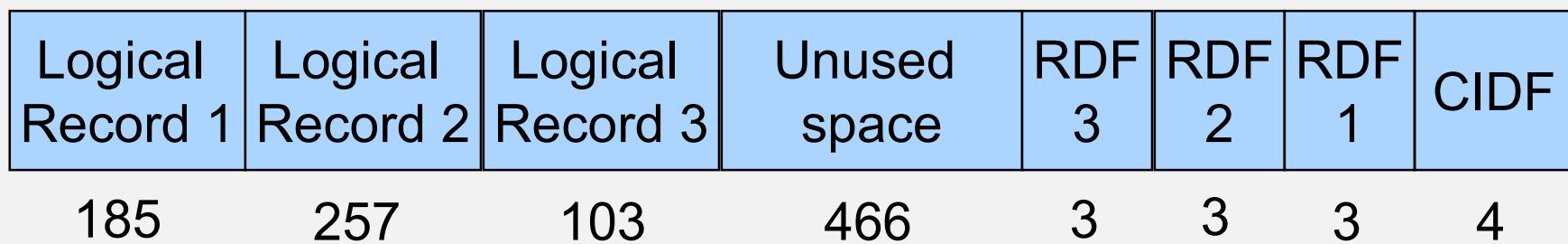


- Control information stores the details of each logical record
- New records added in Unused space

Control Interval structure

Variable length Records

- RDF (Record Descriptor Field)
 - ✓ contains information about each record
- CIDF (Control Interval descriptor Field)
 - ✓ Keeps track of free space
 - ✓ Contains flag that is set when CI is being updated



Control Interval structure

Fixed length Records

- Only two RDFs are used
- First RDF indicates length of the records
- Second RDF indicates number of records with the same length

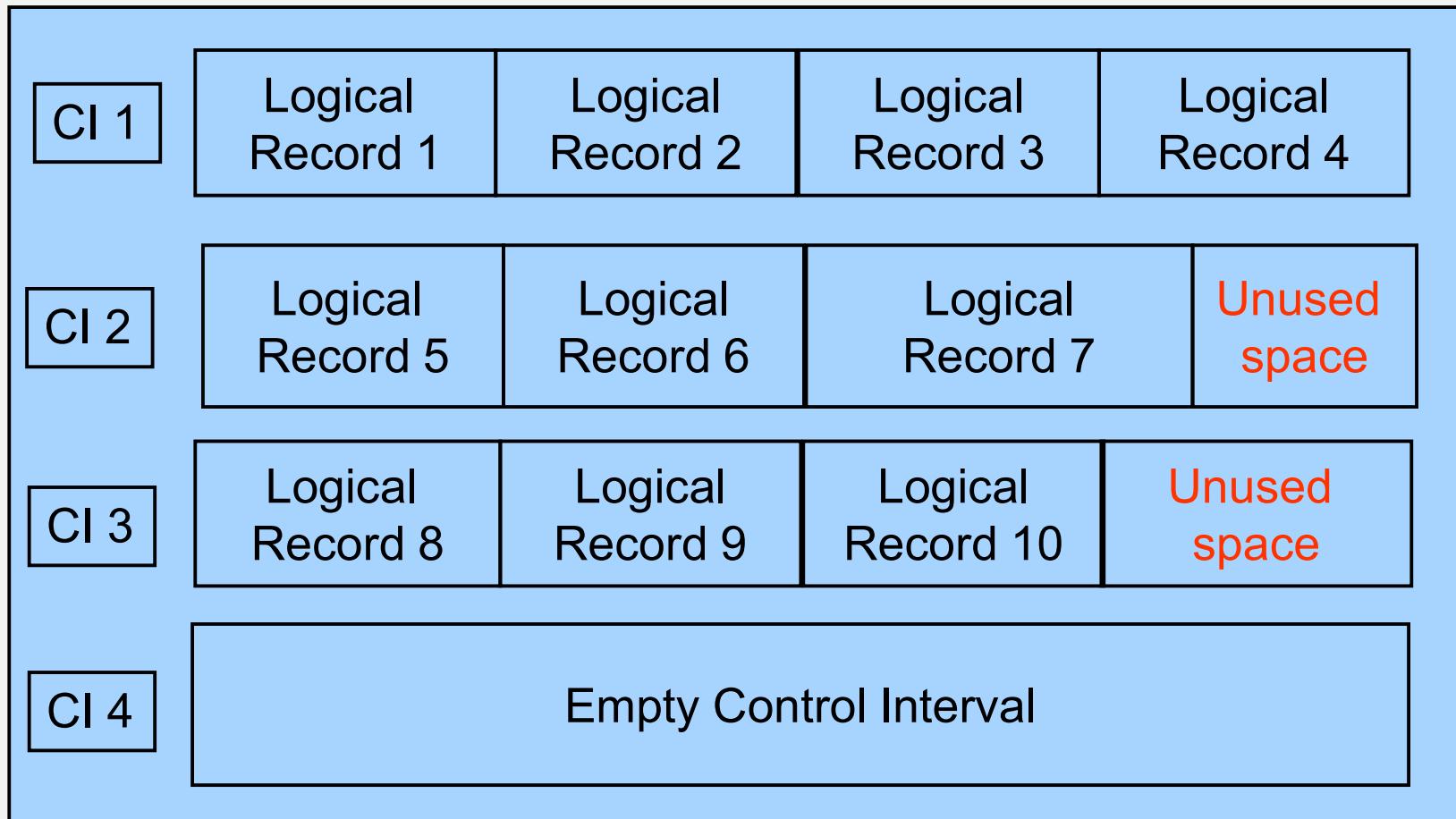
Logical Record 1	Logical Record 2	Logical Record 3	Unused space	RDF C	RDF L	CIDF
200	200	200	414	3	3	4

Entry Sequenced Data Set

- Much like a standard sequential file
- Records are stored in the order they are written
- Additions are always made at the end of the file
- Generally access of records is sequential
- At times record can be accessed directly using Relative Byte Address (RBA)

Entry Sequenced Data Set

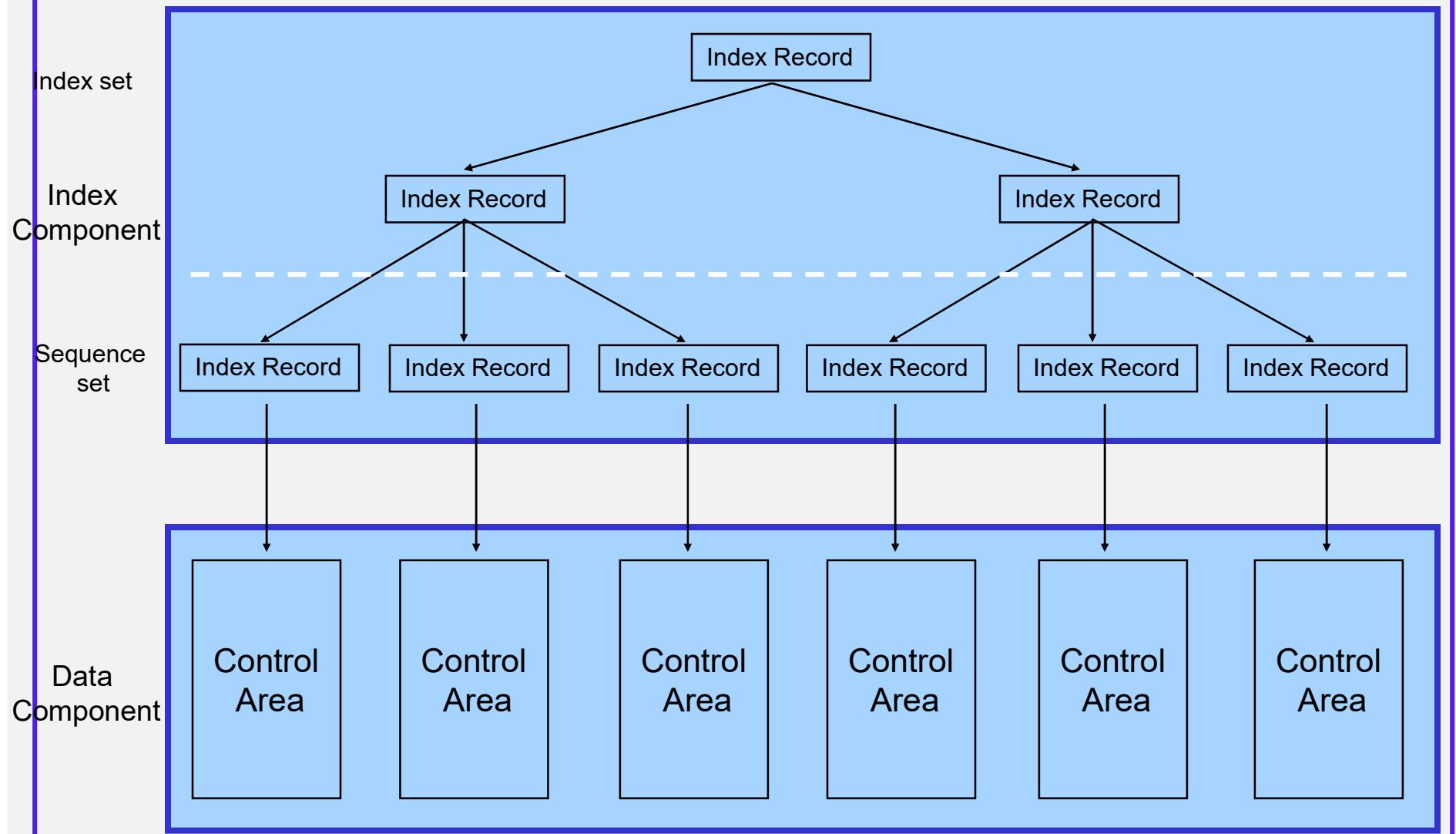
Control Area



Key Sequenced Data Set

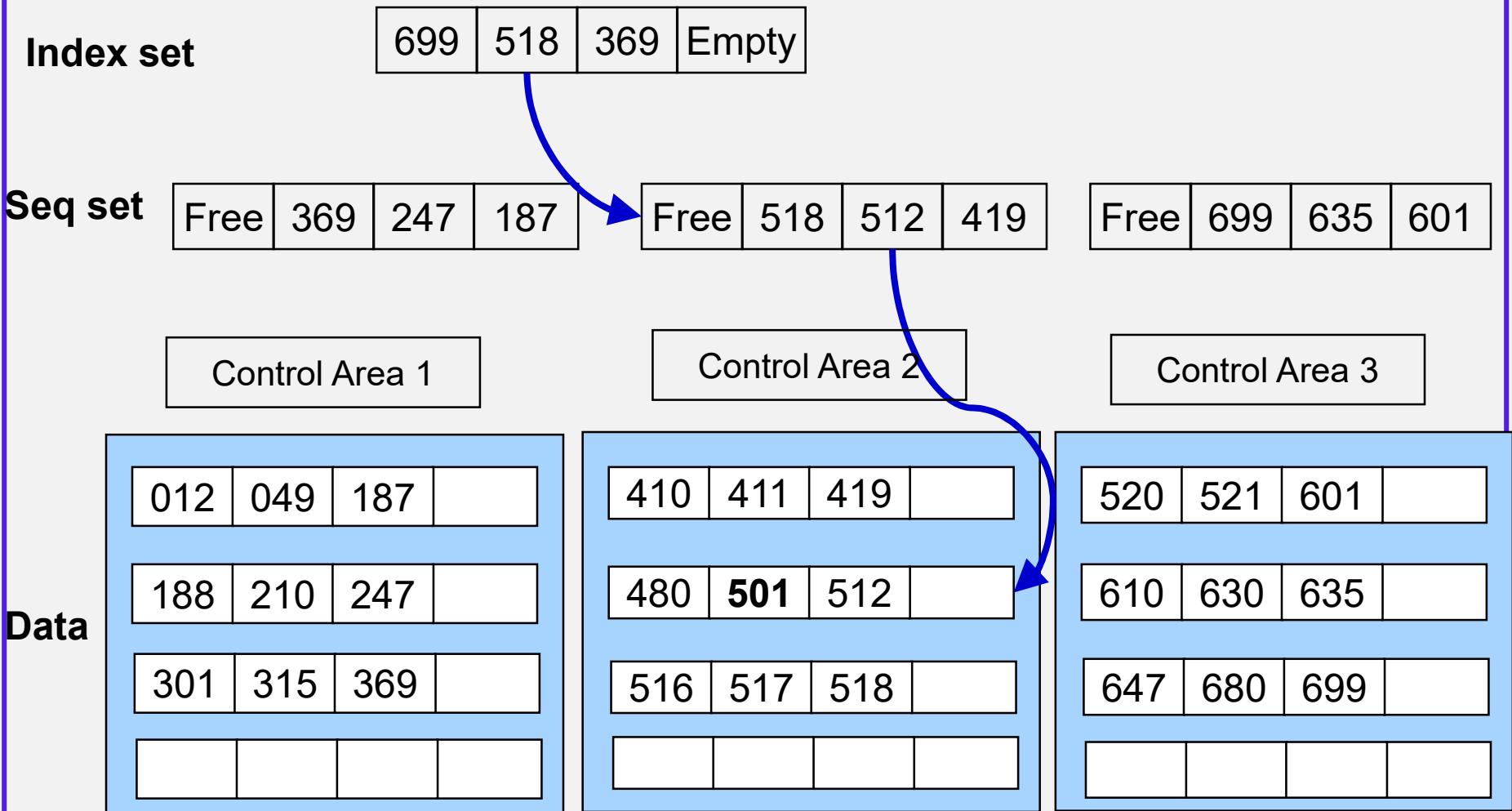
- KSDS cluster is a combination of data component and index component
- Records can be accessed sequentially and randomly
- In sequential access, records processed in the order of key values
- Index component has two parts – sequence set and index set
- Sequence set is at lowest level of the index and is searched to determine which CI of data component contains the record
- For each C A in the data component, there is one record in the sequence set
- Each sequence set record contains an index entry for each CI in the corresponding C A
- This index entry contains the highest key value stored in the C I
- Within the C I, logical records are stored in the order of the Key
- While defining KSDS cluster, free space can be reserved to accommodate new records

Key Sequenced Data Set



KSDS search

Search for Record with key value 501



Adding new records to KSDS

- New records inserted in its correct sequential location in the proper C I
- Other records are shifted as needed
- If no enough space, some records moved to one of the free C Is – this is called Control Interval Split
- If there is no free C I, Control Area Split occurs (allocate new C A and move about half C Is to the new C A)

Alternate Index

- Alternate index helps processing KSDS in an order other than the *Primary Key*
- Alternate index can be created for ESDS also
- Alternate key values can be duplicates in the Base Cluster
- Alternate Index itself is KSDS – each record in its data component contains one alternate key value and one or more corresponding primary key values
- If the Base cluster is ESDS, Alternate index record contains RBAs of the records for alternate key value
- Alternate index may or may not be updated when data in Base Cluster changes
- It becomes updatable only if specified explicitly while creation
- It is a good idea to rebuild the Alternate Indexes periodically or when needed

Alternate Index

Base Cluster

EmpNo	Name	DeptNo
1010	Suresh	10
1024	Sirisha	20
1078	Ashish	10
1110	Amit Rao	10
1123	Anil Kumar	20
1268	Supriya	30
1271	Subroto	40
1421	Sumit	20
1486	Naik	30

Alternate Index

DeptNo	EmpNo
10	1010 1078 1110
20	1024 1123 1421
30	1268 1486
40	1271

RRDS / LDS

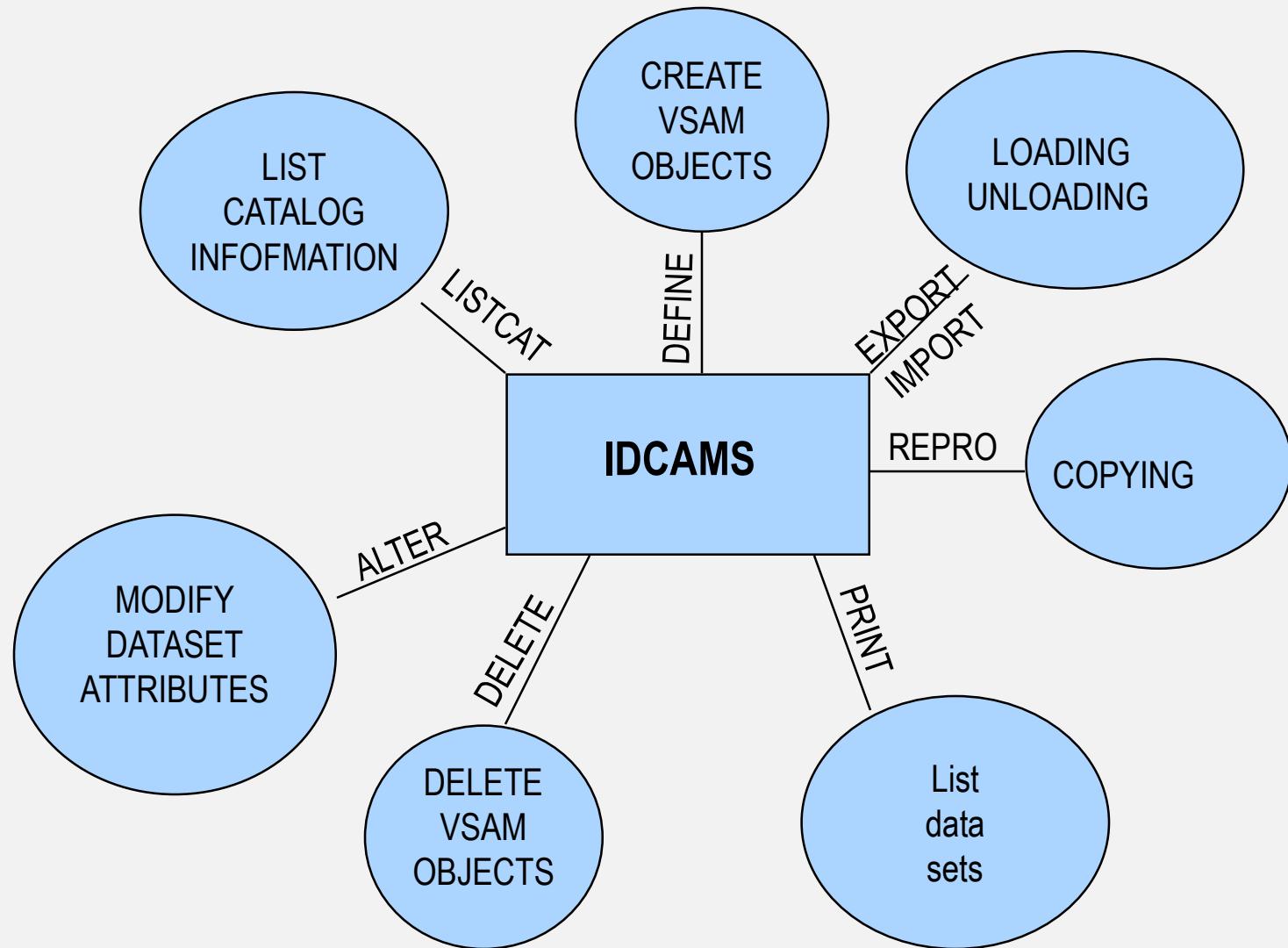
Relative Record Data Set

- RRDS consists of fixed length slots that are numbered
- Records can be accessed sequentially or randomly
- In sequential processing, records accessed in the order of the slots skipping the empty slots
- Random processing is based on each record's relative position in the file

Linear Data Set

- LDS has no record organization
- Data is written or read one C I at a time
- No control information added to the data
- Typically not used for application program data processing

AMS Commands



IDCAMS

- You can code AMS command from column 2 to 72
- Parameters can continue in multiple lines using continuation character - (hyphen)
- Parameter values to be enclosed in parentheses
 - eg ; RECORDS(500)
- Multiple parameters are separated using comma or space
 - eg : KEYS(6 2)
- Parentheses used to group multiple parameters
- comments can be inserted with /* comment */

Skeleton JCL for Invoking AMS

```
//JOBNAME      JOB (AACT), 'NAME'  
//STEPNAME     EXEC PGM=IDCAMS  
//SYSPRINT     DD SYSOUT=*  
//SYSIN  DD *  
    AMS functional commands  
//*
```

Define Cluster

- Used to define all three types of clusters – ESDS, KSDS, RRDS
- Parameters can be specified at CLUSTER, DATA, INDEX level
- If provided at cluster level, they apply to cluster as a whole
- They can be provided at DATA and INDEX levels also
- INDEX component is present only for KSDS
- Normally provided parameter for DATA & INDEX components is only NAME

Define Cluster

- Format :

```
DEFINE CLUSTER
  (NAME(entryname)
   {CYLINDERS(primary] secondary)] | 
    RECORDS(primary[ secondary)] | 
    TRACKS(primary[ secondary])})
  VOLUMES(volser[ volser...])
  CONTROLINTERVALSIZE(size)
  [INDEXED | NONINDEXED | NUMBERED]
  [KEYS(length offset)]
  [RECORDSIZE(average maximum)]
  [REUSE | NOREUSE]
  [TO(date) | FOR(days)]
  [DATA
    ([NAME(entryname)] ) ]
  [INDEX
    ( [NAME(entryname)] ) ] )
```

Define cluster

- Example for KSDS:

```
DEFINE CLUSTER (
    NAME(OZA122.EMPL) -
    VOLUMES(MVS801) -
    RECORDSIZE(80 80) -
    TRACKS(50 10) -
    KEYS(10 0) -
    INDEXED -
    CONTROLINTERVALSIZE(4096) -
    DATA (
        NAME(OZA122.EMPL.DATA) ) -
    INDEX (
        NAME(OZA122.EMPL.INDEX) -
    )
```

Define cluster

- Example for ESDS:

```
DEFINE CLUSTER (
    NAME(MVS801.STUDENT.FILE) -
    VOLUMES(MVS801) -
    RECORDSIZE(80 80) -
    TRACKS(50 10) -
    NONINDEXED -
    CONTROLINTERVALSIZE(4096) -
)
```

REPRO Command

- Used to copy contents of a dataset into another
- Can be used for both VSAM and Non-VSAM data sets
- Can be used for initial loading of records into VSAM dataset
- Example :

```
//IDCAMS EXEC PGM=IDCAMS,REGION=4096K
//VSAM      DD   DSN=OZA122.EMPL,DISP=SHR
//OUTFIL    DD   DSN=OZA122.EMP.OUT,
//                           DISP=(NEW,CATLG),VOL=SER=B00001,
//                           DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSPRINT DD   SYSOUT=A
//SYSIN     DD   *
               REPRO INFILE(VSAM)-
                           OUTFILE(OUTFIL)
/*
//
```

COBOL

Introduction

COBOL

- COmmon Business Oriented Language
- English-like language
- User friendly language
- Self Documenting
- Easy to learn/read/write/maintain
- Various flavors: COBOL VSE, COBOL II, COBOL/370

COBOL Syntax notation

- Words in uppercase are reserved words.
 - When underlined they must be present when the operation of which they are a part is used.
 - When they are not underlined, used for readability only and are optional. If used they must be spelt correctly.
- Words in mixed case represent names which will be devised by the programmer.
- When some options are enclosed in braces { } a choice must be made from the options within the braces.
- Whatever enclosed in square brackets [] indicates that it is optional and may be included or omitted as required.
- The ellipsis symbol ‘...’ indicates that the preceding syntax element may be repeated at the programmer’s discretion.

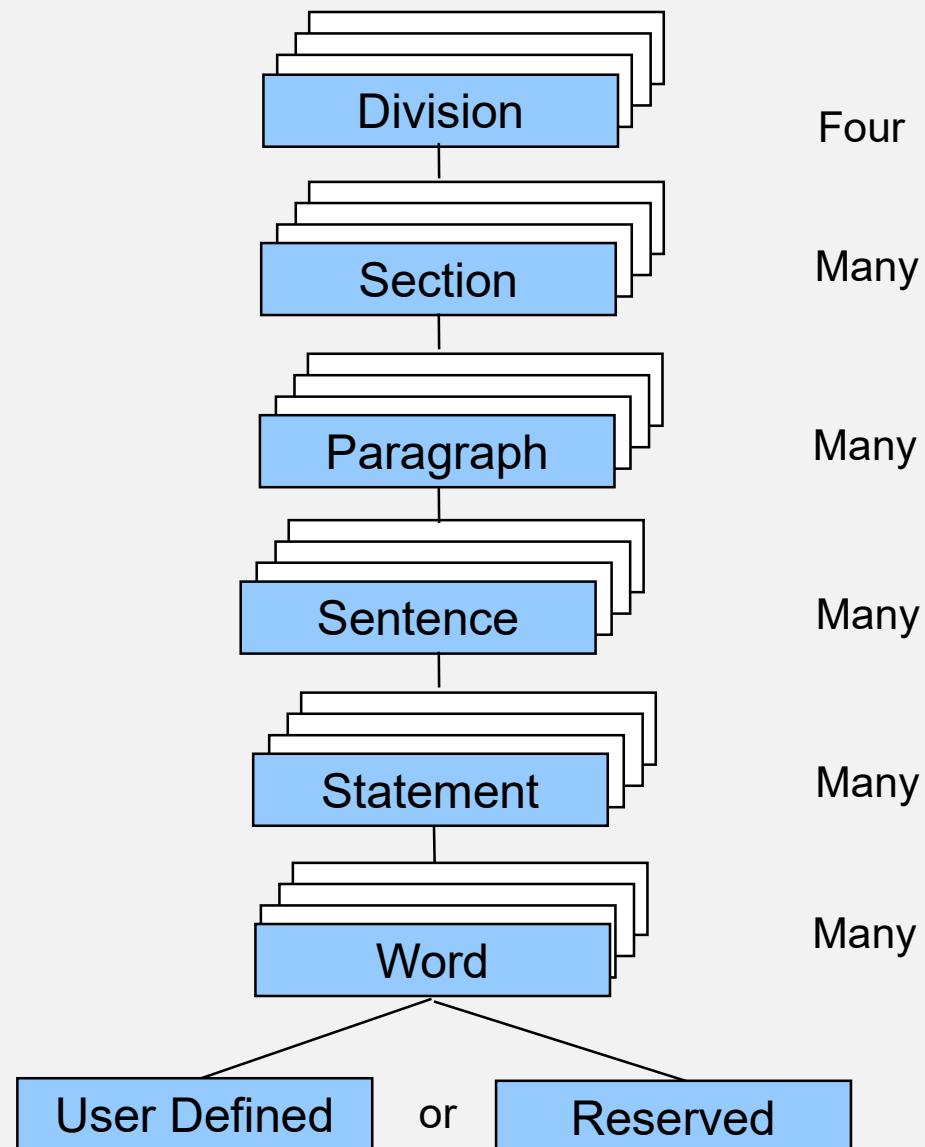
COBOL Syntax notation

Examples :

```
SELECT FileName ASSIGN TO ExternalReference  
[ ORGANIZATION IS SEQUENTIAL ]
```

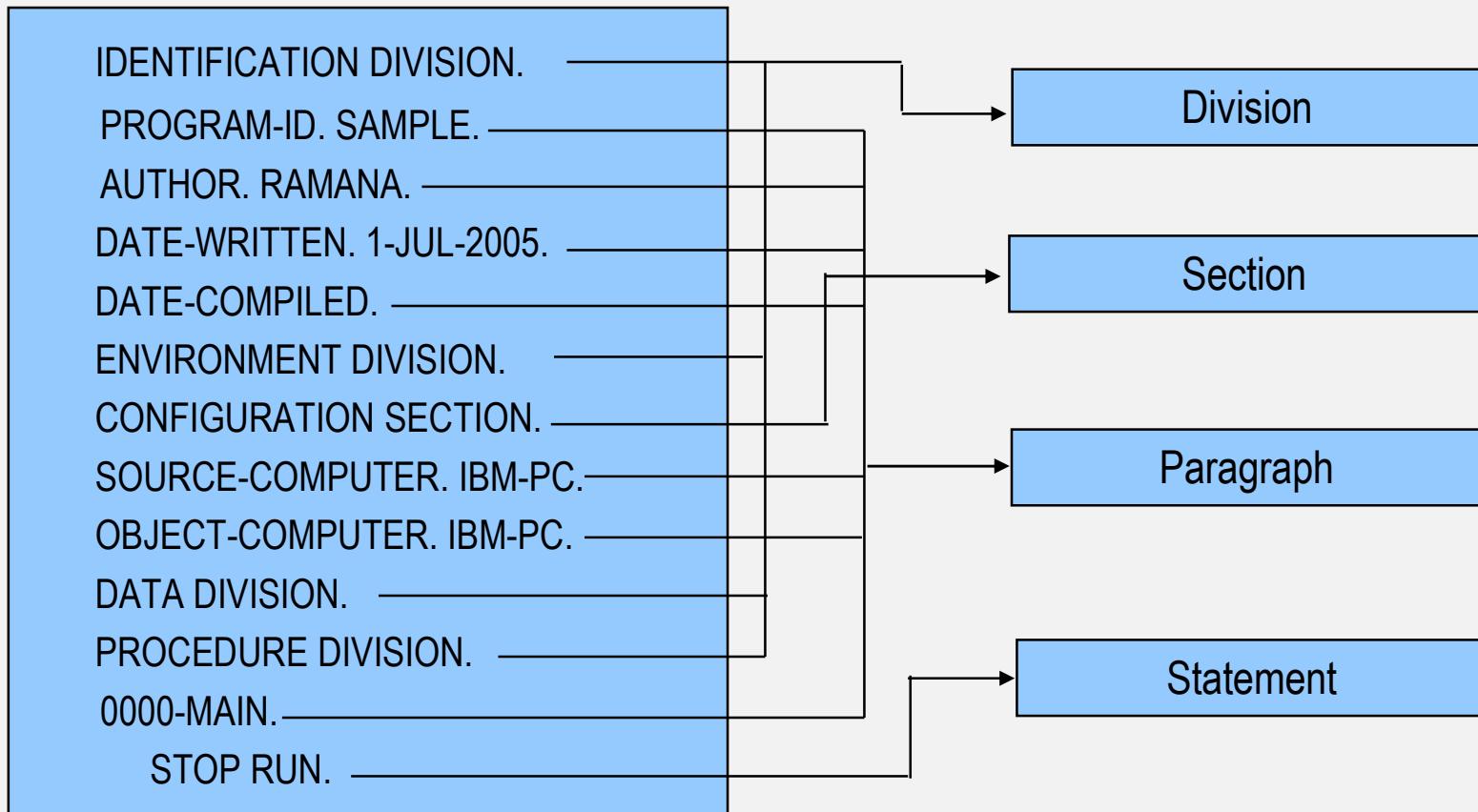
```
MOVE { Identifier } TO Identifier ...  
      { Literal }
```

COBOL Program Structure



COBOL Program Structure

COBOL Program Layout



COBOL Program Structure

COBOL programs are normally coded in 80 column sheets using following rules.

1	6	7	8	11	12	72	73	80
---	---	---	---	----	----	----	----	----

Column Range	Comments
1-6	Sequence Number (optional)
7	Continuation Character '-' or Comment Character '*' or Page Break Character '/'.
8-11	Area A - DIVISION, SECTION, paragraph names, FD and level 01, 77 entries
12-72	Area B - Statements and Sentences
73 - 80	Identification Field (optional)

COBOL Program Structure

Continuation

- Any sentence, entry, clause, or phrase that requires more than one line can be continued in Area B of the next line
- Area A of a continuation line must be blank
- To continue a non numeric literal (string) ---
 - the literal should continue till column 72 without a closing quotation mark
 - all spaces at the end of the continued line (through column 72) are part of the literal
 - Continuation line must contain a hyphen in the indicator area (column 7)
 - In the next line, first nonblank character must be a quotation mark and the literal continues with the character following the quotation mark

COBOL Program Structure

Divisions

- **Identification Division**
 - The Identification Division names the program and, optionally, documents the date the program was written, the compilation date and other pertinent information.
- **Environment Division**
 - This division contains machine dependent details such as computer (s) used and peripheral devices.
 - Also provides link to the data files used by the program
- **Data Division**
 - The Data Division defines the nature and characteristics of all data the program is to process. This includes both Input/Output data and data used for internal processing.
- **Procedure Division**
 - The Procedure Division consists of executable statements that process the data in the manner the programmer defines. The statements are executed in top down sequence.

COBOL Program Structure

Clauses

- Written in Environment and Data Divisions – this specifies an attribute of an entry.
A series of clauses, ending with period, is defined as an entry.

Statements

- Written in Procedure Division – specify an action to be taken by the object program. A series of statements, ending with a period, is defined as a sentence

User-Defined Words

- Used to define variables, paragraph names etc
- Max. Length 30 Chars
- Must contain only Digits, Letters and Hyphens
- Must not be a Reserved Word
- Include at least one Letter
- Hyphens are embedded (not allowed at the start and end)
- **Examples :** emp-name, A5-123W, NO-MORE-RECORDS

COBOL Program Structure

Reserved Words

- Reserved for COBOL use
- Cannot be redefined by the programmer
- **Example:**
 - RECORD CONTAINS 100 TO 300 CHARACTERS DEPENDING ON file-len
 - **Keywords (required)** : RECORD, CONTAINS, DEPENDING
 - **Keywords (optional)** : TO, CHARACTERS ON
 - **User defined Words** : file-len

Literals

- Are Constants
- Can be Numeric (Integer or Floating point) or Non-Numeric
- **Examples:** 123.345, "COBOL PRG"

COBOL Program Structure

- **Numeric Literals**
 - Contains minimum 1 digit and can have maximum 18 digits
 - Sign, if used, must be left-most character and must not be more than one
 - Decimal, if used, may not be right-most character and must not be more than one
 - Can be specified in exponential format also
 - Must fall between 0.54E-78 and 0.72E+76.
 - **Example:** 354, 35.67, 99.99E+45 (equivalent to 99.99 X 10 to the power of 45)
- **Non-numeric Literals**
 - Any character in the Character Set allowed
 - Must be enclosed in Quotes
 - Numeric Literal in Quotes is Non-Numeric
 - Maximum Length is 160 Characters

Figurative Constants

COBOL provides its own, special constants called Figurative Constants.

SPACE or SPACES	= □
ZERO or ZEROS or ZEROES	= 0
QUOTE or QUOTES	= "
HIGH-VALUE or HIGH-VALUES	= Max Value
LOW-VALUE or LOW-VALUES	= Min Value
ALL <i>literal</i>	= Fill With Literal

IDENTIFICATION DIVISION

- Identification Division supplies identifying information about the program.
- Must be the first division in any COBOL Program
- Must begin with the words IDENTIFICATION DIVISION followed by a period.
- It supplies the information about the program to others who may read or use the program

IDENTIFICATION DIVISION.

PROGRAM-ID. Program name.

[AUTHOR. Programmer name.]

[DATE-WRITTEN. Date when program was coded.]

[DATE-COMPILED. Date when program was compiled.]

IDENTIFICATION DIVISION

PROGRAM-ID. Program-name.

- Used to specify the program name.
- Use names of eight characters or less, letter and digits only, because such names are accepted on all systems.
- This is the only paragraph that is mandatory

AUTHOR. author-name.

- Used to specify the programmer's name.

DATE-WRITTEN. Date.

- Specify the date the program was coded.

DATE-COMPILED. Date.

- Can be coded with an actual date. But if it is coded without a date entry, the compiler itself will automatically fill in the actual date of compilation

DATA DIVISION

- The DATA DIVISION is used to describe most of the data that a program processes.
- The DATA DIVISION is divided mainly into two sections
 - FILE SECTION
 - WORKING-STORAGE SECTION
- The FILE SECTION is used to describe most of the data that is sent to, or comes from, the computer's peripherals.
- The WORKING-STORAGE SECTION is used to describe the general variables used in the program
- Other sections used for special purposes
 - Linkage Section : for inter program communication
 - Report Section : for report generation

DATA DIVISION Syntax

The DATA DIVISION has the following structure

DATA DIVISION.
[FILE SECTION.
 File Section entries.]
[WORKING-STORAGE SECTION.
 WS entries.]

IDENTIFICATION DIVISION.
PROGRAM-ID. TestPgm.
AUTHOR. Ramana Reddy.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 Num1 PIC 9 VALUE ZEROS.
01 Num2 PIC 9 VALUE ZEROS.
01 Result PIC 99 VALUE ZEROS.

PROCEDURE DIVISION

- The PROCEDURE DIVISION is where all the data described in the DATA DIVISION is processed and produced. It is here that the programmer describes the algorithm.
- The PROCEDURE DIVISION is hierarchical in structure and consists of Sections, Paragraphs, Sentences and Statements.
- Only the Section is optional. There must be at least one paragraph, sentence and statement in the PROCEDURE DIVISION.
- In the PROCEDURE DIVISION, paragraph and section names are chosen by the programmer. The names used should reflect the processing being done in the paragraph or section.

Sections

- A section is a block of code made up of one or more paragraphs.
- A section begins with the section name and ends where the next section name is encountered or where the program text ends.
- A section name consists of a name devised by the programmer or defined by the language followed by the word SECTION followed by a full stop.
 - SelectUlsterRecords SECTION.
 - FILE SECTION.

Paragraphs

- Each section consists of one or more paragraphs.
- A paragraph is a block of code made up of one or more sentences.
- A paragraph begins with the paragraph name and ends with the next paragraph or section name or the end of the program text.
- The paragraph name consists of a name devised by the programmer or defined by the language followed by a full stop.
 - PrintFinalTotals.
 - PROGRAM-ID.

Sentences and Statements

- A paragraph consists of one or more sentences.
- A sentence consists of one or more statements and is terminated by a period (full stop)

```
MOVE .21 TO VatRate  
COMPUTE VatAmount = ProductCost * VatRate.
```

```
DISPLAY "Enter name " WITH NO ADVANCING  
ACCEPT StudentName  
DISPLAY "Name entered was " StudentName.
```

- A statement consists of a COBOL verb and an operand or operands

```
SUBTRACT Tax FROM GrossPay GIVING NetPay  
READ StudentFile  
    AT END SET EndOfFile TO TRUE  
END-READ
```

A Full COBOL program

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. TestPgm.  
AUTHOR. Ramana Reddy.  
  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 Num1          PIC 99 VALUE ZEROS.  
01 Num2          PIC 99 VALUE ZEROS.  
01 Result        PIC 99 VALUE ZEROS.  
  
PROCEDURE DIVISION.  
CalculateResult.  
    ACCEPT Num1.  
    ACCEPT Num2.  
    MULTIPLY Num1 BY Num2 GIVING Result.  
    DISPLAY "Result is = ", Result.  
    STOP RUN.
```

The minimum COBOL program

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. SmallProgram.  
  
PROCEDURE DIVISION.  
DisplayPrompt.  
    DISPLAY "Hello World".  
    STOP RUN.
```

Quick Review of “Data Typing”

- In “typed” languages simply specifying the type of a data item provides quite a lot of information about it.
- The type usually determines the range of values the data item can store.
- For instance a short data type in C language can store values from -32,768 to 32,767
- From the type of the item the compiler can establish how much memory to set aside for storing its values

COBOL Data Types

- COBOL is not a “typed” language
- For the time being we will focus on just two data types
 - numeric
 - text or string
- Data type is important because it determines the operations which are valid on the type
- COBOL uses what could be described as a “declaration by example” strategy
- In effect, the programmer provides the system with an example, or template, or PICTURE of what the data item looks like
- From the “picture”, the system derives the information necessary to allocate it

COBOL ‘PICTURE’ Clause symbols

- To create the required ‘picture’ a set of symbols are used.
- The following symbols are used frequently in picture clauses
 - **X** (the character X) is used to indicate the occurrence of any character from the character set at the corresponding position in the picture
 - **A** (the character A) is used to indicate the occurrence of any letter (alphabet) at the corresponding position in the picture
 - **9** (the digit nine) is used to indicate the occurrence of a digit at the corresponding position in the picture. This is generally used to describe numeric data
 - **V** (the character V) is used to indicate position of the decimal point in a numeric value! It is often referred to as the “assumed decimal point” character.
 - **S** (the character S) indicates the presence of a sign and can only appear at the beginning of a picture. This is also referred as “assumed sign”.

COBOL ‘PICTURE’ Clauses

- **Some examples**

PICTURE 999 a three digit (+ve only) integer

PICTURE S999 a three digit (+ve/-ve) integer

PICTURE XXXX a four character text item or string

PICTURE 99V99 a +ve ‘real’ in the range 0 to 99.99

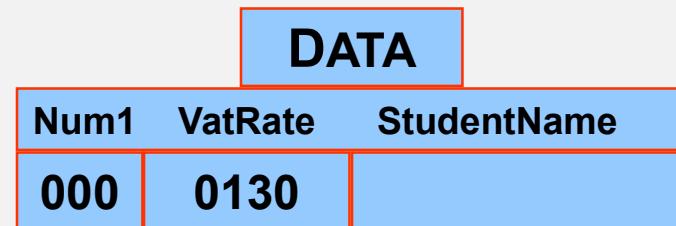
PICTURE S9V9 a +ve/-ve ‘real’ in the range ?

- **Abbreviation PIC can also be used.**
- **Recurring symbols can be specified using a ‘repeat’ factor inside round brackets**
 - PIC 9(6) is equivalent to PICTURE 999999
- **Numeric values can have a maximum of 18 (eighteen) digits**
- **The limit on string values is usually system-dependent.**

Declaring DATA in COBOL

- In COBOL, a variable declaration consists of a line which has
 - A level number.
 - A data-name or identifier.
 - A PICTURE clause.
- VALUE clause can be used optionally to specify the initial value stored in the variable (data name)

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 Num1          PIC 999    VALUE ZEROS.  
01 VatRate       PIC 99V99  VALUE 1.3.  
01 StudentName   PIC X(10)  VALUE SPACES.
```



LEVEL Numbers express DATA hierarchy

- Level numbers are used to decompose a structure into it's constituent parts.
- In this hierarchical structure the higher the level number, the lower the item is in the hierarchy. At the lowest level the data is completely atomic.
- The level numbers 01 through 49 are used for describing data hierarchy
- There are other level numbers such as 66, 77 and 88 which are for special purpose
- In a hierarchical data description what is important is the relationship of the level numbers to one another, not the actual level numbers used.

```
01 StudentDetails.  
  02 StudentName.  
    03 Surname      PIC X(8).  
    03 Initials     PIC XX.  
  02 StudentId     PIC 9(7).  
  02 CourseCode    PIC X(4).  
  02 Grant         PIC 9(4).  
  02 Gender        PIC X.
```

```
01 StudentDetails.  
  05 StudentName.  
    10 Surname      PIC X(8).  
    10 Initials     PIC XX.  
  05 StudentId     PIC 9(7).  
  05 CourseCode    PIC X(4).  
  05 Grant         PIC 9(4).  
  05 Gender        PIC X.
```

Group and Elementary items.

- The term “group item” is used to describe a data item which is further subdivided.
- A Group item is declared using a level number and a data name. It cannot have a picture clause.
- Where a group item is the highest item in a data hierarchy it is referred to as a record and uses the level number 01.
- The term “elementary item” is used to describe data items which are atomic; that is, not further subdivided.
- An elementary item should always have PICTURE clause
- The size of a group item is the sum of the sizes of its subordinates and its type is always assumed to be alphanumeric
- Every group or elementary item declaration must be followed by a period

LEVEL Number 77

- Level number 77 is used for independent elementary items (not part of any group) in Working-Storage section
- This was used in older versions of COBOL where 01 was not allowed for elementary items
- In later versions (COBOL-74 onwards) level number 01 is allowed for elementary items
- Due to this reason, level number 77 can be treated as obsolete

Assignment in COBOL

- In COBOL, assignment is achieved using the MOVE verb.

```
MOVE {Identifier  
      Literal} TO Identifier ...
```

- The MOVE copies data from the source identifier or literal to one or more destination identifiers.
- The source and destination identifiers can be group or elementary data items.
- When the destination item is alphanumeric or alphabetic (PIC X or A) data is copied into the destination area from left to right with space filling or truncation on the right.
- When the destination item is numeric, or edited numeric, then data is aligned along the decimal point with zero filling or truncation as necessary.
- When data is MOVED into an item the contents of the item are completely replaced. If the source data is too small to fill the destination item entirely the remaining area is zero or space filled.

MOVE examples - Non numeric data

01 Surname PIC X(8) value 'SUKUMAR' .



MOVE "ANIL" TO Surname .



MOVE "SELVARAJAN" TO Surname .



MOVE examples - Numeric data

01 Quantity

PIC 999.

01 Price

PIC 999V99.

MOVE 1234 TO Quantity

2	3	4
---	---	---

MOVE 12.4 TO Quantity

0	1	2
---	---	---

MOVE 154 TO Price

1	5	4	0	0
			↑	•

MOVE 3552.753 TO Price

5	5	2	7	5
			↑	•

The DISPLAY Verb

- From time to time it may be useful to display messages and data values on the screen
- A simple DISPLAY statement can be used to achieve this
- A single DISPLAY can be used to display several data items or literals or any combination of these
- The WITH NO ADVANCING clause suppresses the carriage return/line feed
- Format :

```
DISPLAY { Identifier  
          Literal } ... [ WITH NO ADVANCING ]
```

The ACCEPT verb

- ACCEPT has two formats
- First format used to accept values from standard input and store in a variable
- Second format used to get the system date / time and store in given variable in several formats

- Format 1:

ACCEPT Identifier

- Format 2:

ACCEPT Identifier FROM

{ DATE
DAY
DAY-OF-WEEK
TIME**}**

The ACCEPT verb

Date / Time formats

DATE	PIC 9(6)	YYMMDD
DAY	PIC 9(5) .	YYDDD
DAY-OF-WEEK	PIC 9.	D (1=Monday)
TIME	PIC 9(8) .	HHMMSSss

DISPLAY / ACCEPT example

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 StudentDetails.  
  02 StudentName      PIC X(10).  
  02 StudentId        PIC 9(7).  
  02 CourseCode       PIC X(4).  
  02 Grant            PIC 9(4).  
  02 Gender           PIC X.  
01 CurrentDate.  
  02 CurrentYear     PIC 99.  
  02 CurrentMonth    PIC 99.  
  02 CurrentDay      PIC 99.  
01 DayOfYear.  
  02 FILLER          PIC 99.  
  02 YearDay         PIC 9(3).
```

Run of Accept and Display program

```
Enter student details using template below  
NNNNNNNNNNSSSSSSSSCCCCGGGGS  
SRINIVAS K59476532LM511245M  
Name is SRINIVAS K  
Date is 24 05 08  
Today is day 145 of the year
```

```
PROCEDURE DIVISION.
```

```
Begin.
```

```
  DISPLAY "Enter student details using template below".  
  DISPLAY "NNNNNNNNNNSSSSSSSSCCCCGGGGS".  
  ACCEPT StudentDetails.  
  ACCEPT CurrentDate FROM DATE.  
  ACCEPT DayOfYear FROM DAY.  
  DISPLAY "Name is ", StudentName.  
  DISPLAY "Date is " CurrentDay SPACE CurrentMonth SPACE CurrentYear.  
  DISPLAY "Today is day " YearDay " of the year".  
STOP RUN.
```

Qualifying data names

- It is possible to have same data names in different places
- While referring a data name in procedure division, it should be properly qualified
- Example :

```
01 INDATA.  
  02 EmpNo      PIC X(4).  
  02 NAME       pic X(10)  
  02 DOB        PIC X(8).  
  
01 OUTDATA.  
  02 EmpNo      PIC X(4).  
  02 NAME       pic X(10)  
  02 DOB        PIC XX/XX/XX.  
  
PROCEDURE DIVISION.  
-----  
MOVE EMPNO OF INDATA TO EMPNO OF OUTDATA  
MOVE NAME OF INDATA TO NAME IN OUTDATA  
MOVE DOB IN INDATA TO DOB IN OUTDATA  
  
-----  
DISPLAY NAME OF OUTDATA
```

STOP statement

- Format

```
STOP { RUN }  
          { literal }
```

- Halts execution of the program :
 - Permanently (RUN option)
 - ✓ Control is returned to the system
 - Temporarily (literal option)
 - ✓ literal communicated to operator and execution suspended.
 - ✓ Program execution is resumed only after operator intervention.
- STOP RUN statement closes all files defined in any of the programs comprising the run unit.

Reference Modification

- Reference Modification allows you to treat a numeric (PIC 9) or alphanumeric (PIC X) data-item as if it were an array of characters.
- To access sub-strings using Reference Modification you must specify
 - the name of the data-item
 - the start character position of the sub-string
 - the number of characters in the sub-string
- Syntax

DataName (*StartPos* [:*SubStrLength*])

- *StartPos* is the character position of the first character in the sub-string and *SubStrLength* is number of characters in the substring
- If *SubStrLength* is omitted, the substring from *StartPos* to the end of the string is assumed.
- Reference Modification may be used almost anywhere an alphanumeric data-item is permitted

Reference Modification - examples

- Working-storage section for the examples

```
01 CAT-TYPE  PIC X(15) VALUE 'CALICO'.
01 DOG-TYPE  PIC X(15) VALUE 'SCHNAUZER'.
01 CAT-ABRV   PIC X(5).
01 DOG-END    PIC X(10).
```

- Example 1

```
MOVE CAT-TYPE(1 : 5) TO CAT-ABRV
```

This will move "CALIC"
to CAT-ABRV

- Example 2

```
MOVE DOG-TYPE(5:) TO DOG-END
```

This will move "AUZER"
to DOG-END

- Example 3

```
IF CAT-TYPE (2 : 3) = 'ABC'
```

IF "ALI" = "ABC"

Processing Files

File Basics

File is a collection of records

File Types

- Disk, Tape and Printer Files
- Disk Files can have all the Organizations - Sequenced, Indexed and Relative. They can be implemented using either QSAM or VSAM.
- Printer Files can have only Sequential Organization and can be opened only in Output Mode. Special controls like skipping lines, pages are available.

File Characteristics

- Record Size and Block Size
- Block contains one or more records
- Record contains one or more fields

How files are processed.

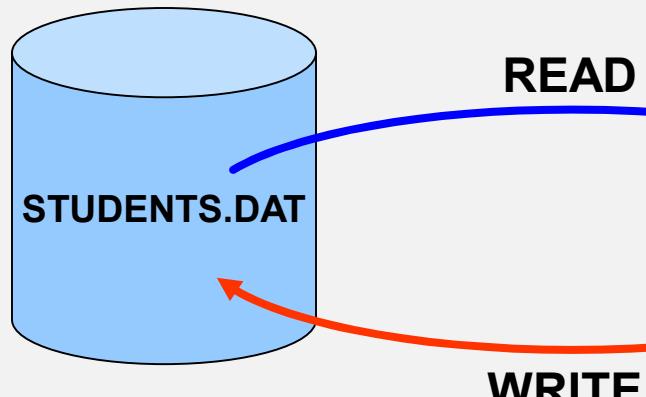
- Files are repositories of data that reside on backing storage (hard disk or magnetic tape).
- A file may consist of hundreds, thousands or even millions of records.
- Suppose we want to keep information about all the TV license holders in the country and each record is about 150 characters/bytes long. If we estimate the number of licenses at 1 million this gives us a size for the file of $150 \times 1,000,000 = 150$ megabytes.
- If we want to process a file of this size we cannot do it by loading the whole file into the computer's memory at one time.
- Files are processed by reading them into the computer's memory one record at a time.

Record Buffers

- To process a file, records are read from the file into the computer's memory one record at a time.
- The computer uses the programmer's description of the record (i.e. the record template) to set aside sufficient memory to store one instance of the record.
- Memory allocated for storing a record is usually called a "record buffer"
- The record buffer is the only connection between the program and the records in the file.

READ / WRITE operations on a File

READ transfers one record from the file to the Record Buffer



Program

IDENTIFICATION DIVISION.
etc.
ENVIRONMENT DIVISION.
etc.
DATA DIVISION.
FILE SECTION.

**RecordBuffer
Declaration**

WRITE transfers one record from Record Buffer to the file

Implications of 'Buffers'

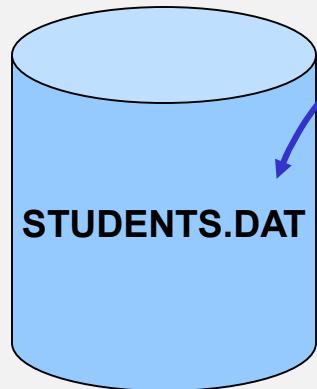
- If your program processes more than one file you will have to describe a record buffer for each file
- To process all the records in an INPUT file, each record instance must be copied (read) from the file into the record buffer when required
- To create an OUTPUT file containing data records, each record must be placed in the record buffer and then transferred (written) to the file
- To transfer a record from an input file to an output file the process is --
 - read the record into the input record buffer
 - transfer it to the output record buffer using MOVE statement
 - write the data to the output file from the output record buffer
 - repeat this process till end of input file

Describing the record buffer in COBOL

```
DATA DIVISION.  
FILE SECTION.  
FD StudentFile.  
01 StudentDetails.  
    02 StudentId          PIC 9(7) .  
    02 StudentName.  
        03 Surname         PIC X(8) .  
        03 Initials        PIC XX .  
    02 DateOfBirth.  
        03 YOBirth          PIC 9(2) .  
        03 MOBirth          PIC 9(2) .  
        03 DOBirth          PIC 9(2) .  
    02 CourseCode        PIC X(4) .  
    02 Grant             PIC 9(4) .  
    02 Gender            PIC X .
```

- The record type / template / buffer of every file used in a program must be described in the FILE SECTION by means of an FD (file description) entry
- The FD entry consists of the letters FD and an internal file name
- Internal file name is connected to the external name in Environment Division

The Select and Assign Clause.



```
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
  SELECT StudentFile  
    ASSIGN TO "STUDENTS.DAT".  
  
DATA DIVISION.  
FILE SECTION.  
FD StudentFile.  
01 StudentDetails.  
  02 StudentId          PIC 9(7) .  
  02 StudentName         PIC X(8) .  
  03 Surname             PIC XX.  
  03 Initials            PIC X(8) .  
  02 DateOfBirth          PIC X(4) .  
  02 CourseCode           PIC 9(4) .  
  02 Grant                PIC X .  
  02 Gender               PIC X .
```

The internal file name used in the FD entry is connected to an external file (on disk or tape) by means of the Select and Assign clause.

File Organization

COBOL supports several types of file organizations

- **Sequential Files**

- Records can be accessed in the order in which they appear in the file
- Records can be updated
- Records can be appended but not inserted
- Records cannot be deleted
- **Access Mode** : Sequential Only

- **Indexed Files**

- Random Access of Records using a Record Key
- Records can be Inserted, Updated and Deleted
- Indexed Access is faster than Sequential Access
- Unique or Non-Unique Indexes can be created
- **Access Modes** : Sequential, Random, Dynamic

File Organization

- **Relative Files**

- Also called Direct Access Files
- Each record has a unique address and is identified by its Relative Record Number in the file
- Records can be Accessed Randomly using their Relative Record Numbers
- **Access Modes** : Sequential, Random, Dynamic

File Access Modes

ACCESS →	SEQUENTIAL	RANDOM	DYNAMIC
ORGANIZATION			
SEQUENTIAL	Order of write	Invalid	Invalid
RELATIVE	Ascending REC. NO.	Value of RELATIVE KEY	SEQUENTIAL or RANDOM
INDEXED	Ascending KEY Value	Value of RECORD KEY	SEQUENTIAL or RANDOM

Select and Assign Syntax.

```
SELECT FileName ASSIGN TO External Reference  
[ ORGANIZATION IS SEQUENTIAL ]
```

- Other types of Organization (INDEXED / RELATIVE) will be discussed later
- ORGANIZATION clause is optional.
- If omitted, SEQUENTIAL is assumed by default

COBOL file handling Verbs

OPEN

Before a program can access the data in an input file or place data in an output file you must make the file available to the program by OPENing it.

READ

The READ copies a record occurrence/instance from the file and places it in the record buffer.

WRITE

The WRITE copies the record in the record buffer to the file.

CLOSE

You must ensure that (before terminating) your program closes all the files it has opened. Failure to do so may result in data not being written to the file or users being prevented from accessing the file.

OPEN verb

```
OPEN {  
    {  
        INPUT  
        OUTPUT  
        EXTEND  
    } InternalFileName  
}
```

- When you open a file you have to indicate to the system what how you want to use it (e.g. INPUT, OUTPUT, EXTEND) so that the system can manage the file correctly
- INPUT mode for input files which should be already existing
- OUTPUT mode for creating new file. If the file with the same name already exists, it is deleted
- EXTEND mode for adding records at the end of the file
- Opening a file does not transfer any data to the record buffer
- It simply provides access.

CLOSE verb

```
CLOSE fileName ...
```

- Closes files
- READ / WRITE operations not allowed after the file is closed

The READ verb

- Once the system has opened a file and made it available to the program it is the programmer's responsibility to process it correctly.
- Remember, the file record buffer is our only connection with the file and it is only able to store a single record at a time.
- To process all the records in the file we have to transfer them, one record at a time, from the file to the buffer.
- COBOL provides the READ verb for this purpose.

READ verb syntax

```
READ InternalFilename [NEXT] RECORD
    [INTO Identifier]
    AT END StatementBlock
END - READ
```

- The **InternalFilename** specified must be a file that has been OPENed for INPUT.
- The **NEXT RECORD** clause is optional and generally not used.
- Using **INTO Identifier** clause causes the data to be read into the record buffer and then copied from there to the specified Identifier in one operation.
 - When this option is used there will be two copies of the data. It is the equivalent of a READ followed by a MOVE
- **AT END clause is used to sense end-of-file. On sensing end-of-file, the StatementBlock is executed.**

WRITE Syntax.

WRITE RecordName [FROM Identifier]

[
 {BEFORE
 AFTER} ADVANCING {
 AdvanceNum [LINE]
 LINES]
 MnemonicName
 PAGE
 }]

- To WRITE data to a file, move the data to the record buffer (declared in the FD entry) and then WRITE the contents of record buffer to the file.
- Using FROM Identifier causes the data moved to the record buffer from the specified Identifier and then the contents of record buffer transferred to the file
- ADVANCING option used for printing reports

Report Printing example

```
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT ReportFile ASSIGN TO "STUDENTS.RPT"  
        ORGANIZATION IS LINE SEQUENTIAL.  
  
DATA DIVISION.  
FILE SECTION.  
FD ReportFile.  
01 PrintLine          PIC X(40).  
  
WORKING-STORAGE SECTION.  
  
01 Header-1  PIC X(21) VALUE " Student List".  
01 Header-2  PIC X(40) VALUE "Roll NO      Name      Marks".  
01 Header-3  PIC X(40) VALUE All "-".
```

```
WRITE PrintLine FROM Header-1 AFTER ADVANCING PAGE  
WRITE PrintLine FROM Header-3 AFTER ADVANCING 2 LINES  
WRITE PrintLine FROM Header-2  
WRITE PrintLine FROM Header-3 BEFORE ADVANCING 2 LINES.
```

Environment & Data Divisions

Environment Division

- Machine dependent division of a COBOL program
- Supplies information about the computer equipment and files (datasets) used in compiling and executing the program
- Includes following sections
 - **CONFIGURATION SECTION**
 - ✓ Used to Indicate the computer which will be used for compiling (SOURCE-COMPUTER) and executing the program (OBJECT-COMPUTER)
 - **INPUT-OUTPUT SECTION**
 - ✓ Supply information on the input and output devices used
 - ✓ Consists of FILE-CONTROL paragraph, which is used to select a file and assign it to a device

Environment Division

CONFIGURATION SECTION.

- Format

```
ENVIRONMENT DIVISION.  
  CONFIGURATION SECTION.  
    [SOURCE-COMPUTER. source machine.]  
    [OBJECT-COMPUTER. target machine.]
```

- SOURCE-COMPUTER Paragraph
 - The Computer on which the source program is to be compiled.
 - The Computer Name in a system name in the form of IBM-3090
- OBJECT-COMPUTER Paragraph
 - Identifies the computer on which the object program is to be executed..

Environment Division

INPUT-OUTPUT SECTION.

- The Input-Output Section defines each file, identifies its external storage medium, assigns the file to input / output device.
- This section is divided into two paragraphs:
 - FILE-CONTOL Paragraph: Which names and associates the files with external media.
 - I-O-CONTROL Paragraph: Which defines special input output techniques to be used.

Environment Division

- FILE-CONTROL Paragraph
 - Associates each file in the COBOL program with an external medium, and allows specification of file organization, access mode and other information.
 - Each file described in an FD or SD entry in Data Division must be described in one and only one entry in File-Control Paragraph.
 - The keyword FILE-CONTROL may appear only once, at the beginning of the FILE-CONTROL Paragraph.
 - The word FILE-CONTROL must begin in Area A, and be followed by a period.
 - Each FILE-CONTROL entry must start in Area B with the SELECT Clause.

Environment Division

➤ Sequential File Entries

FILE-CONTROL.

```
SELECT filename-1 ASSIGN TO assignment-name  
[ ORGANIZATION IS SEQUENTIAL ]  
[ ACCESS MODE IS SEQUENTIAL ]  
[ FILE STATUS IS dataname-1 ]
```

- ✓ ORGANIZATION is SEQUENTIAL: A predecessor-successor relationship of records in the file is established by the order in which the records are placed in the file when it is created or extended.
- ✓ ACCESS MODE is SEQUENTIAL: For files with sequential organization, records in the file are accessed in the sequence established when the file is created or extended
- ✓ format of assignment-name is system dependent

FILE STATUS Clause

- Allows the user to monitor the execution of each input / output request for the file.
- Data-name-1 is the status key data item.
- Data-name-1 must be defined in the Data Division as a two-character alphanumeric item.
- When the FILE STATUS clause is specified, the system moves a value into the status key data item after each input/output request
- FILE STATUS value can be tested by the program for the success / failure of any file operation
- Some common FILE STATUS values :
 - 00 Success
 - 10 *reached end of file*
 - 23 *no record found with the key specified*
 - 47 *wrong open mode for READ / START*

Data Division Concepts

DATA DIVISION defines and describes fields, records and files in storage processed by the object program.

External Data (described in FILE SECTION)

- External data is contained in Files
- A File is a collection of records stored in some input/output device.
- The records can be thought of as physical records or logical records.
- A Physical Record is a unit of data that is treated as an entity when moved into or out of the storage device.
- The size of physical record is determined by the particular input/output device.
- The size does not have relationship with the logical information contained in the file.
- The FD (File Description) entry specifies the physical aspects of data.

Data Division Concepts

- A Logical record is a unit of data whose subdivisions have a logical relationship.
- A logical record may itself be a physical record or several logical records may be contained within a physical record .
- Record Description entries, which follow the FD entry for a specific file, describe the logical records in the file.
- Once the relationship between logical and physical records is established, only logical records are available to COBOL program.

Internal Data (described in WORKING-STORAGE SECTION)

- The logic of the program may develop additional data within storage. Such data is called internal data.

Data Division - Data Description

- Valid Level Numbers
 - 01: Specifies the record itself and is the most inclusive level number possible. May be either an elementary item or a group item.
 - 02 – 49: Specify group and elementary item in a record. Less inclusive data items are assigned higher level numbers.
 - 66: Identifies elementary or group items described by RENAMES Clause
 - 77: Identifies independent Working-Storage or Linkage Section items which are not subdivisions of other items, and which are not themselves subdivided. This level number is obsolete
 - 88: Identifies any condition-name entry that is associated with a particular value of a conditional variable
- data names (variables)
 - Fields can be identified by data names
 - data names of subordinate items need not be unique. But they should be properly qualified while referring them in Procedure Division

Data Division - Data Description

Data Description Entry - General Format

```
level-number {data-name / FILLER}
    [REDEFINES clause] [PIC clause] [VALUE clause]
    [BLANK-WHEN-ZERO clause] [JUSTIFIED clause]
    [OCCURS clause] [USAGE clause] [SYNCHRONIZED clause]
```

- Level-number can be any number from 01 to 49 ,or 77.
- The clauses may be written in any order with few exceptions
 - The data-name/FILLER entry must immediately follow the level-number.
 - When specified, REDEFINES clause must follow the data-name clause.
- The BLANK WHEN ZERO, JUSTIFIED, PICTURE and SYNCHRONIZED clauses are valid only for elementary items.

Data Division - Data Description

- data-name-1/FILLER Clause
 - The data-name-1 explicitly identifies the data being described; the key word FILLER specifies an item that is not explicitly referred to in a program.
 - A data-name-1 identifies a data item used in the program.
 - The data item may assume number of different values during program execution.
 - Under no circumstances may a FILLER item be explicitly referred to in Procedure Division

- PICTURE Clause

$$\left\{ \begin{array}{l} \text{PICTURE} \\ \text{PIC} \end{array} \right\} \text{ IS } <\text{picture string}>$$

- Specified for every Elementary Item
- picture string can contain a max of 30 Code Characters

Data Division - Data Description

Basic PICTURE characters

- Used to represent data internally
- Used to specify the type of data item
- Characters S, V, P do not occupy space. Used only to assume sign and position of decimal point

PIC Char	Meaning	Sample Picture
9	Numeric Data Item	9999
X	Any allowable character from the character set	XXXX
A	a Letter or Space	AAAA
V	Assumed Decimal Point	99V99
P	Position of Assumed Decimal Point when it lies outside the data item	99PP
S	Data Item is Signed	S9(4)

Data Division - Data Description

Editing PICTURE characters

- Used to represent data in the output
- Data is edited to be represented in presentable and human understandable format
- Numeric edited fields will no more be numeric. They cannot be used in any arithmetic operations
- Characters Z * \$, . + - CR DB are used to edit numeric data. These characters are counted in the size of the data item

Example :

S99V99	(basic data)	size 4 bytes
-99.99	(edited data)	size 6 bytes
999CR	(edited data)	size 5 bytes

Data Division - Data Description

Editing PICTURE characters

Code Char	Meaning	Sample Picture
B	Inserts space	XXBXXX
0	Inserts zero	9(5)000
/	Inserts Slash	99/99/99
,	Inserts Comma	99,999,999
.	Inserts Decimal Point	9999.99
Z	Replaces leading zeros with space	ZZZZ.ZZ
*	Replaces leading zeros with *	****9.99
\$	Inserts Currency sign	\$999.99
+	Inserts Sign (+ for +ve value and – for –ve value)	+99.99 99.99+
–	Inserts Sign (space for +ve value and – for –ve value)	–99.99 999–
CR	CR for –ve value, two spaces for +ve value	99.99CR
DB	DB for –ve value, two spaces for +ve value	99.99DB

Data Division - Data Description

Data editing types

- Insertion
 - Simple insertion
 - Special insertion
 - Fixed insertion
 - Floating insertion
- Suppression & Replacement
 - ✓ Zero suppression and replacement with asterisks or spaces
- EDITING ALLOWED ONLY for Categories:
 - ✓ Numeric-edited (All edits)
 - ✓ Alphanumeric-edited (Only Simple insertion)

Data Division - Data Description

Simple insertion B 0 / ,

PICTURE	Value of Data	Edited Result
X(2)BX(3)Bx(4)	19Mar1995	19 Mar 1995
9(2)/9(2)/9(2)	190395	19/03/95
99,999,000	1234	01,234,000
99,999	12345	12,345

Special insertion .

PICTURE	Value of Data	Edited Results
999.99	1.234	001.23
999.99	12.34	012.34
999.99	123.45	123.45
999.99	1234.5	234.50

Data Division - Data Description

Fixed Insertion

\$ (currency symbol)

+ - CR DB (sign control symbols)

PICTURE	Value of Data	Edited Result
999.99+	6555.556	555.55+
-9999.99	-6555.555	- 6555.55
+9999.99	1234.56	+1234.56
\$999.99	123.45	\$123.45
\$999.99	23.456	\$023.45
\$99.99	123.456	\$23.45
\$9999.99CR	123.45	\$0123.45bb
\$9999.99DB	-123.45	\$0123.45DB
\$9999.99DB	123.45	\$0123.45bb
\$9999.99CR	-123.45	\$0123.45CR

Data Division - Data Description

Floating Insertion

\$ + -

PICTURE	Value of Data	Result
\$\$\$\$.99	2.123	bb\$2.12
\$\$\$9.99	0.12	bb\$0.12
\$,\$\$\$,999.99	1234.56	bbb\$1,234.56
+++-999.99	-1256.789	bb-1256.78
\$\$,\$\$\$,\$\$\$,.99CR	-1234567	\$1,234,567.00CR
++,++,++,++	0	
\$\$\$99.99CR	123.45	b\$123.45bb
\$\$\$99.99CR	-123.45	b\$123.45CR
\$\$\$99.99DB	123.45	b\$123.45bb
\$\$\$99.99DB	-123.45	b\$123.45DB

Data Division - Data Description

Leading zero Suppression

Z * (Mutually exclusive)

PICTURE	Value of Data	Result
**** . **	0	*****
ZZZZ.ZZ	0	bbbbbbb
ZZZZ.99	56.67	bb56.67
****.99	56.67	**56.67
Z,ZZZ.ZZ	123.456	bb123.45

Data Division - Data Description

REDEFINES Clause

- Allows the same area of memory to be referenced by more than one data-name with different format and sizes.
- Format :

```
level-number data-name-1 REDEFINES data-name-2
```

- The level-numbers of data-name-1 and data-name-2 must be identical and must not be level 88, 77 or 66.
- data-name-2 is the redefined item.
- data-name-1 identifies an alternate description for the same area.
- Multiple redefinitions of the same storage area are permitted.
- Multiple redefinitions must all use the data-name of the original entry that defined this storage area.
- The redefining entry (data-name-1) and any sub-ordinate entries must not contain any VALUE clauses

Data Division - Data Description

REDEFINES Example 1 :

```
01 DATA-REC.  
      05 VAR1    PIC X(4).  
      05 VAR2 REDEFINES VAR1 PIC 9(4).  
      05 VAR3.  
          10 VAR4 PIC XX.  
          10 VAR5 PIC 999.  
      05 VAR6 REDEFINES VAR3.  
          10 VAR7 PIC 9999.  
          10 VAR8 PIC X.
```

Data Division - Data Description

REDEFINES Example 2 :

- In this example, 8 Byte date can be referred in DDMMYYYY or MMDDYYYY format based on the data present at a given point of time

```
01 DATE-DETAILS.
```

```
  05 EURO-DATE.
```

```
    10 EDD  PIC 99.
```

```
    10 EMM  PIC 99.
```

```
    10 EYY  PIC 9999.
```

```
  05 US-DATE REDEFINES EURO-DATE.
```

```
    10 UMM  PIC 99.
```

```
    10 UDD  PIC 99.
```

```
    10 UYY  PIC 9999.
```

Data Division - Data Description

BLANK WHEN ZERO Clause

- Specifies that an item is to be filled entirely with spaces when its value is zero.
- Format :

BLANK WHEN ZERO

- May be specified ONLY for elementary numeric or numeric edited items.
- Must not be specified for level-66 or level-88 items.

Data Division - Data Description

JUSTIFIED Clause

- Overrides standard positioning rules for a receiving item of the alphabetic or alphanumeric categories.
- Format :

```
{ JUSITIFIED }  
{ JUST }      RIGHT
```

- Must not be specified for any numeric item.
- When JUSTIFIED clause is specified for a receiving item, the data is aligned at the rightmost character position
- If the sending item is larger than the receiving item, the leftmost characters are truncated.
- If the sending item is smaller than the receiving item, the unused character positions at the left are filled with spaces.

Data Division - Data Description

OCCURS

- Used for defining Tables (Arrays) – Specified in detail in Table handling section.
- It eliminates the need for separate entries for repeated data items.
- Tables can be Fixed Length or Variable Length
- Can't be defined for a data item which
 - Has a level number of 01, 66, 77, or 88.
 - Describes a redefined data item

Data Division - Data Description

USAGE Clause

- Specifies the format in which data is represented in storage.
- Format :

```
USAGE IS { BINARY
           COMP          COMPUTATIONAL
           COMP-1        COMPUTATIONAL-1
           COMP-2        COMPUTATIONAL-2
           COMP-3        COMPUTATIONAL-3
           DISPLAY
           INDEX }
```

- The USAGE clause cannot be specified for a data description entry with a level-number 66 or 88.
- If it is specified at the group level, it applies to each elementary item in the group.
- The usage of an elementary item must not contradict the usage of a group to which the elementary item belongs.

Data Division - Data Description

USAGE Clause – DISPLAY Option

- Display option is default
- Specifies that the data item is stored in character form, one character per byte.

USAGE Clause – COMPUTATIONAL Option

- A computational item is a value used in arithmetic operations. It must be numeric.
- The maximum length of a computational item is 18 decimal digits.
- The PICTURE of a computational item can contain only:
 - 9 :- One or more numeric character positions
 - S :- One operational sign
 - V :- One implied decimal point
 - P :- One or more decimal scaling positions

Data Division - Data Description

BINARY – Same as COMPUTATIONAL (Comp)

- Specified for binary data items.
- Such items have a decimal equivalent consisting of the decimal digits 0 through 9, plus a sign.
- Negative numbers are represented as the two's complement of the positive number with the same absolute value.
- The amount of storage occupied by a binary item depends on the number of decimal digits defined in its PICTURE clause:

Digits in PIC Clause	Storage Occupied
1 through 4	2 Bytes (half word)
5 through 9	4 bytes (fullword)
10 through 18	8 bytes (double word)

Data Division - Data Description

COMPUTATIONAL-1 or COMP-1 (Floating-Point)

- Specified for internal floating-point items (single precision).
- COMP-1 items are 4 bytes long.
- The PIC clause cannot be specified for this item.

COMPUTATIONAL-2 or COMP-2 (Long Floating-Point)

- Specified for internal floating-point items (double precision).
- COMP-2 items are 8 bytes long.
- The advantage is that this increases the precision of the data, which means more significant digits can be available for the item.
- The PIC clause cannot be specified for this item.

Data Division - Data Description

COMPUTATIONAL-3 or COMP-3

- This is the equivalent of PACKED-DECIMAL
- In this form, each digit is represented by 4 bits
- Each byte can accommodate 2 digits
- The number of bytes = $(n+1) / 2$ Where n is the number of places specified in PIC clause
- For e.g.

01	ITEM-1	PIC	S9(7) USAGE COMP-3
----	--------	-----	--------------------

will occupy 4 bytes.

- This option is normally not used nowadays

Data Division - Data Description

INDEX

- This is fixed binary number which can store only +ve values
- 4-byte elementary data item
- Normal arithmetic operations and MOVE statement are not allowed for this item
- Value is assigned, increased or decreased using a SET statement
- Direct reference can be made in SET, SEARCH, USING and relational condition

Data Division - Data Description

SET Syntax

SET {**IndexName**
Identifier} ... **TO** {**IndexName**
Identifier
Integer}

SET {**IndexName**} ... {**UP**
DOWN} **BY** {**Identifier**
Integer}

SET {**ConditionName**} ... **TO** **TRUE**

Data Division - Data Description

VALUE Clause

- The VALUE clause is used to set an initial value to the working storage variable.
- This can be specified only in the working storage variable.
- The VALUE Clause cannot be a part of RECORD DESCRIPTION entry in the FILE SECTION
- Example :

```
01 W04-G-VAR.  
      03 RUN-TIME          PIC X(20) VALUE SPACES.  
      03 PGM-NAME          PIC X(08) VALUE 'PXXBB290'.  
      03 NO-OF-TIMES       PIC 999   VALUE 30.  
      03 BASIC-VALUE       PIC 99V99 VALUE ZERO.
```

Data Division - Data Description

RENAMES Clause

- Allows Re-grouping of Data Items
- Format :

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

- Cannot Rename 77, 88, 66 or 01 levels
- Data Names must be in proper sequence
- Not Used in Good Programs

Data Division - Data Description

RENAMES Example

```
01 EMP-REC.  
      03 EMPNO          PIC X(20).  
      03 NAME .  
          04 FNAME PIC X(10).  
          04 LNAME PIC X(10).  
      03 SALARY .  
          04 RUPEES PIC 99999.  
          04 PAISE   PIC 99.  
  
66 SOME-DATA RENAMES LNAME THRU PAISE.
```

INITIALIZE Statement

- Initializes the variables
- **Format**

INITIALIZE id-1

- Can be used at
 - **Group level**
 - **Elementary level**
- SPACE is implied source for
 - **Alpha, Alpha-numeric and Alpha-numeric edited**
- ZERO is implied source for
 - **Numeric and Numeric edited**

INITIALIZE – Examples

INITIALIZE VAR1

<u>VAR1</u>	<u>Before</u>	<u>After</u>
9(5)	12345	00000
X(5)	AB123	bbbbb
XXXXX	12AB3	bbbbb
XXBX/XX	ABbC/DE	bbbb/bb
**99.9	1234.5	**00.0
A(5)	ABCDE	bbbbb

Data Division - Data Description

Data Division Organization

- Our application program needs to know the layout of the data used.
- We define the file layout and internal data in the Data Division.
- The DATA DIVISION has the following sections.
 - FILE SECTION
 - WORKING-STORAGE SECTION
 - LINKAGE SECTION

Data Division - Data Organization

- General Format

DATA DIVISION.

FILE SECTION.

file description entry

record description entry

WORKING-STORAGE SECTION.

record / data description entries

LINKAGE SECTION.

Record / data description entries.

Data Division - Data Organization

FILE SECTION

- Layouts for the files described in this section.
- Contains File Description (FD) entry for each file defined in the input-output section of ENVIRONMENT DIVISION .
- Followed by Record Description
 - Record Description begins with Record Name with level number 01
- Contains Sort Description (SD) entry for the work files in SORT / MERGE operations (discussed in later sections)

Data Division - Data Organization

- Format

```
FD file-name-1
  [ BLOCK CONTAINS [ int-1 TO ] int-2 { CHARACTERS | RECORDS } ]
  [ RECORDING MODE IS { F V S U } ]
  [ RECORD { CONTAINS int-3 [ TO int-4 ] CHARACTERS } ]
  [ DATA { RECORD IS | RECORDS ARE } data-name-2 ... ]
```

➤ **file-name-1**

- ✓ Must be same as that specified in the association FILE-CONTROL paragraph of INPUT-OUTPUT section of ENVIRONMENT DIVISION

➤ **RECORDING MODE**

- ✓ F Fixed record size
- ✓ V Variable record size
- ✓ S Spanned record
- ✓ U Unknown record size

Data Division - Data Organization

➤ BLOCK CONTAINS

- ✓ Specifies the size of physical record
- ✓ This clause can be omitted when:
 - Each physical record contains one and only one complete logical record.
- ✓ Int-1 and int-2 must be nonzero unsigned integers.
- ✓ When the CHARACTERS option is specified, the physical record size is specified as number of character positions required to store record.
- ✓ When the RECORDS option is specified, the physical record size is expressed as number of logical records contained in each physical record.

➤ RECORD CONTAINS

- ✓ Specifies the size of a data record.
- ✓ The RECORD CONTAINS Clause is never required as the size of each record is completely defined in the record description entries.

➤ DATA RECORDS

- ✓ Serves only as documentation for the names of data records associated with this file.

Data Division - Data Organization

WORKING-STORAGE SECTION.

- Succeeds File Section.
- Used to define internal data.
- Common Practice to Group related items together.

LINKAGE SECTION.

- Describes data made available from another program.
- Storage within the program is not reserved because the data area exists elsewhere.

Handling Conditions

Conditional Statements

- Specifies a truth value of a condition is to be determined and that the subsequent action of the object program depends on the truth value.
- Various Conditional Statements are :
 - IF
 - EVALUATE
- There are other statements that can perform depending on conditions specified. These will be discussed in individual statement description.

Conditional Statements

IF

- Causes a condition to be evaluated, and provides for alternative actions in the object program, depending on truth value
- Format :

```
IF condition THEN
  { statement-1
    NEXT SENTENCE
  }
[ ELSE
  { statement-2
    NEXT SENTENCE
  }
]
END-IF
```

Conditional Statements

IF

- END-IF or a Period can be the scope terminator of an IF Statement.
- NEXT SENTENCE means execution should move to statement after the next separator period. It is equal to 'Do Nothing'
- It is not a good programming practice to use NEXT SENTENCE and Period as IF terminator.

Conditional Statements

CONTINUE

- Allows you to specify a no operation statement. It indicates that no executable instruction is present.
- Syntax :
 - CONTINUE
- It can be used anywhere a conditional statement or an imperative statement can be used. It has no effect on the execution of the program.

Example

```
IF YEAR < 2000
    CONTINUE
ELSE
    ADD 1 TO AGE
END-IF
```

Conditional Expressions

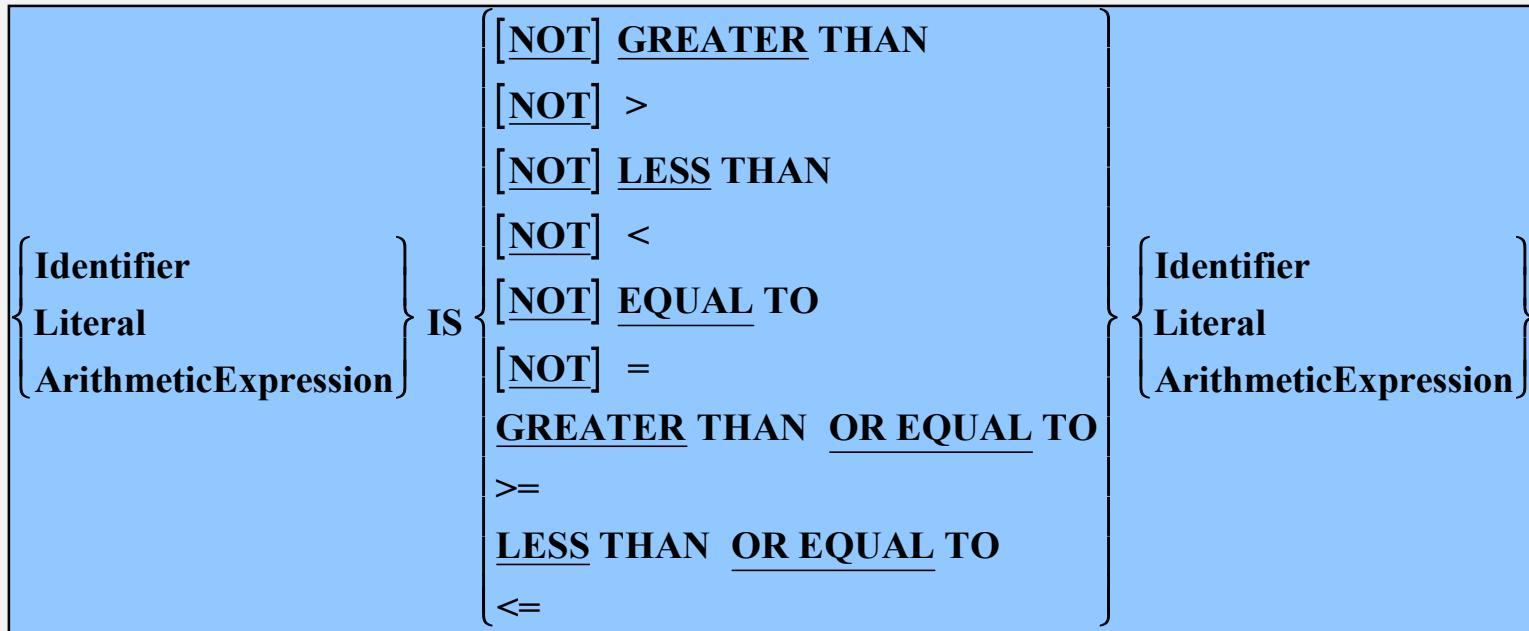
Simple Conditions

- Relational Conditions
- Class Conditions
- Sign Conditions

Complex Conditions

Condition Names

Relational Condition



- For numeric expressions, value is compared. Comparison permitted regardless of Usage clause
 - Example: 012 12.00 12 +12 are all equal
- For non-numeric operands, comparison starts from left most character
 - For operands size unequal, the comparison proceeds as if the shorter operand had been padded by blanks on the right

Class Condition

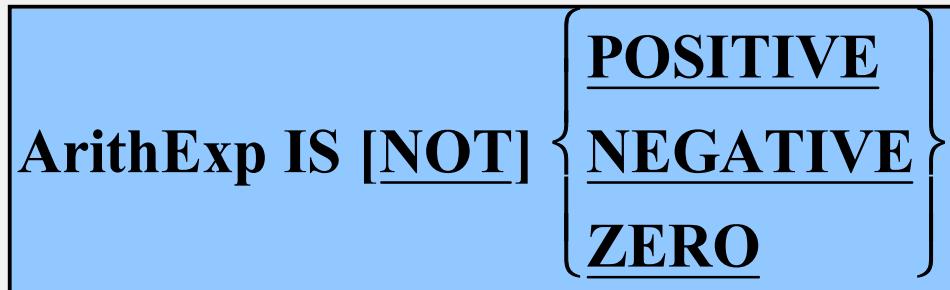
Identifier IS [NOT] {

NUMERIC
ALPHABETIC
ALPHABETIC-LOWER
ALPHABETIC-UPPER
User Defined Class Name

}

- Although COBOL data items are not ‘typed’, they do fall into some broad categories, or classes, such a numeric or alphabetic etc
- A class condition determines the value of data item is a member of one of these classes
- User defined class can be specified in SPECIAL-NAMES paragraph of CONFIGURATION SECTION in ENVIRONMENT DIVISION

Sign Condition



- The sign condition determines whether or not the value of an arithmetic expression is less than, greater than or equal to zero
- Sign conditions are just another way of writing some of the relational conditions

Complex conditions.

Condition $\left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\}$ Condition ...

- Programs often require conditions which are more complex than single value testing or determining a data class.
- Like all other programming languages, COBOL allows simple conditions to be combined using OR and AND to form composite conditions.
- Like other conditions, a complex condition evaluates to true or false.
- A complex condition is an expression which is evaluated from left to right unless the order of evaluation is changed by the precedence rules or brackets

Complex conditions

Precedence Rules.

1. NOT
2. AND
3. OR

- Complex conditions are evaluated using precedence rules and the order of evaluation may be changed by using brackets.
- Examples :

```
IF Row > 0 AND Row < 26 THEN  
    DISPLAY "On Screen"  
END-IF
```

```
IF (VarA > VarC OR VarC = VarD) AND VarA NOT = VarF  
    DISPLAY "Done"  
END-IF
```

Implied Subjects.

- When a data item is involved in a relational condition with many other items, it can be tedious to have to repeat the data item for each condition.
- Example :

```
IF TotalAmt > 10000 AND TotalAmt < 50000 THEN  
  IF Grade = "A" OR Grade = "B+" OR GRADE = "B" THEN  
    IF VarA > VarB AND VarA > VarC AND VarA > VarD  
      DISPLAY "VarA is the Greatest"  
    END-IF
```

- In these situations COBOL provides an abbreviation mechanism called implied subjects.
- The statements above may be re-written using implied subjects as --

```
IF TotalAmt > 10000 AND < 50000 THEN  
  IF Grade="A" OR "B+" OR "B" THEN  
    IF VarA > VarB AND VarC AND VarD  
      DISPLAY "VarA is the Greatest"  
    END-IF
```

Implied Subjects
TotalAmt
Grade =
VarA >

Nested IFs

```
IF ( VarA < 10 ) AND ( VarB NOT > VarC ) THEN
    IF VarG = 14 THEN
        DISPLAY "First"
    ELSE
        DISPLAY "Second"
    END-IF
ELSE
    DISPLAY "Third"
END-IF
```

VarA	VarB	VarC	VarG	DISPLAY
3	4	15	14	First
3	4	15	15	Second
3	4	3	14	Third
13	4	15	14	Third

Condition Names.

```
88 ConditionName {VALUE } { Literal  
          VALUES } { LowValue {THROUGH } HighValue } ...  
          { THRU }
```

- Wherever a condition can occur, such as in an IF statement or an EVALUATE or a PERFORM..UNTIL, a CONDITION NAME (Level 88) may be used.
- A Condition Name is essentially a BOOLEAN variable which is either TRUE or FALSE.
- Condition Names are defined in the DATA DIVISION using the special level number 88.
- They are always associated with a data item and are defined immediately after the definition of the data item.
- A condition name takes the value TRUE or FALSE depending on the value in its associated data item.
- The VALUE clause is used to identify the values which make the Condition Name TRUE

Condition Names example

```
02 GRADE-ID PIC 99
    88 PRIMARY-OTHER VALUE 1 THRU 3,5,6.
    88 PRIMARY-FOUR VALUE 4.

02 END-FLAG PIC 9 VALUE 0.
    88 END-OF-FILE VALUE 1.

IF PRIMARY-OTHER . . . (Tests GRADE-ID for values 1,2,3,5,6)
IF PRIMARY-FOUR. . . (Same as IF GRADE-ID = 4)

IF END-OF-FILE . . . (Same as IF END-FLAG = 1)
```

Using the SET verb for Condition names

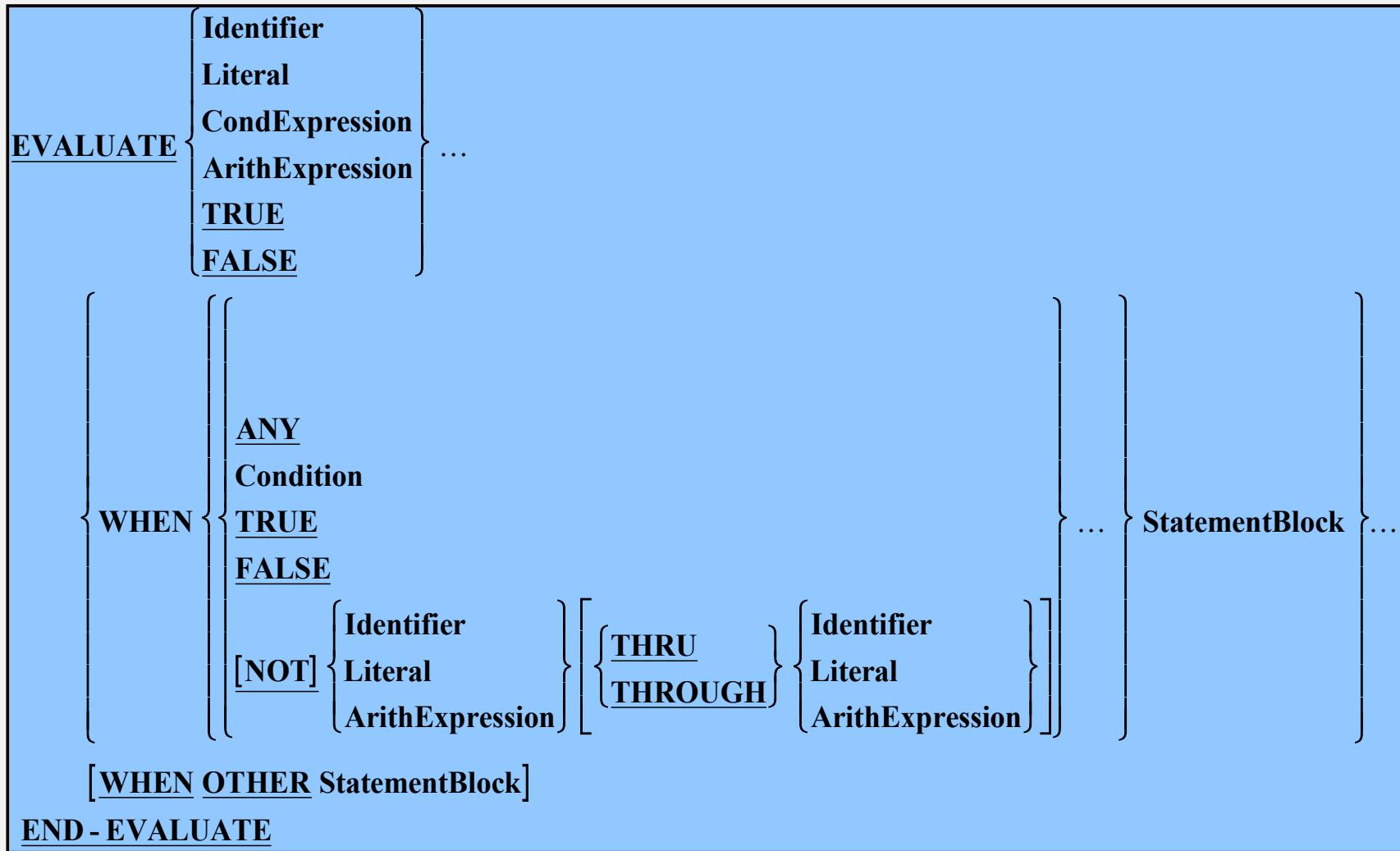
```
02 GRADE-ID PIC 99
    88 PRIMARY-OTHER VALUE 1 THRU 3,5,6.
    88 PRIMARY-FOUR VALUE 4.
02 END-FLAG PIC 9 VALUE 0.
    88 END-OF-FILE VALUE 1.
```

SET PRIMARY-FOUR TO TRUE . . . (Same as MOVE 4 TO GRADE-ID)

SET PRIMARY-OTHER TO TRUE. . . (Same as MOVE 1 TO GRADE-ID)

SET END-OF-FILE TO TRUE . . . (Same as MOVE 1 TO END-FLAG)

EVALUATE Statement



EVALUATE Statement

- It can evaluate multiple conditions. .
- Provides better readability.
- When a WHEN condition is satisfied, processing breaks off from the rest of the evaluate block.
- WHEN OTHER is used when none of the conditions is satisfied

Example - 1

```
EVALUATE proc-type ALSO cust-type
WHEN 1 ALSO 1
      MOVE 1 TO RESULT
WHEN 3 ALSO 1 THRU 2
      MOVE 2 TO RESULT
WHEN OTHER
      MOVE 0 TO RESULT
END-EVALUATE
```

EVALUATE Statement

Example – 2

```
EVALUATE TRUE
    WHEN I-VALUE < 100
        MOVE 1 TO RESULT
    WHEN I-VALUE < 1000
        MOVE 2 TO RESULT
    WHEN I-VALUE < 10000
    WHEN I-VALUE = 20000
        MOVE 0 TO RESULT
END-EVALUATE
```

- Multiple WHENs are allowed for a single imperative statement. In the above example ‘0’ will be moved to RESULT on “When I-value < 10000 and Also When I-VALUE = 20000”

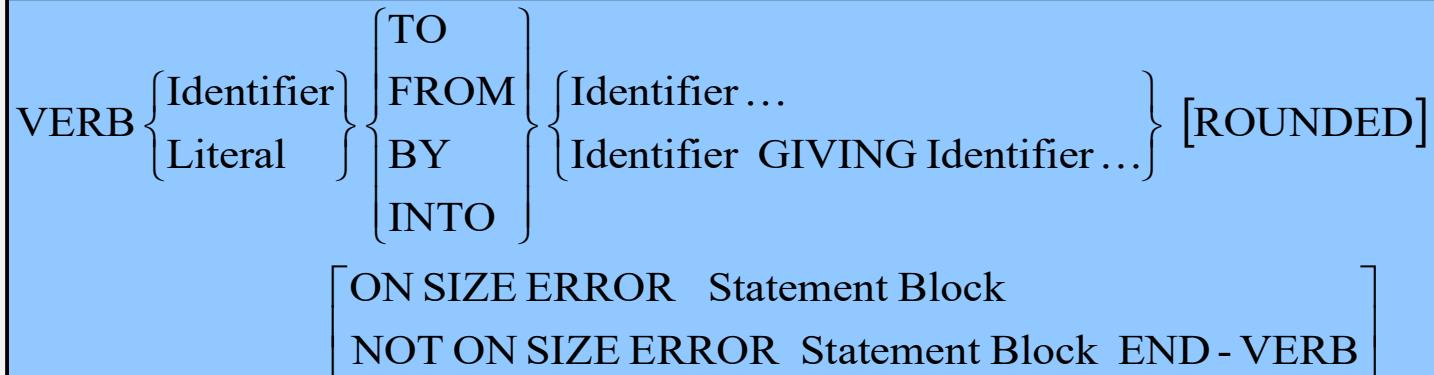
Handling Calculations

Arithmetic Expressions

Arithmetic Expressions

- May consist of any of the following:
 - An identifier described as a numeric elementary item.
 - A numeric literal
 - Identifiers and literals separated by arithmetic operators
 - Two arithmetic expressions separated by an arithmetic operator.
- Operators
 - BINARY : + - * / **
 - Unary : + -
- Precedence
 - Unary, Exponentiation, Multiplication and Division, Addition and Subtraction
 - Parentheses may be used in arithmetic expressions to specify the order in which elements are to be evaluated.

Arithmetic Verb Template



- Most COBOL arithmetic verbs conform to the template above.
- Examples :

ADD Takings TO CashTotal.

ADD Males TO Females GIVING TotalStudents.

SUBTRACT Tax FROM GrossPay.

SUBTRACT Tax FROM GrossPay GIVING NetPay.

DIVIDE Total BY Members GIVING MemberAverage.

DIVIDE Members INTO Total GIVING MemberAverage.

MULTIPLY 10 BY Magnitude.

MULTIPLY Members BY Subs GIVING TotalSubs.

The exceptions are the COMPUTE and the DIVIDE with REMAINDER

Arithmetic operations

- **ON SIZE ERROR Option**
 - If the ON SIZE ERROR phrase is specified and a size error condition occurs, the value of the resultant identifier affected by the size error is not altered--that is, the error results are not placed in the receiving identifier.
- **ROUNDED Option**
 - When size of the result exceeds the number of places provided in the storage, truncation occurs if ROUNDED option is not used.
 - With ROUNDED option, low order digit incremented by 1 whenever absolute value of the next least significant digit is ≥ 5

ROUNDED option

- ◆ The ROUNDED option takes effect when, after decimal point alignment, the result calculated must be truncated on the right hand side.
- ◆ The option adds 1 to the receiving item when the right most truncated digit has an absolute value of 5 or greater.
- ◆ Example :

Receiving Field	Actual Result	Truncated Result	Rounded Result
PIC 9(3)V9	123.25	123.2	123.3
PIC 9(3)	123.55	123	124

ON SIZE ERROR option

- A size error condition exists when, after decimal point alignment, the result is truncated on either the left or the right hand side.
- If an arithmetic statement has a rounded phrase then a size error only occurs if there is truncation on the left hand side (most significant digits)
- Example :

Receiving Field	Actual Result	SIZE ERROR
PIC 9(3)V9.	245.96	Yes
PIC 9(3)V9.	1245.9	Yes
PIC 9(3).	124	No
PIC 9(3).	1246	Yes
PIC 9(3)V9	124.45	Yes
PIC 9(3)V9 Rounded	124.45	No
PIC 9(3)V9 Rounded	3124.45	Yes

ADD Examples

	ADD	Cash	TO	Total.
Before		3		1000
After		3		1003

	ADD	Cash,	20	TO	Total,	Wage.
Before			3		1000	100
After			3		1023	123

	ADD	Cash,	Total	GIVING	Result.
Before		3	1000		0015
After		3	1000		1003

	ADD	Males	TO	Females	GIVING	TotalStudents.
Before		1500		0625		1234
After		1500		0625		2125

SUBTRACT Examples

SUBTRACT Tax FROM GrossPay, Total.

Before	120	4000	9120
After	120	3880	9000

SUBTRACT Tax, 80 FROM Total.

Before	100	480
After	100	300

SUBTRACT Tax FROM GrossPay GIVING NetPay.

Before	750	1000	0012
After	750	1000	0250

MULTIPLY Examples

	MULTIPLY	Subs	BY	Members	GIVING	TotalSubs
Before		15.50		100		0123.45
After		15.50		100		1550.00

MULTIPLY 10 BY Magnitude, Size.

Before		355		125
After		550		250

MULTIPLY Total BY Members GIVING Average

Before	9234.55		100	1234.56
After	9234.55		100	3455.00

DIVIDE Examples

DIVIDE 15 INTO Male, Female

Before	100	123
After	006	008

DIVIDE 100 INTO Magnitude GIVING Size.

Before	3587	125
After	3587	035

DIVIDE Total BY Members GIVING Average

Before	9234.55	100	1234.56
After	9234.55	100	0092.34

The Divide Exception

DIVIDE {*Identifier*
 Literal} INTO {*Identifier*
 Literal} GIVING {*Identifier* [ROUNDED]} REMAINDER *Identifier*
 [{ON SIZE ERROR
 {NOT ON SIZE ERROR} } StatementBlock END - DIVIDE]

DIVIDE {*Identifier*
 Literal} BY {*Identifier*
 Literal} GIVING {*Identifier* [ROUNDED]} REMAINDER *Identifier*
 [{ON SIZE ERROR
 {NOT ON SIZE ERROR} } StatementBlock END - DIVIDE]

DIVIDE 201 BY 10 GIVING Quotient REMAINDER Remain.

Before

209

424

After

020

001

COMPUTE statement

```
COMPUTE {Identifier [ ROUNDED ]} ... = ArithmeticExpression  
      [ {ON SIZE ERROR  
           [NOT ON SIZE ERROR] } StatementBlock END - COMPUTE ]
```

Precedence Rules

1. ** POWER
2. * MULTIPLY
- / DIVIDE
3. + ADD
- SUBTRACT

Example :

	Compute	TotPer	=	Total	/ Rate	* 100.
Before		1000.50		156.25	87	
After		0179.59		156.25	87	

CORRESPONDING option

- Valid in ADD, SUBTRACT and MOVE statements
- CORR also can be used
- The CORRESPONDING option allows operations to be performed on subordinate items of the same name simply by specifying the group items to which they belong.
- A pair of subordinate items from the identifiers of CORRESPONDING phrase correspond if
 - ✓ Both have same name and qualifiers
 - ✓ At least one is an elementary item in case of MOVE
 - ✓ Both are elementary Numeric items for ADD/SUBTRACT
 - ✓ Usage Is not INDEX
 - ✓ Both do not include a REDEFINES, RENAMES, OCCURS

ADD	CORRESPONDING	item1	TO	item2
SUBTRACT	CORRESPONDING	item1	FROM	item2
MOVE	CORRESPONDING	item1	TO	item2

MOVE CORRESPONDING Example

```
01 INREC.  
  02 EmpNo      PIC X(4).  
  02 Name       PIC X(16).  
  02 Salary     PIC 9(6)V99.  
  02 Gender     PIC X.  
  
01 OUTREC.  
  02 EmpNo      PIC X(4).  
  02 FILLER    PIC XXX.  
  02 Name       PIC X(16).  
  02 FILLER    PIC XXX.  
  02 Salary     PIC Z(6).99.  
  02 FILLER    PIC XXX.  
  02 Net-Sal   PIC Z(6).ZZ.
```

EmpNo, Name, Salary will
be moved

MOVE CORRESPONDING INREC TO OUTREC

Sequence Control

The PERFORM Verb

- Iteration is an important programming construct. We use iteration when we need to repeat the same instructions over and over again.
- Most programming languages have several iteration keywords (e.g. WHILE, FOR, REPEAT) which facilitate the creation of different ‘types’ of iteration structure.
- COBOL has only one iteration construct; PERFORM.
- But the PERFORM has several variations.
- Each variation is equivalent to one of the iteration ‘types’ available in other languages.
- First three formats are discussed now and Format 4 will be discussed along with Table handling

Format 1 Syntax.

```
PERFORM [ 1stProc {THRU THROUGH} EndProc ]
```

- This is the only type of PERFORM that is not an iteration construct.
- It helps to transfer control to an out-of-line block of code which can be a section or paragraph.
- When the end of the block is reached, control reverts to the statement immediately following the PERFORM statement.
- 1stProc and EndProc are the names of Paragraphs or Sections.
- The PERFORM..THRU instructs to treat the Paragraphs or Sections from 1stProc TO EndProc as a single block of code.

Format 1 Example.

PARA-1.

```
DISPLAY "In PARA-1. Starting to run program"  
PERFORM PARA-3  
DISPLAY "Back in PARA-1"  
STOP RUN.
```

PARA-2.

```
DISPLAY "Now in PARA-2".
```

PARA-3.

```
DISPLAY "Now in PARA-3"  
PERFORM PARA-2  
DISPLAY "Back in PARA-3".
```

OUTPUT

```
In PARA-1. Starting to run program  
Now in PARA-3  
Now in PARA-2  
Back in PARA-3  
Back in PARA-1
```

Format 2 - Syntax

```
PERFORM [ 1stProc [ { THRU  
THROUGH } EndProc ] ]  
          RepeatCount TIMES  
          [ StatementBlock END - PERFORM ]
```

- Makes a block of code executed fixed number of times
- If a procedure (section or paragraph) is referred, it is called out of line perform. control is transferred to an out-of-line block of code which is repeated given number of times.
- If StatementBlock is used followed by END-PERFORM, it is called in line perform. The block of code is executed given number of times.
- END-PERFORM cannot be used for out of line perform

Format 2 Example

PARA-1

```
DISPLAY "Starting to run program"
PERFORM 2 TIMES
    DISPLAY "This is an in line Perform"
END-PERFORM
DISPLAY "Finished in line Perform"
PERFORM PARA-2 4 TIMES
DISPLAY "Back in PARA-1. About to Stop".
STOP RUN.
```

PARA-2.

```
DISPLAY "This is an out of line Perform".
```

OUTPUT

```
Starting to run program
This is an in line Perform
This is an in line Perform
Finished in line Perform
This is an out of line Perform
Back in PARA-1. About to Stop
```

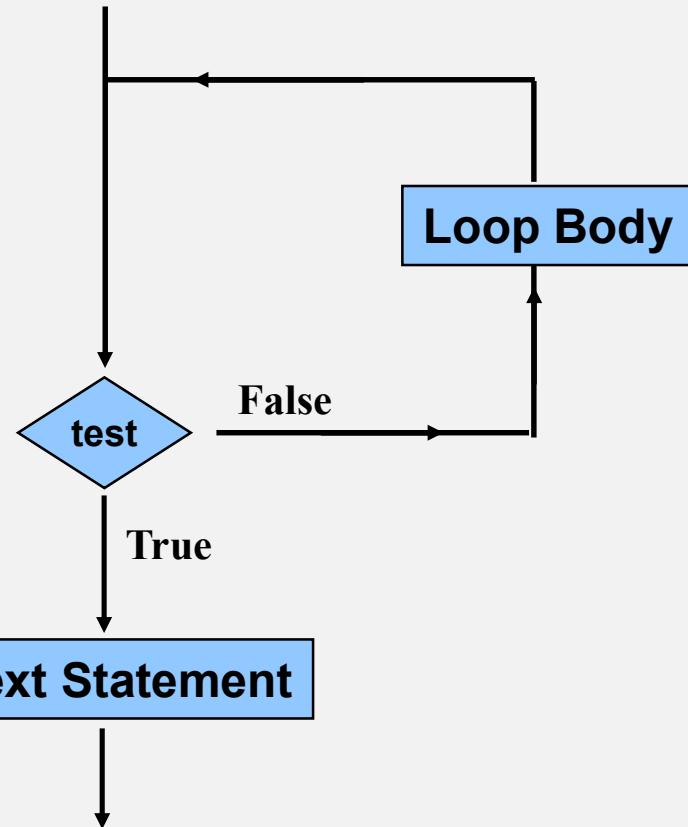
Format 3 Syntax

```
PERFORM [ 1stProc [ { THRU } | THROUGH ] EndProc ] ] [ WITH TEST { BEFORE } |  
          UNTIL Condition  
          [ StatementBlock END - PERFORM ] ]
```

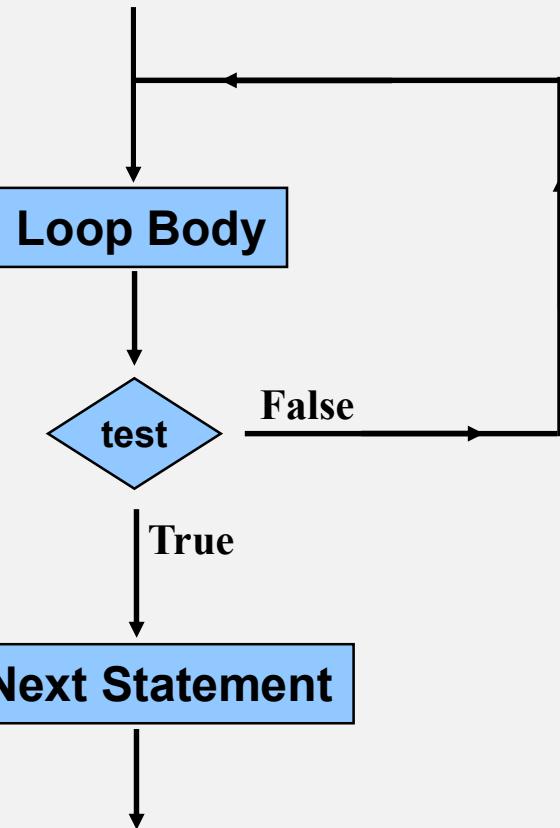
- This format is equivalent to WHILE loop in C language.
- Iteration will continue as long as the condition is FALSE and stops when the condition becomes TRUE
- If the WITH TEST BEFORE phrase is used, the PERFORM behaves like a WHILE loop in C language and the condition is tested before the loop body is entered.
- If the WITH TEST AFTER phrase is used, the PERFORM behaves like a DO-WHILE loop in C language and the condition is tested after the loop body is executed every time.
- The WITH TEST BEFORE phrase is the default .
- Both in line and out of line performs can be constructed with this format.

Test Before / After

**PERFORM WITH
TEST BEFORE**



**PERFORM WITH
TEST AFTER**



PERFORM in sequential file processing

- In general terms, the PERFORM UNTIL loop is an ideal construct for processing sequential files to process records one by one.
- Because the number of records is not known in advance we have to be careful to process the data and also detect the end of the file.
- A useful way for solving this problem uses a strategy known as “read ahead”.
- With the “read ahead” strategy we always try to stay one record ahead of the processing.
- The general format of the “read ahead” algorithm is as below

```
Attempt to READ first record
WHILE NOT EndOfStream
    Process data item
    Attempt to READ next record
ENDWHILE
```

PERFORM in sequential file processing - example

```
DATA DIVISION.  
FILE SECTION.  
FD StudentFile.  
01 StudentRecord PIC X(25).
```

```
WORKING-STORAGE SECTION.  
01 END-FLAG PIC 9.
```

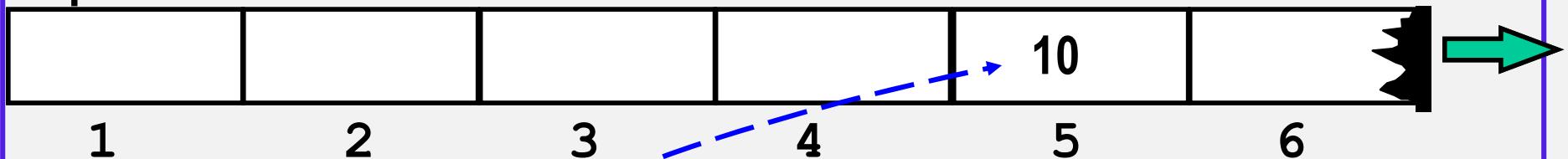
```
PROCEDURE DIVISION.  
MAIN-PARA.  
    OPEN INPUT StudentFile  
    MOVE ZERO TO END-FLAG  
    READ StudentFile  
        AT END MOVE 1 TO END-FLAG  
    END-READ  
PERFORM UNTIL END-FLAG = 1  
    DISPLAY StudentRecord  
    READ StudentFile  
        AT END MOVE 1 TO END-FLAG  
    END-READ  
END-PERFORM  
    CLOSE StudentFile
```

Tables

Tables/Arrays

A table is a contiguous sequence of memory locations called elements, which all have the same name, and are uniquely identified by that name and by their position in the sequence.

DepTot



MOVE 10 TO DepTot (5)

Declaring Tables.

Elementary item as a table

```
01 TaxTotals.  
    02 LocTax PIC 9(10)V99 OCCURS 26 TIMES.  
        OR  
    02 LocTax OCCURS 26 TIMES PIC 9(10)V99.
```

Group item as a table

```
01 TaxTotals.  
    02 LocTaxDetails OCCURS 26 TIMES.  
        03 LocTax          PIC 9(10)V99.  
        03 PayerCount      PIC 9(7).
```

Using Tables.

- Tables are declared in DATA DIVISION using OCCURS clause.
- Elements of the Table can be referred in PROCEDURE DIVISION using subscript which identifies the position of the element in the table.
- Subscript can be a numeric literal, a numeric variable or a numeric expression
- Navigation through all the elements by varying subscript can be done with PERFORM VARYING loop
- Multi dimensional tables can be created by using OCCURS clause in a subordinate element for a group which is a table itself
- Elements in multi-dimensional table can be referred using multiple subscripts

Two Dimensional Table.

01 JeansTable .

 02 Department OCCURS 4 TIMES .

 03 Gender OCCURS 2 TIMES .

 04 SaleValue PIC 9(8)V99 .

 04 NumSold PIC 9(7) .

Gender(4 1)

1	2	3	4
1	2	1	2

NumSold(2 1)

SaleValue(3 1)

PERFORM VARYING loop

```
PERFORM [ 1stProc [ { THRU  
      THROUGH } EndProc ] ] [ WITH TEST { BEFORE  
      AFTER } ]  
      VARYING { Identifier1  
      IndexName1 } FROM { Identifier2  
      IndexName2  
      Literal }  
      BY { Identifier3  
      Literal } UNTIL Condition1  
      [ AFTER { Identifier4  
      IndexName3 } FROM { Identifier5  
      IndexName4  
      Literal }  
      BY { Identifier6  
      Literal } UNTIL Condition2 ] ...  
      [ StatementBlock END - PERFORM ]
```

PERFORM VARYING example

```
PERFORM VARYING X FROM 3 BY 1 UNTIL X > 5  
DISPLAY X  
END-PERFORM
```

3
4
5

```
01 Filler Value '2345126789'  
02 STD-COUNT PIC 99 OCCURS 5 TIMES.  
.  
.  
.
```

```
PERFORM VARYING X FROM 1 BY 1 UNTIL X > 5  
DISPLAY STD-COUNT(X)  
END-PERFORM
```

23
45
12
67
89

PERFORM VARYING example

```
PERFORM ADD-PARA  
      VARYING X FROM 1 BY 1 UNTIL X > 4  
      AFTER Y FROM 1 BY 1 UNTIL Y > 2.
```

```
ADD-PARA.  
      ADD SaleValue(X Y) TO DepTot(X).
```

Random Access Files

Random Access files

Two types of organization (INDEXED, RELATIVE) allow accessing files both sequentially and randomly

Indexed organization

- Records are stored in the key sequence order.
- When accessed sequentially, records are read in the ascending value of the key
- To access indexed files randomly, key value has to be provided

Relative organization

- Each record has a unique address and is identified by its Relative Record Number (RRN) in the file
- The relative record number specifies the position of the record from the beginning of the file
- To access relative files randomly, record number has to be provided

Environment Division

Indexed File Entries

FILE-CONTROL.

SELECT filnam1 ASSIGN TO assignmtname1

[ORGANIZATION IS INDEXED]

[ACCESS MODE IS

{ SEQUENTIAL
RANDOM
DYNAMIC }

]

RECORD KEY IS data-name-3

[ALTERNATE RECORD KEY IS data-name-4]

[WITH DUPLICATES]

[FILE STATUS IS dataname-1]

Environment Division

- ORGANIZATION is INDEXED: The position of each logical record in the file is determined by the indexes created with the file and maintained by the system.
- ACCESS MODE is SEQUENTIAL: Records in the file are accessed in the sequence of ascending record key values within the currently used key reference.
- ACCESS MODE is RANDOM: The value placed in a record key data item specifies the record to be accessed.
- ACCESS MODE is DYNAMIC: The records in the file can be accessed sequentially or randomly, depending on the form of specific input/output request.
- RECORD KEY Clause: Specifies the data item within the record that is the prime record key for an indexed file.
 - ✓ The values contained in the prime record key data item must be unique among all records in the file.
 - ✓ Data-name-3 is the prime RECORD KEY data item. It must be defined within a record description entry associated with the file.

Environment Division

- ALTERNATE RECORD KEY Clause: Specifies the data item within the record that provides an alternative path to the data in an indexed file.
 - ✓ Data-name-4 is the ALTERNATE RECORD KEY data item
 - ✓ The leftmost character position of data-name-4 must not be same as the leftmost character position of the RECORD KEY or any other ALTERNATE RECORD KEY.
 - ✓ If DUPLICATES option is not specified , the values contained in the ALTERNATE RECORD KEY data item must be unique among all records in the file.
 - ✓ If the DUPLICATES option is specified, the values contained in the ALTERNATE RECORD KEY data item may be duplicated within any record in the file.

Environment Division

Relative File Entries

FILE-CONTROL.

SELECT filnam1 ASSIGN TO assignmtnname1

[ORGANIZATION IS RELATIVE]

ACCESS MODE IS

{ SEQUENTIAL
RANDOM
DYNAMIC }]

RELATIVE KEY IS dataname-3

[FILE STATUS IS dataname-1]

Environment Division

- ✓ ORGANIZATION is RELATIVE : The position of each logical record in the file is determined by its relative record number.
- ✓ ACCESS MODE is SEQUENTIAL: Records in the file are accessed in the sequence of ascending record numbers.
- ✓ ACCESS MODE is RANDOM: The value placed in a relative key (record number) data item specifies the record to be accessed.
- ✓ ACCESS MODE is DYNAMIC: The records in the file can be accessed sequentially or randomly, depending on the form of specific input/output request

Environment Division

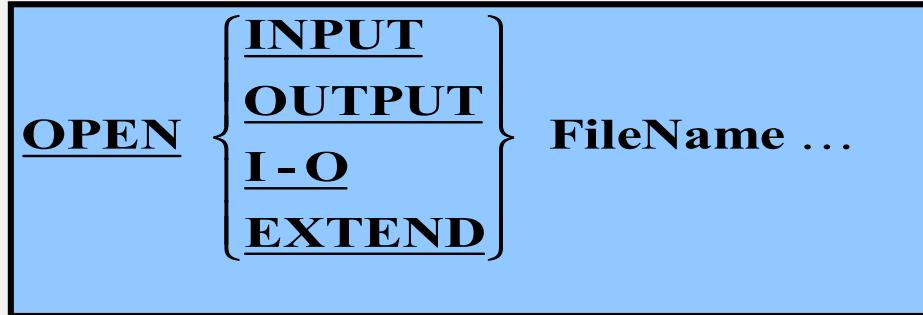
✓ RELATIVE KEY Clause

- Specifies the RELATIVE RECORD NUMBER for a specific logical record within a relative file.
- Data-name-3 is the RELATIVE KEY data item. It must be defined as an unsigned integer data item and must NOT be defined in a record description entry associated with this file.
- The Relative key is not part of record.

Defaults

- When the ORGANIZATION Clause is omitted, SEQUENTIAL ORGANIZATION is assumed
- When ACCESS MODE Clause is omitted, SEQUENTIAL ACCESS MODE is assumed.

Indexed / Relative Files - OPEN



- Opens the file for processing
- EXTEND is allowed only for sequential files.
- OUTPUT clears the file of its existing records
- I-O allows both reading and writing

Indexed Files - READ

```
READ FileName RECORD [INTO DestItem]  
    [KEY IS KeyName]  
    [INVALID KEY StatementBlock ]  
[END – READ]
```

For
Random
access

```
READ FileName NEXT RECORD [INTO DestItem]  
    [AT END StatementBlock ]  
[END – READ]
```

For
Sequential
access

- NEXT phrase is optional for SEQUENTIAL access mode & a must for DYNAMIC
- KeyName is either a RECORD KEY or ALTERNATE RECORD KEY of the Indexed file. Default is RECORD KEY
- When dynamic access is specified, this key of reference is used for subsequent executions of sequential READ
- Leads to INVALID KEY if a record does not exist with the key specified

Relative Files - READ

```
READ FileName RECORD [INTO DestItem]  
      [INVALID KEY StatementBlock ]  
[END – READ]
```

For
Random access

```
READ FileName NEXT RECORD [INTO DestItem]  
      [AT END StatementBlock ]  
[END – READ]
```

For
Sequential access

- NEXT phrase is optional for SEQUENTIAL access mode & a must for DYNAMIC
- Leads to INVALID KEY if a record does not exist in the position indicated by the Relative key specified

Indexed / Relative Files - Write

```
WRITE RecName [FROM SourceItem]  
[INVALID KEY StatementBlock]  
[END – WRITE]
```

- Writes a new record based on the key value
- INVALID KEY Condition arises when
 - Attempt to write beyond file boundary
 - Record with specified key already present in file
 - Record already exists in the position indicated by RELATIVE KEY for relative files

Indexed / Relative Files - Rewrite

REWRITE RecName [**FROM** SourceItem]

[**INVALID KEY** StatementBlock]

[**END – REWRITE**]

- Overwrites a record based on the RECORD KEY / RELATIVE KEY
- Invalid Key arises when
 - Record does not exist with the RECORD KEY / RELATIVE KEY value specified
 - Duplicate ALTERNATE RECORD KEY when duplicates are not allowed

Indexed / Relative Files - DELETE

```
DELETE FileName RECORD
  [INVALID KEY StatementBlock]
END – DELETE
```

- Removes a record from an indexed / relative file.
- For indexed files, the key can then be reused for record addition.
- For relative files, the space is then available for a new record with the same RELATIVE KEY value.

Indexed / Relative Files - START

- Enables the positioning of the pointer at a specific point in an indexed or relative file
- File should be opened in Input or I-O mode
- Access mode must be Sequential or Dynamic
- Does not read the record

```
START FileName [KEY {  
    IS EQUAL TO  
    IS =  
    IS GREATER THAN  
    IS >  
    IS NOT LESS THAN  
    IS NOT < } KeyName ]  
[INVALID KEY StatementBlock]  
END – START
```

CALL & COPY

Sub Programming

- CALL statement is used to call one program from the other
- Transfers control from one object program to another within the run unit.
- Control returns to the main program after completion of called program
- Parameters can be sent to called program:
 - By reference
 - By content
- Called program must not execute a CALL statement that directly or indirectly calls the calling program (Recursion not allowed)

CALL Statement

Format :

```
CALL {ProgNameIdentifier  
ProgNameLiteral}  
  
[ USING {[ [BY REFERENCE] }  
{ [BY CONTENT] } ParamIdentifier ... } ...]  
  
[ { ON { EXCEPTION  
OVERFLOW } StatementBlock }  
{ NOT ON EXCEPTION StatementBlock } ] [ END - CALL ]
```

CALL Statement

- If the CALL passes parameters, then the called program must have a USING phrase after the PROCEDURE DIVISION header and a LINKAGE SECTION to describe the parameters passed.

The CALL statement has a USING phrase only if a USING phrase is used in the PROCEDURE DIVISION header of the called program.

- Both USING phrases must have the same number of parameters.
- COBOL does not check the type of the parameters passed to a called program. It is the programmer's responsibility to make sure that only parameters of the correct type and size are passed.
- Parameters passed from the calling program to the called program correspond by position, not by name. That is, the first parameter in the USING phrase of the CALL corresponds to the first in the USING phase of the called program, and so on.

CALL Statement

- If the program being called has not been linked (does not exist in the executable image,) the statement block following the ON EXCEPTION/OVERFLOW will execute. Otherwise, the program will terminate abnormally.
- In standard COBOL, the CALL verb has two parameter passing mechanisms
 - **BY REFERENCE** is used when the called program needs to pass data back to the caller
 - **BY CONTENT** is used when data needs to be passed to, but not received from, the called program
- BY REFERENCE is the default passing mechanism, and so is sometimes omitted
- To terminate the called program EXIT PROGRAM statement is used.
- Using STOP RUN stops the whole run unit

CALL Statement

Example :

```
CALL "DateValidate"  
      USING BY CONTENT TempDate  
      USING BY REFERENCE DateCheckResult.
```

```
IDENTIFICATION DIVISION.  
PROGRAM-ID DateValidate.  
DATA DIVISION.  
WORKING-STORAGE SECTION.
```

```
LINKAGE SECTION.  
01 DateParam          PIC X(8).  
01 DateResult          PIC 9.
```

```
PROCEDURE DIVISION USING DateParam, DateResult.  
Begin.
```

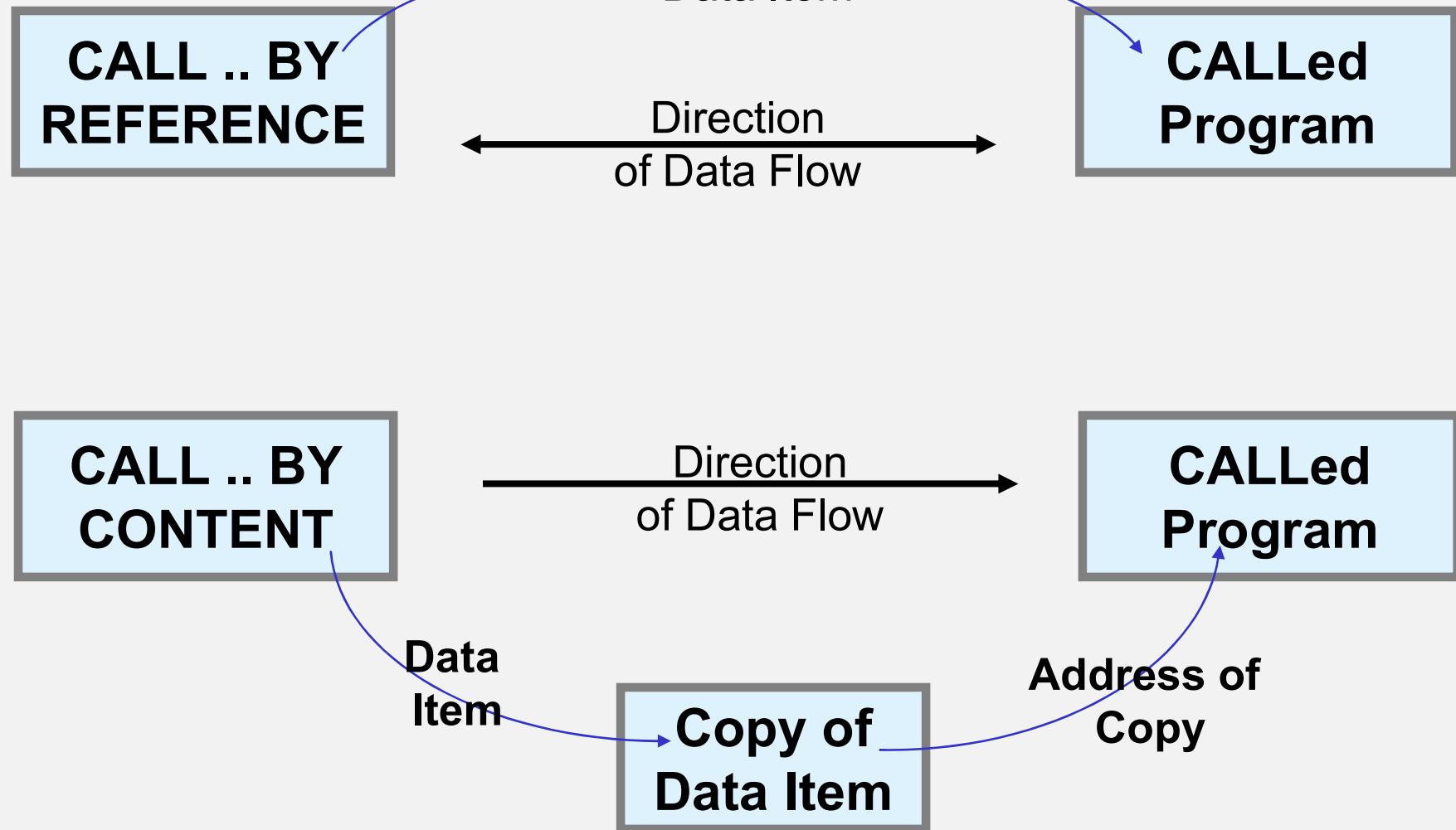
```
    ? ? ? ? ? ? ? ? ? ?
```

```
    ? ? ? ? ? ? ? ? ? ?
```

```
    EXIT PROGRAM.
```

Parameter Passing Mechanisms

Example :



Sub Programs

EXIT Program

- Specifies the end of a called program and returns control to the calling program.
- When no CALL statement is active, control passes through the exit point to the next executable statement.
- When there is no next executable statement in a called program, an implicit EXIT PROGRAM statement is executed.

Compiler Directives

COPY

- The COPY statement is a library statement that places prewritten text in a COBOL program.
- Prewritten source program entries can be included in a source program at compile time.
- Usage:- For standard file descriptions, record descriptions, or procedures. These entries and procedures can be saved in user-created libraries; they can then be included in the source program by means of the COPY statement.
- Helps avoid repeated coding in different programs

DB2



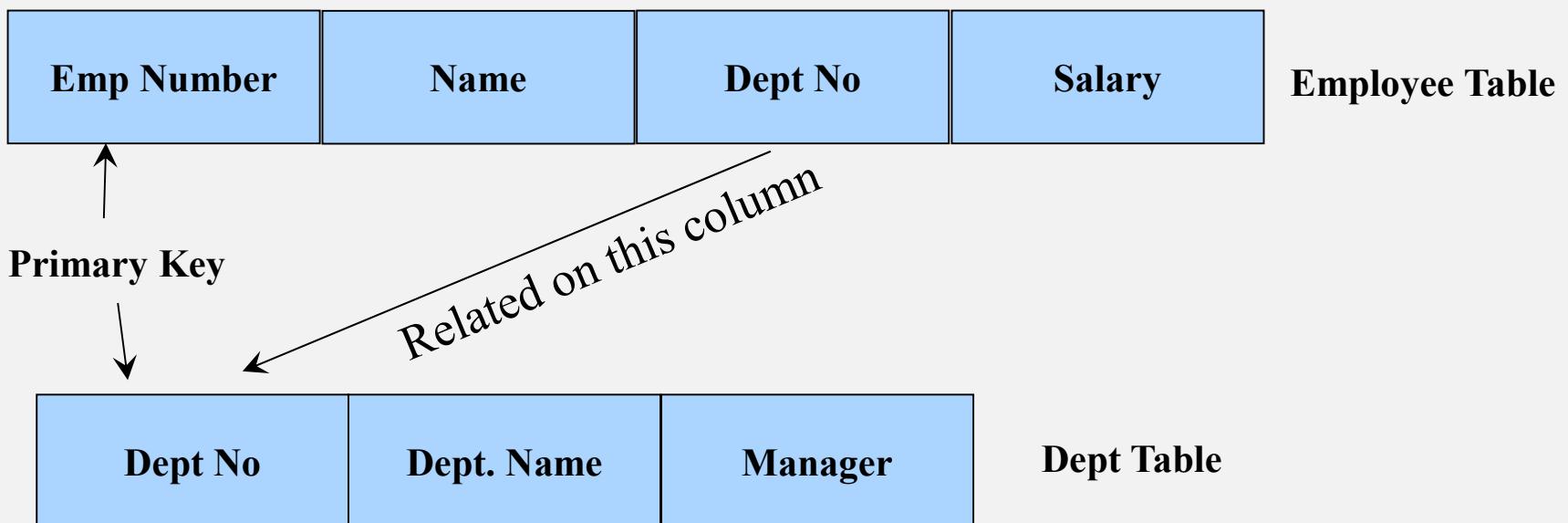
Database Concepts

- A data base is a way of organizing data and software, to solve some of the problems found in conventional file processing such as
 - Duplication of data and effort.
 - Programs depending on the exact file structure.
 - People looking at files to which they should not have access.
 - Programs needs to be changed when the data is organized differently.
- Data bases are used to meet 4 main data related goals
 - Increase Data Independence
 - Reduce Data Redundancy
 - Increase Data Security
 - Maintain Data Integrity

Relational Database

In a Relational Database --

- Data is represented in the form of Table (Rows/Cols)
- Records are referred as Rows
- Fields are referred as Columns
- Based on Relational Algebra
- Columns in one table refer columns in some other table



Relational database - Tables

Employee Table

EMP NO	NAME	SALARY	Dept No
1111	Srihari	400000	D1
1112	Srilatha	280000	D2
1113	Vivek	300000	D2
1114	Ramesh	700000	D1
1115	John	300000	D3

Dept Table

Dept No	DEPTNAME
D1	H R
D2	Admin
D3	Finance

Both the above tables are related on 'Dept No' column

Data Redundancy

Redundant Data

StaffDept Table

StaffId	Name	Salary	DeptNo	Dept Name	Dept Location
1111	Srihari	400000	D1	H R	Cyber Towers
1112	Srilatha	280000	D2	Admin	Cyber Pearl
1113	Vivek	300000	D2	Admin	Cyber Pearl
1114	Ramesh	700000	D1	H R	Cyber Towers
1115	John	300000	D3	Finance	Cyber Gateway

Staff Table

StaffId	Name	Salary	Dept No
1111	Srihari	400000	D1
1112	Srilatha	280000	D2
1113	Vivek	300000	D2
1114	Ramesh	700000	D1
1115	John	300000	D3

Dept Table

DeptNo	DeptName	DeptLocation
D1	H R	Cyber Towers
D2	Admin	Cyber Pearl
D3	Finance	Cyber Gateway

Types of Integrity

Entity Integrity

- Rule states that no column that is part of a primary key can have a null value

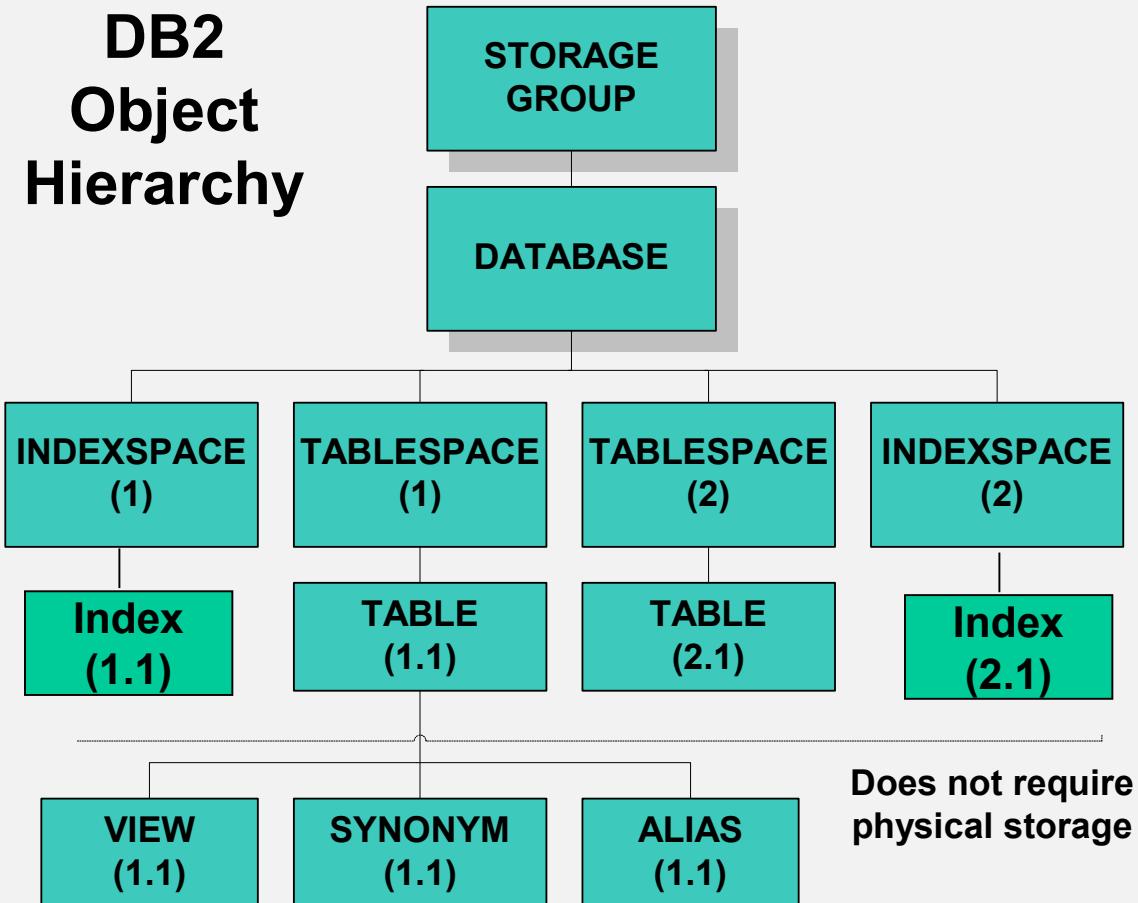
Referential Integrity

- Rule states that every foreign key in the first table must either match a primary key value in the second table or must be wholly null

Domain Integrity

- Integrity of information allowed in column

DB2 Objects



DB2 objects

Storage Group

- It is a collection of DASD volumes on which VSAM datasets can be allocated for associated storage structures
- All volumes of a storage group must have same device type
- The VSAM datasets can be managed either by the User or Storage Management Subsystem (SMS)

Database

- It is a collection of logically related objects - like Tablespaces, Indexspaces and data contained in them.
- Stogroup and user-defined VSAM are the two storage allocations for a DB2 database definition.
- In a given database, all the spaces need not have the same storage group
- A database holds no data of its own but just exists to group DB2 objects

DB2 objects

Tablespace

- Logical address space on secondary storage to hold one or more tables
- A 'SPACE' is basically an extendable collection of pages with each page of size 4K or 32K bytes.
- It is the storage unit for recovery and reorganizing purpose

Tables

- Structures to store data and a combination of rows and columns
- Every table defined with column description and constraint description
- Constraints help maintain data integrity

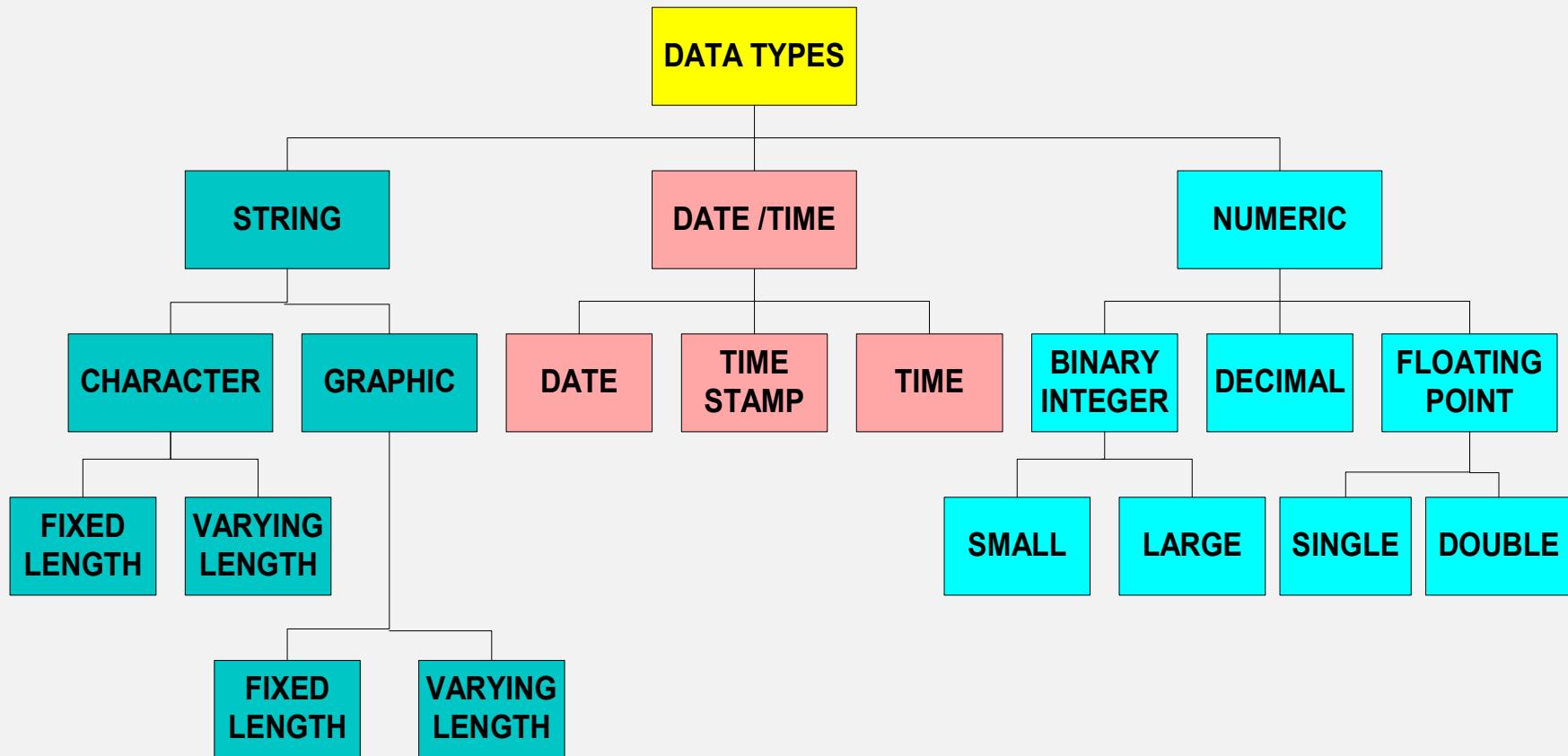
Views

- A subset of columns created as a virtual table
- A view can include all or some columns from one or more tables

Indexes

- Index created for faster access of data in a table
- Used to enhance performance

DB2 Data Types



DB2 Data Types – Non numeric

CHAR(n)

- Fixed-length character strings with a length of ‘n’ bytes.
- ‘n’ must be greater than 0 and less than 255 bytes
- Constant value to be enclosed in single quotes

VARCHAR(n)

- Varying-length character strings with a maximum length of n bytes
- ‘n’ must be greater than 0 and less than a number that depends on the pagesize of the table space(4K or 32K)
- Constant value to be enclosed in single quotes

DB2 Data Types - Date & Time

DATE

- A Date is a three-part value representing a Year, Month, and Day in the range '0001-01-01' to '9999-12-31'
- Stored as a 4 byte number internally
- Date constant format : 'YYYY-MM-DD'

TIME

- A time is a three-part value representing a time of day in hours, minutes, and seconds, in the range '00.00.00' to '23.59.59'
- Time constant format : 'HH.MM.SS'

TIMESTAMP

- A timestamp is a seven-part value representing a date and time by year, month, day, hour, minute, second, and microsecond
- The range is '0001-01-01-00.00.000000' to '9999-12-31-24.00.00.000000'
- Timestamp constant format : 'YYYY-MM-DD-HH.MM.SS.ssssss'

DB2 Data Types - Numeric

SMALLINT

- A small integer is a two-byte binary integer of 16 bits
- The range is -32768 to +32767

INTEGER or INT

- A large integer is full word binary integer of 32 bits
- The range is -2147483648 to +2147483647

REAL or FLOAT(n)

- Single / Double precision floating-point numbers
- Single precision - 'n' must be in the range 1 through 21 (internal size 4 bytes)
- Double precision - 'n' must be in the range 22 through 53 (internal size 8 bytes)

DECIMAL(p,s) or DEC(p,s) or NUMERIC(p,s)

- Packed decimal numbers with precision 'p' and scale 's'
- The precision 'p' is the total number of digits including the digits following the decimal point
- 'p' must be greater than 0 and less than 32
- The scale s, is the number of digits in the fractional part of the number
- 's' must be greater than or equal to 0 and less than or equal to the precision

DB2 Data types : Default Values

Data Type	Default Value
Numeric	Zero
Fixed length String	Blanks
Varying length String	String of length Zero
Date	Current Date
Time	Current Time
Time Stamp	Current Timestamp

DB2 Data types : Null Value

- Null - missing or unknown information.
- Either the Column does not apply for the row (inapplicable data) or the value does not exist currently (unknown data)..
- Every column is NULLable by default
- nulls can be prohibited by specifying NOT NULL or NOT NULL WITH DEFAULT during column definition

Expressions

- **Numeric expressions**

Numeric expressions can have the symbols + - * / ()

- Examples :

```
SELECT BONUS+COMM    VARIABLE_COMP  
FROM EMPLOYEE  
WHERE DEPTNO = 'A00'
```

```
SELECT EMPNO, (SALARY/12)  MONTHLY_GROSS  
FROM EMPLOYEE  
WHERE DEPTNO = 'A00'
```

Expressions

- **String expressions**
 - Only operator allowed is `||` which is known as concatenation operator
 - Example :

```
SELECT FIRSTNAME || ' ' || LASTNAME NAME  
FROM EMPLOYEE
```

Expressions

- **Date / Time expressions**

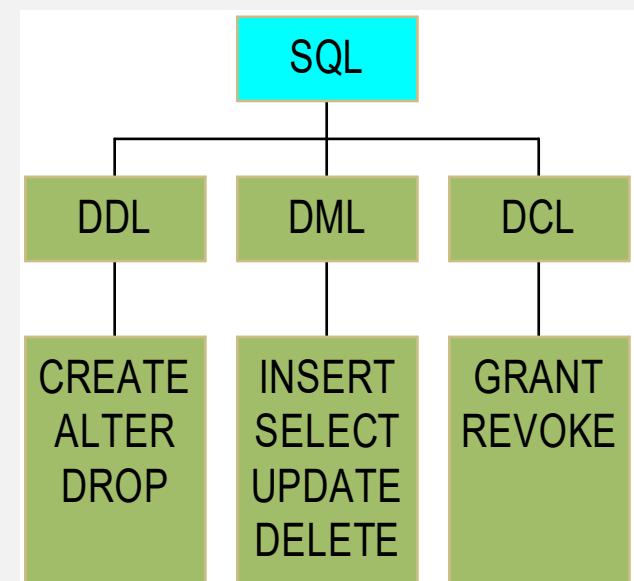
- date + duration = date
 - time + duration = time
 - timestamp + duration = timestamp
 - date – duration = date
 - time – duration = time
 - timestamp – duration = timestamp
 - date – date = duration 8 digits yyyyymmdd
 - time – time = duration 6 digits hhmmss
-
- Duration can be n YEAR, n MONTH, n DAY, n HOUR, n MINUTE etc
 - Example :

```
SELECT * from EMPLOYEE  
where DOJ > CURRENT DATE – 1 year – 2 month
```

Structured Query Language

Introduction to SQL

- Structured Query Language (SQL) is a standard language for defining and manipulating data in a relational database
- SQL specifies what data to retrieve or manipulate but does not specify how to accomplish these tasks (non-procedural language)
- The DBMS analyzes SQL and formulates data -navigational instructions called *access paths*
- Statement Categories
 - **DDL** (Data Definition Language)
 - ✓ To manipulate database structures
 - **DML** (Data Manipulation Language)
 - ✓ To manipulate data
 - **DCL** (Data Control Language)
 - ✓ To manipulate access control



DB2 SQL : Tools

Query Management Facility (QMF)

- QMF is an IBM database management tool that allows interactive query and report writing support
- It allows end users to enter SQL queries to produce a variety of reports and graphs as a result of this query

SQL Processing Using File Input (SPUFI)

- One can execute SQL statements at a TSO terminal using the SPUFI facility
- The SQL statements that require processing are written in the input data set
- SPUFI passes the input data set to DB2 for processing
- DB2 executes the SQL statement in the input data set and provides the results in the output data set
- The output sequential data set contains these items for each SQL statement that DB2 executes:
 - The executed SQL statement, copied from the input data set
 - The results of executing the SQL statement
 - SQLCODE and error messages to indicate the status of execution of the statement

CREATE TABLE

SQL statement to create a table

```
CREATE TABLE <Table_name>
    [( Column definitions , Constraint specifications )]
    [ LIKE Table name / View name ]
    IN Database.Tablespace
```

Constraint specification format :

CONSTRAINT <constraint name>

- PRIMARY KEY (column)
- FOREIGN KEY (column) REFERENCES
tablename(columnName) [ON DELETE CASCADE / SET NULL]
- UNIQUE(column)
- CHECK(condition)

Constraints can be specified at column level also with format :

{ PRIMARY KEY / REFERENCES tablename.column name / UNIQUE / NOT NULL }

CREATE TABLE : Example

```
CREATE TABLE EMPLOYEE (
```

EMPNO	CHAR(5) NOT NULL PRIMARY KEY,
FIRSTNAME	VARCHAR(60) DEFAULT,
LASTNAME	VARCHAR(60) DEFAULT,
DOB	DATE NOT NULL,
DEPTNO	CHAR(3) REFERENCES DEPT(DEPTNO),
SALARY	DECIMAL(8,2) NOT NULL,
GENDER	CHAR(1) CHECK (GENDER IN ('M','F')),
HIREDATE	DATE DEFAULT,
CONSTRAINT SALCHK CHECK (SALARY > 20000)	

```
) in MSDB.MSTS
```

ALTER TABLE

- Statement used to modify table structure
- Column type can be changed to other compatible type
- Not possible to drop a column

```
ALTER TABLE <Table_name >
    ADD COLUMN <column specifications>
    DROP CONSTRAINT <constraint name>
    ALTER COLUMN <column name> SET DATA TYPE <type>
```

- Example :

```
ALTER TABLE Employee
    ADD COLUMN BONUS DECIMAL(7,2)
```

DROP TABLE

- Statement used to drop the table from the database
- Format :

```
DROP TABLE <table name>
```

INSERT

Statement used to insert one row (format 1) or many rows (format2) in a table

Format 1 :

```
INSERT INTO Tablename [ (Column List) ]  
VALUES( value1, value2, value3 ,.....)
```

Format 2 :

```
INSERT INTO Tablename [ (Column List) ] <Select statement>
```

- Insert fails if value omitted for a column with NOT NULL constraint
- If the column is NOT NULL DEFAULT it is set to that default value if value not specified
- If Column list is omitted, values are required for all columns in the order

INSERT

Example 1 :

```
INSERT INTO EMPLOYEE
```

```
VALUES ('A123', 'Ramesh', 'Kumar', '1974-10-21', 'D12', 12500, 'M', '2008-02-26' )
```

Values supplied
to all columns.
Hence column list
not required)

Column list required
as only few values
are provided.

Example 2 :

```
INSERT INTO EMPLOYEE (EMPNO, FIRSTNAME, LASTNAME, DOB, SALARY)
```

```
VALUES ( 'B234', 'Ramu', 'Sharma', '1978-02-02', 23000)
```

SELECT

- Statement to extract data from tables
- Also referred as QUERY
- Simple SELECT format

SELECT < * / columns / expressions>

FROM <table_name>

[WHERE <condition>] - optional

[ORDER BY <sort_order>] - optional

- Examples :

Select all
columns

ALIAS for SALARY * 0.1

SELECT * FROM EMPLOYEE

ORDER BY FIRSTNAME

SELECT EMPNO,FIRSTNAME, SALARY * 0.1 ALLOWANCE FROM EMPLOYEE

SELECT EMPNO, DOB, SALARY FROM EMP

WHERE SALARY > 30000

Conditions in WHERE clause

Several types of conditions in WHERE clause with examples

- Simple conditions using relational operators

= EQUAL TO

<> NOT EQUAL TO

< LESS THAN

> GREATER THAN

<= LESS THAN OR EQUAL TO

>= GREATER THAN OR EQUAL TO

```
SELECT * FROM EMPLOYEE WHERE SALRY > 50000
```

- BETWEEN / NOT BETWEEN (check for range of values)

```
SELECT * FROM EMPLOYEE WHERE SALRY BETWEEN 20000 AND 30000
```

```
SELECT NAME FROM EMPLOYEE WHERE DOB BETWEEN '1976-01-01' AND '1976-12-31'
```

Conditions in WHERE clause

- IN / NOT IN (check with a list of values)

```
SELECT * FROM EMPLOYEE WHERE DEPTNO IN ('A01', 'B02', 'C11')
```

```
SELECT * FROM STUDENT WHERE CLASS NOT IN (3, 5, 7, 8)
```

- LIKE / NOT LIKE
 - used for pattern matching
 - % matches any pattern
 - _ matches one character

```
SELECT * FROM EMPLOYEE WHERE FIRSTNAME LIKE 'AK%'
```

```
SELECT * FROM EMPLOYEE WHERE DEPTNO LIKE 'B_6'
```

```
SELECT * FROM EMPLOYEE WHERE NAME LIKE 'B%RAO'
```

Conditions in WHERE clause

- IS [NOT] NULL
 - Relational operators cannot be used to check null values
 - For example 'WHERE SALARY = NULL' is invalid as value cannot be compared with NULL
 - IS NULL or IS NOT NULL is used to check null values

```
SELECT * FROM EMPLOYEE WHERE DOB IS NULL
```

```
SELECT * FROM STUDENT WHERE NAME IS NOT NULL
```

- Logical grouping of conditions
 - All above conditions can be combined with operators AND, OR, NOT
 - Order of precedence – NOT, AND, OR
 - Use brackets to override precedence

```
SELECT * FROM EMPLOYEE WHERE SALARY > 20000 AND GENDER = 'M'
```

```
SELECT * FROM EMPLOYEE WHERE DEPTNO = 'A11' OR SALARY > 30000
```

ORDER BY

- This clause is used to sort the data in a specific order
- Format

ORDER BY COLUMN / expression [ASC / DESC]

- Ascending order is default
- Multiple columns can be used for the order
- Null values appear last in ascending sort and first in descending sort
- Examples :

SELECT * FROM EMPLOYEE ORDER BY GENDER

SELECT * FROM EMPLOYEE WHERE GENDER = 'M'

ORDER BY SALARY DESC

SELECT * FROM EMPLOYEE

ORDER BY DEPTNO DESC , NAME ASC

Scalar functions

- A scalar function is applied to a column or expression and operate on one row at a time
- These functions can be used where ever expressions are allowed (SELECT , WHERE etc)
- The argument of a scalar function can be a function
- Return value of the function is used in the corresponding operation
- Examples :

```
SELECT MONTH(DOB) FROM EMPLOYEE
```

```
SELECT * FROM EMPLOYEE WHERE SUBSTR(NAME, 3,1) = 'H'
```

```
SELECT EMPNO, NAME, DECIMAL(SALARY, 8, 2) SAL FROM EMPLOYEE
```

Scalar functions – string type

DIGITS(number)

- converts a numeric value to a character string value. Removes sign and decimal point
- Example :

Function	Return value	Return type
DIGITS(-345.67)	'34567'	string

UPPER(string) / LOWER(string)

- Converts string into uppercase / lower case
- Example :

Function	Return value	Return type
UPPER('abcd1234x+y')	'ABCD1234X+Y'	string
LOWER('ABCDefGH234')	'abcdefgh234'	string

Scalar functions – string type

LEFT(string , number) / RIGHT(string , number)

- Returns specified no of characters from left / right side
- Example :

Function	Return value	Return type
LEFT('abcd123', 4)	'abcd'	string
RIGHT('abcd123', 4)	'd123'	string

SUBSTR(string, start, length)

- Returns no of characters (length) from a given position(start) of a string
- If length is omitted, takes till the end of the string
- Example :

Function	Return value	Return type
SUBSTR('KUMARAN', 2, 3)	'UMA'	string
SUBSTR('KUMARAN', 2)	'UMARAN'	string

Scalar functions – string type

LENGTH(string)

- Returns length of the string
- Example :

Function	Return value	Return type
LENGTH('abcd123')	7	numeric

LOCATE(string1, string2)

- searches for the string defined as first parameter (string1) within the source string defined as second parameter(string2) and returns the position. Returns 0 (zero) if the string not found
- Example :

Function	Return value	Return type
LOCATE('AB', 'VTDERABUKH')	6	numeric
LOCATE('X', 'VTDERABUKH')	0	numeric

Scalar functions – string type

LTRIM(string) / RTRIM(string)

- Returns a string after removing spaces on left side / right side
- Example :

Function	Return value	Return type
LTRIM(' abcd123')	'abcd123'	string
RTRIM('ab cd ')	'ab cd'	string

SPACE(number)

- Returns a string of spaces.
- Example :

Function	Return value	Return type
SPACE(10)	' '	string

Scalar functions – string type

CHAR(number)

CHAR(date/ time , format)

- Converts a numeric value / date / time into a string.
- Date / Time will be converted into a string with the format specified
- Formats for date / time
 - ISO yyyy-mm-dd hh.mm.ss
 - USA mm/dd/yyyy hh.mm AM/PM
 - EUR dd.mm.yyyy hh.mm.ss
- Example : (dob assumed as 12 August 1978)

Function	Return value	Return type
CHAR(345.67)	'345.67'	string
CHAR(dob, EUR)	'12.08.1978'	string
CHAR(dob, USA)	'08/12/1978'	string

Scalar functions – numeric type

ABS(number)

- Returns the absolute value of the number
- Example :

Function	Return value	Return type
ABS(3546)	3546	numeric
ABS(-3546)	3546	numeric

MOD(number, number)

- Calculates a remainder when the first parameter is divided by the second
- Example :

Function	Return value	Return type
MOD(28, 3)	1	numeric

Scalar functions – numeric type

CEILING(number) / FLOOR(number)

- CEILING function rounds the supplied value to the next higher integer value.
- FLOOR function rounds to the first integer value that is less than or equal to the supplied value.
- Example :

Function	Return value	Return type
CEILING(35.67)	36	numeric
FLOOR(35.67)	35	numeric

ROUND(number1, number2)

- Returns the number1 to number2 places to the right of the decimal point if number2 is positive or to the left of the decimal point if number2 is zero or negative
- Example :

Function	Return value	Return type
ROUND(276.379, 2)	276.38	numeric
ROUND(276.679, 0)	277	numeric
ROUND(276.679, -2)	300	numeric

Scalar functions – numeric type

SIGN(number)

- Returns the indicator of the sign of the number (-1 if negative, 1 if positive, 0 if zero)
- Example :

Function	Return value	Return type
SIGN(-2341)	-1	numeric
SIGN(35.67)	1	numeric

INT / INTEGER (string / number)

- returns an integer representation of a number or a string representation of a number (decimal, integer, float)
- Example :

Function	Return value	Return type
INT('2386.45')	2386	numeric
INT(45.67)	45	numeric
INTEGER('356')	356	numeric

Scalar functions – date / time type

DAY(date) / MONTH(date) / YEAR(date)

- Returns day / month / year of the given date as a number
- Example (dob assumed as 23 september 1986) :

Function	Return value	Return type
DAY(dob)	23	numeric
MONTH(dob)	9	numeric
YEAR(dob)	1986	numeric

HOUR(time) / MINUTE(time) / SECOND(time)

- Returns hour / minute / second of the given time or timestamp as a number
- Example (tod assumed as 10:34:21):

Function	Return value	Return type
HOUR(tod)	10	numeric
MINUTE(tod)	34	numeric
SECOND(tod)	21	numeric

Scalar functions – date / time type

DAYOFWEEK(date)

- returns an integer between 1 and 7 that represents the day of the week, where 1 is Monday and 7 is Sunday.

DAYOFYEAR(date)

- returns an integer between 1 and 366 that represents the day of the year where 1 is January 1

DATE(string)

- Converts string in default date format into date (return type is date)

CURRENT DATE

- Not a function. This is a special register representing current system date

CURRENT TIME

- Not a function. This is a special register representing current system time

Group functions

- Group functions are also known as Column functions and Aggregate functions
- A group function applies on a group of rows
- These functions cannot be mixed with row values (columns, expressions or scalar functions)
- By default, whole table is treated as a single group
- GROUP BY clause can be used to divide the table into multiple groups based on values of a column
- For example, table can be divided into multiple groups for all departments to calculate total salary for each department

Group functions

- Most commonly used group functions :

SUM(column)

- Returns the total value

MIN(column)

- Returns the minimum value

AVG(column)

- Returns the average value

MAX(column)

- Returns the maximum value

COUNT(* / column)

- Returns the number of rows

- The result of any column function (except the COUNT function) will be of same data type as the column to which it was applied
- The result of any column function (except the COUNT function) can be null
- When using group functions on nullable column, all rows with null in the column are eliminated before applying the function
- Examples :

```
SELECT SUM(SALARY) FROM EMPLOYEE
```

```
SELECT COUNT(*) FROM EMPLOYEE WHERE SALARY IS NULL
```

```
SELECT MAX(SALARY), MIN(SALARY) FROM EMPLOYEE
```

GROUP BY / HAVING clauses

- Use GROUP BY to group rows by the values of one or more columns
 - Only columns named in the GROUP BY clause are allowed in SELECT list besides group functions
-
- Use HAVING to specify a search condition that each retrieved group must satisfy
 - The HAVING clause acts like a WHERE clause for groups, and can contain the same kind of conditions that can be specified in a WHERE clause
 - Only group values can be used in condition in HAVING clause
 - Format :

```
SELECT group values / group functions  
FROM <table_name>  
[ WHERE <condition> ]  
GROUP BY columns  
HAVING condition
```

GROUP BY / HAVING clauses

- Examples :

```
SELECT DEPTNO, SUM(SALARY) FROM EMPLOYEE  
GROUP BY DEPTNO
```

```
SELECT GENDER, MAX(SALARY), MIN(SALARY) FROM EMPLOYEE  
GROUP BY GENDER
```

```
SELECT DEPTNO,COUNT(*) FROM EMPLOYEE  
WHERE SALARY < 5000  
GROUP BY DEPTNO
```

```
SELECT DEPTNO, SUM(SALARY) FROM EMPLOYEE  
GROUP BY DEPTNO  
HAVING COUNT(*) > 5
```

Duplicates Elimination

- Duplicates are eliminated using DISTINCT
- Example :

```
SELECT DISTINCT DEPTNO FROM EMPLOYEE
```

- In the above example, if DISTINCT is not used, there will be DEPTNO repeating number of times.
- With DISTINCT each value will occur only once

UPDATE TABLE

- Updates the values of columns with new values in one or many rows
- Rows selected for updation depend on condition in WHERE clause
- If WHERE clause is omitted all rows will be updated
- Format :

```
UPDATE tablename  
SET Columnname = expression .....  
WHERE condition
```

- Example :

```
UPDATE EMPLOYEE  
SET SALARY = 50000, NAME = 'Ramesh'  
WHERE EMPNO = '1234'
```

DELETE

- DELETE statement can be used to delete rows from the table
- Which rows to be deleted depends on the condition in WHERE clause
- If WHERE clause is omitted all rows will be deleted leaving only structure of the table
- Format :

```
DELETE FROM Tablename
```

```
WHERE condition
```

- Example :

```
DELETE FROM EMPLOYEE
```

```
WHERE SALARY IS NULL
```

Joins

Joins

- Join is a concept to retrieve data from more than one table
- SELECT statement is used to select data from different tables which are joined on a given condition
- For example, Employee details from EMPLOYEE table along with department details from DEPARTMENT table can be extracted in single SELECT statement by joining both the tables matching department numbers in both the tables
- The types of join supported in DB2 are –
 - Inner Join
 - Left outer Join
 - Right Outer Join
 - Full outer Join

Joins

- General format for join :

List of columns selected from both the tables

SELECT X.column3, X.column4, Y.column1, Y.column3

FROM tableA X { INNER JOIN
LEFT OUTER JOIN
RIGHT OUTER JOIN
FULL OUTER JOIN } tableB Y

ON X.column1 = Y.column2

JOIN condition to match columns in the tables

- Type of join.
- If omitted (placing comma between table names), INNER JOIN is assumed

Joins

Inner Join

- If two tables are involved in a join operation then for all possible combinations of rows, a row of first table is paired with a matching row of second table provided the join condition is true
- Inner join is default

Left Outer Join

- Selects all the rows from the table on the left side
- Selects only matching rows from the table on the right side

Right Outer Join

- Selects all the rows from the table on the right side
- Selects only matching rows from the table on the left side

Full Outer Join

- Includes all the rows from both the tables

Inner Join example

Employee table

EmpNo	Name	Salary	DeptNo
1111	Srihari	40000	D1
1112	Srilatha	28000	D2
1113	Vivek	30000	null
1114	Ramesh	70000	D1
1115	John	30000	null

Dept table

DeptNo	DeptName
D1	H R
D2	Admin
D3	Finance

Table alias used only if column names are same in both the tables

Query

```
SELECT EmpNo, Name, Salary, X.DeptNo, DeptName  
FROM Employee X INNER JOIN Dept Y  
ON X.DeptNo = Y.DeptNo
```

Output

EmpNo	Name	Salary	DeptNo	DeptName
1111	Srihari	40000	D1	H R
1112	Srilatha	28000	D2	Admin
1114	Ramesh	70000	D1	H R

Left Outer Join example

Employee table

EmpNo	Name	Salary	DeptNo
1111	Srihari	40000	D1
1112	Srilatha	28000	D2
1113	Vivek	30000	null
1114	Ramesh	70000	D1
1115	John	30000	null

Dept table

DeptNo	DeptName
D1	H R
D2	Admin
D3	Finance

Query

```
SELECT EmpNo, Name, Salary, X.DeptNo, DeptName  
FROM Employee X LEFT OUTER JOIN Dept Y  
ON X.DeptNo = Y.DeptNo
```

Output

EmpNo	Name	Salary	DeptNo	DeptName
1111	Srihari	40000	D1	H R
1112	Srilatha	28000	D2	Admin
1113	Vivek	30000	null	null
1114	Ramesh	70000	D1	H R
1115	John	30000	null	null

Right Outer Join example

Employee table

EmpNo	Name	Salary	DeptNo
1111	Srihari	40000	D1
1112	Srilatha	28000	D2
1113	Vivek	30000	null
1114	Ramesh	70000	D1
1115	John	30000	null

Dept table

DeptNo	DeptName
D1	H R
D2	Admin
D3	Finance

Query

```
SELECT EmpNo, Name, Salary, Y.DeptNo, DeptName  
FROM Employee X RIGHT OUTER JOIN Dept Y  
ON X.DeptNo = Y.DeptNo
```

Output

EmpNo	Name	Salary	DeptNo	DeptName
1111	Srihari	40000	D1	H R
1114	Ramesh	70000	D1	H R
1112	Srilatha	28000	D2	Admin
null	null	null	D3	Finance

Full Outer Join example

Employee table

EmpNo	Name	Salary	DeptNo
1111	Srihari	40000	D1
1112	Srilatha	28000	D2
1113	Vivek	30000	null
1114	Ramesh	70000	D1
1115	John	30000	null

Dept table

DeptNo	DeptName
D1	H R
D2	Admin
D3	Finance

Query

```
SELECT EmpNo, Name, Salary, X.DeptNo, DeptName  
FROM Employee X FULL OUTER JOIN Dept Y  
ON X.DeptNo = Y.DeptNo
```

Output

EmpNo	Name	Salary	DeptNo	DeptName
1111	Srihari	40000	D1	H R
1112	Srilatha	28000	D2	Admin
1113	Vivek	30000	null	null
1114	Ramesh	70000	D1	H R
1115	John	30000	null	null
null	null	null	null	Finance

Joins – Other selection criteria

- WHERE clause can be used to specify other conditions in JOIN
- Example :

```
SELECT EmpNo, Name, Salary, X.DeptNo, DeptName  
FROM Employee X LEFT OUTER JOIN Dept Y  
ON X.DeptNo = Y.DeptNo  
WHERE SALARY > 25000
```

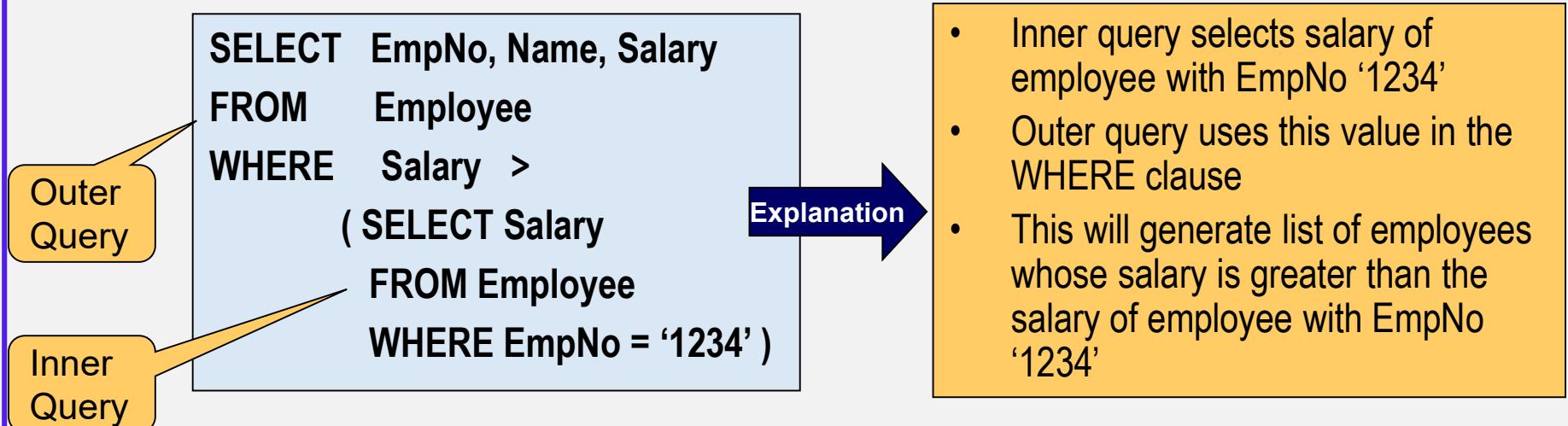
- WHERE clause can be used for join condition also.
- It will be purely inner join
- Example :

```
SELECT EmpNo, Name, Salary, X.DeptNo, DeptName  
FROM Employee X , Dept Y  
WHERE X.DeptNo = Y.DeptNo AND SALARY > 25000
```

Sub Queries

Sub queries

- One or more SELECT statements nested in another SELECT statement is referred as subquery
- A subquery enables a user to base the search criteria of one SELECT statement on the results of another SELECT statement
- Relational operators (<, >, <=, = etc) are used in WHERE clause to compare the result of inner query , if inner query selects only one row
- IN / NOT IN predicates are used if inner query selects more than one row
- Predicates ALL, ANY, SOME can also be used for multiple values in Inner Query
- Example :



Sub queries - Examples

```
SELECT DeptNo, DeptName FROM Dept  
WHERE DeptNo NOT IN  
( SELECT DISTINCT DeptNo  
    FROM EMPLOYEE )
```

This will generate list of departments not having employees

```
SELECT EmpNo, Name, Salary  
FROM Employee  
WHERE Salary > ANY  
( SELECT AVG (Salary)  
    FROM Employee  
    WHERE DeptNo IN ( 'A12', 'B23' )  
    GROUP BY DeptNo )
```

This will generate list of employees whose salary is greater than **any** of average salaries of departments 'A12' and 'B23'

COMMIT / ROLLBACK

- Transaction is a recoverable unit of work, or a group of SQL statements that can be treated as one atomic operation
- This means that all the operations within the group are to be completed (committed) or undone (rolled back), as if they were a single operation
- SQL provides statements to handle transaction
- COMMIT statement is used to make changes permanent
- ROLLBACK statement is used to undo all uncommitted changes
- Format :

COMMIT

ROLLBACK

Application Programming

Application Programming in DB2

- DB2 supports host language interface through which programs can be written in a 3GL (COBOL, C, PL/1 etc)
- Programs in COBOL can be written to handle DB2 data
- SQL statements are embedded in COBOL programs to interact with DB2 database to send (UPDATE, INSERT etc) or receive data (SELECT)
- COBOL variables are used to store data received from DB2 or to send data to DB2
- These variables when used in embedded SQL statements are referred as HOST VARIABLES

Embedded SQL

- The SQL statements that are included in a source program are said to be embedded in the application program
- An embedded statement can be placed anywhere in the application program where a host language statement is allowed
- Every SQL statement is embedded in EXEC SQL – END-EXEC block
- Only one statement is allowed in each block
- Format :

```
EXEC SQL  
SQL statement  
END-EXEC
```

- Pre compiler converts these embedded SQL statements into pure COBOL CALL statements

Host variables

- A host variable is an area of storage allocated by the host language and referenced in an SQL statement
- They are the means of moving data from the program to DB2 and vice versa
- Explicitly declare each host variable in DATA DIVISION before its use in the SQL statement
- When host variables are used in the SQL statements they should be prefixed with a colon (:)
- Example :

```
EXEC SQL
  UPDATE Employee
    SET Salary = :ESAL
    WHERE EmpNo = :ENO
END-EXEC
```

Host variables

- Host variables can be used in
 - INTO Clause of SELECT & FETCH statements
 - As input of SET clause of UPDATE statements
 - AS input for the VALUES clause of INSERT statement
 - In WHERE clause of SELECT, UPDATE & DELETE

SQL Statements

- No change in format for embedded SQL statements except for SELECT
- Host variables are used where ever expressions are required
- SELECT statement comes with INTO clause
- INTO clause is followed by host variable list to store values received from DB2
- **Format :**

Example :

```
SELECT expressions
INTO Host variable list
FROM table name
WHERE condition
```

```
SELECT Name, Salary
INTO :Ename, :Esal
FROM Employee
WHERE EmpNo = :Eno
```

- It is not possible to select more than one row using SELECT statement
- Cursors are used for multiple rows (discussed later)

SQL data types – Host variable mapping

Table showing types of COBOL variables corresponding to DB2 data types

SQL data type	COBOL data type
CHAR(n)	PIC X(n)
VARCHAR(n)	PIC S9(4) COMP PIC X(n)
SMALLINT	PIC S 9(4) COMP
INTEGER	PIC S9(9) COMP
DECIMAL(p, s)	PIC S9(p-s)V9(s) COMP-3
FLOAT(n) n <= 21	COMP-1
FLOAT(n) n > 21	COMP-2
DATE	PIC X(10)
TIME	PIC X(8)
TIMESTAMP	PIC X(26)

DCLGEN

- DCLGEN, the ‘Declarations Generator’ supplied with DB2, produces a DECLARE statement and a list of host variables that can be used in the COBOL program
- The DCLGEN can be invoked from option 8.1 of Primary Option Menu panel
- Issued for a single table
- Prepares the structure of the table in a COBOL copybook (PDS member)
- The copybook contains a ‘SQL DECLARE TABLE’ statement along with a working storage host variable definition for the table
- This copybook is included in COBOL program using format :

```
EXEC SQL
    INCLUDE membername
END-EXEC
```

Sample DCLGEN output

```
EXEC SQL DECLARE EMPLOYEE TABLE
  ( EMPNO          CHAR(6)      NOT NULL,
    NAME           VARCHAR(12)  NOT NULL,
    SALARY         DECIMAL(9, 2),
    DOB            DATE
  )
END-EXEC.

*****
* COBOL DECLARATION FOR TABLE EMPLOYEE *
*****
01 DCLEMPLOYEE .
  10 EEMPNO          PIC X(6) .
  10 ENAME .
    49 ENAME-LEN      PIC S9(4) USAGE COMP .
    49 ENAME-TEXT     PIC X(12) .
  10 ESALARY         PIC S9(7)V9(2) USAGE COMP-3 .
  10 EDOB            PIC X(10) .
```

Embedded SQL - examples

```
MOVE '2345'          TO      EEMPNO
MOVE 'Sridhar'        TO      ENAME-TEXT
MOVE 7                TO      ENAME-LEN
MOVE 25000             TO      ESALARY
MOVE '1987-10-06'     TO      EDOB
```

```
EXEC SQL
```

```
    INSERT INTO EMPLOYEE   VALUES( :EEMPNO, :ENAME, :ESALARY, :EDOB )
END-EXEC
```

```
IF SQLCODE < ZERO
```

```
    DISPLAY 'INSERT failure'
```

```
ELSE
```

```
    EXEC SQL
```

```
        COMMIT
```

```
    END-EXEC
```

```
END-IF
```

Embedded SQL - examples

```
MOVE '3333'          TO      EEMPNO  
MOVE 22000           TO      ESALARY  
  
EXEC SQL  
    UPDATE EMPLOYEE SET SALARY = :ESALARY  
    WHERE EMPNO = :EEMPNO  
END-EXEC  
  
IF SQLCODE < ZERO  
    DISPLAY 'UPDATE failure'  
ELSE  
    EXEC SQL  
        COMMIT  
    END-EXEC  
END-IF
```

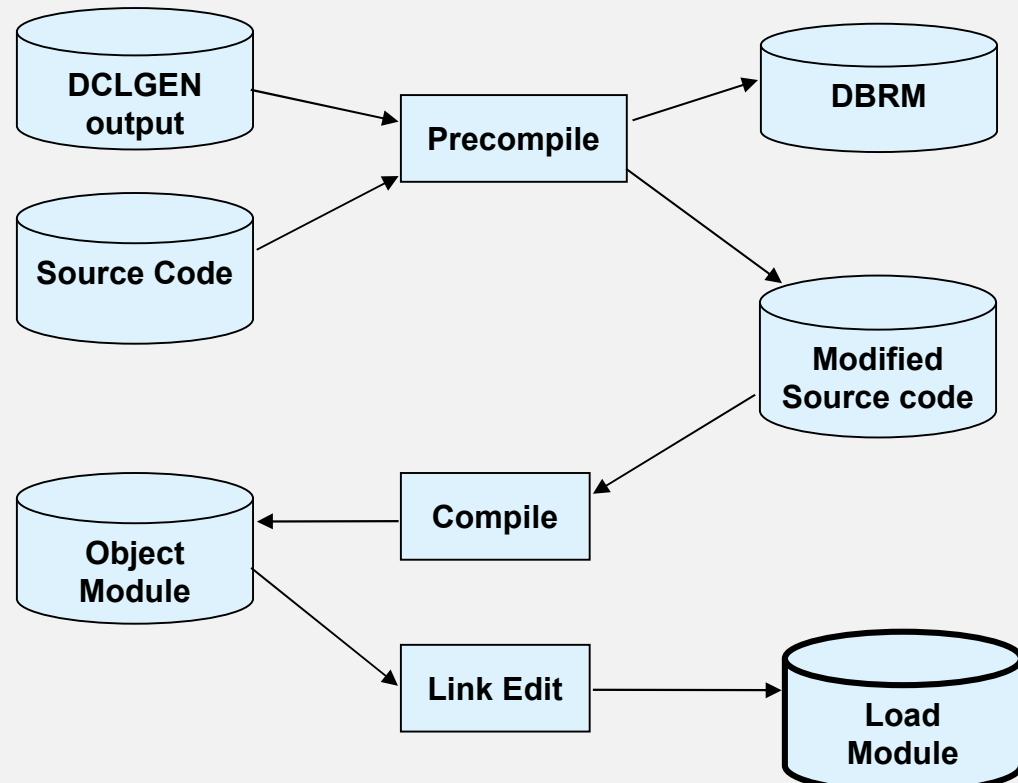
Embedded SQL - examples

```
MOVE '4444'          TO      EEMPNO

EXEC SQL
  SELECT NAME, DOB INTO :ENAME, :EDOB
  FROM EMPLOYEE
  WHERE EMPNO = :EEMPNO
END-EXEC

IF SQLCODE = 100
  DISPLAY 'Invalid Employee Number'
ELSE
  DISPLAY 'Emp Name : '  ENAME-TEXT ( 1 : ENAME-LEN )
  DISPLAY 'Date of Birth : ' EDOB
END-IF
```

Program preparation



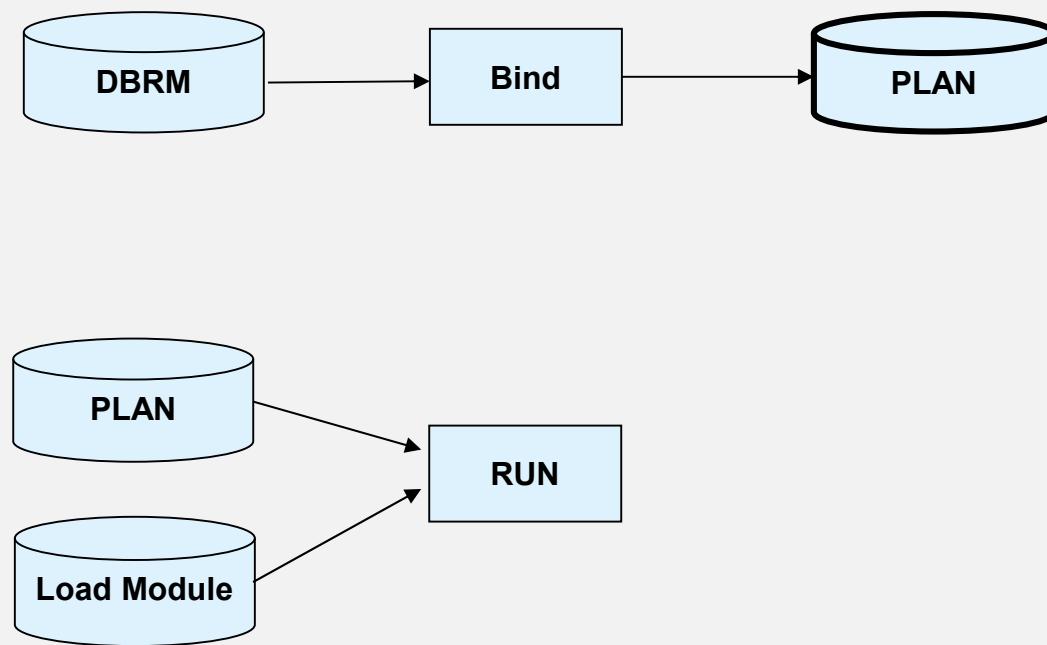
Program preparation – Pre compiler

- DB2 programs must be parsed and modified before normal compilation with the help of Precompiler
- Pre compiler performs the following tasks
 - Searches for and expands DB2 related INCLUDE members
 - checks table and column references against SQL DECLARE TABLE statements in the program
 - Searches for SQL statements in the source code
 - Creates a modified version of source program in which
 - ✓ every SQL statement is commented
 - ✓ CALL statements to the DB2 runtime interface module, along with applicable parameter are added
 - Creates a Data Base Request Module (DBRM) for all the SQL statements of the program

Program preparation – DBRM

- Pre compiler creates DBRM based on the SQL statements in the program
- The DBRM is added as a member in DBRM library
- Contains the SQL statements and host variable information that is extracted from the source program, along with information that identifies the program and ties the DBRM to the translated source statements
- It becomes the input to the bind process to create PLAN in DB2 for the SQL statements

Program preparation



Bind

- It reads the SQL statements from the DBRM and produces a mechanism to access data (in an efficient manner) as directed by the SQL statements being bound
- Checks syntax, checks for correctness of table & column definitions against the catalog information & performs authorization validation
- The output of the BIND PLAN command is an application plan containing executable logic representing optimized access paths to DB2 data
- An application plan is executable only with a corresponding load module

SQL Communication Area

- SQLCA is a list of COBOL variables used for feedback from DB2 after executing one SQL statement
- It is a structure or collection of variables that is updated after execution of each SQL statement
- An application program that contains executable SQL statements must provide exactly one SQLCA.
- This is added to the program with the following statement

```
EXEC SQL
  INCLUDE SQLCA
END-EXEC
```

SQL Communication Area

Structure of the SQLCA (shown partly)

01 SQLCA.

05 SQLCAID	PIC X(8).
05 SQLCABC	PIC S9(9) COMP-4.
05 SQLCODE	PIC S9(9) COMP-4.
05 SQLERRM.	
49 SQLERRML	PIC S9(4) COMP-4.
49 SQLERRMC	PIC X(70).
05 SQLERRP	PIC X(8).
05 SQLERRD	PIC S9(9) COMP-4 OCCURS 6 TIMES.
05 SQLWARN.	
10 SQLWARN0	PIC X.
.....	
10 SQLWARN7	PIC X.

SQL Communication Area

Commonly used variables in COBOL program

- **SQLCODE**

- Used to check error in execution of SQL statement

- Values

- 0 success

- > 0 success with some exception

- +100 Row not found

- < 0 serious error

- **SQLERRD(3)**

- Used to check number of rows affected by INSERT, UPDATE, DELETE etc

Null Indicator Variables

- Every nullable column used in embedded SQL, should be associated with an indicator variable which is used to check for null value
- An indicator variable is a 2-byte integer declared in working-storage section
- When indicator variable is used with any column, the value can be tested for different situations
 - A negative value indicates that the column has been set to null
 - Value is -1 if the column value is null
 - value is -2 if the column has been set to null as a result of data conversion error
 - A positive or zero value indicates that the column is not null
 - A positive value indicates the original length of the data if it is truncated on retrieval

Null Indicator Variables - Example

Declaration

```
01 IND-VARIABLES.  
 02 IND  PIC S9(4) COMP.
```

Usage

```
EXEC SQL  
  SELECT SALARY INTO :ESALARY :IND  
  FROM EMPLOYEE  
  WHERE EMPNO = :EEMPNO  
END-EXEC  
  
IF IND < ZERO  
  DISPLAY 'Salary is null'  
ELSE  
  DISPLAY 'Salary : ' ESALARY  
END-IF
```

Null Indicator Variables

- Null indicator values with corresponding host variables can be used for the following :
 - VALUES clause of the INSERT statement
 - SET clause of the UPDATE statement
 - INTO clause of the SELECT or FETCH statement.
- Failure to check for Null values gives SQLCODE –305

Cursors

- In DB2, an application program uses a cursor to handle multiple rows retrieved from a table or tables
- Cursor allows an application program to retrieve a **set of rows** that satisfy the search condition of an embedded SQL statement
- Program can process the cursor to read current row similar to reading a file
- When a cursor is used, the program can retrieve each row **sequentially** from the cursor until it reaches an end-of-data
- Cursors help Row level processing as against set level processing by SQL
- A program can have several cursors. Each cursor has the following requirements:
 - DECLARE CURSOR statement to define the cursor
 - OPEN and CLOSE statements to open and close the cursor
 - FETCH statement to retrieve rows from the result table of the cursor

Cursors - statements

DECLARE

- It defines the cursor and gives it a name unique to the program in which it is embedded
- It specifies a SELECT statement which defines the criteria for the rows that will make up the result table
- Written in the Working-Storage Section in COBOL
- Format :

```
DECLARE <cursor name> CURSOR  
FOR <Query>
```

Cursors - statements

OPEN

- On execution of OPEN statement, SELECT statement within DECLARE CURSOR is executed and the cursor is created with the set of rows
 - It does not assign values to the host variables
 - This statement is part of PROCEDURE DIVISION
-
- Format :

```
OPEN <cursor name>
```

Cursors - statements

FETCH

- point to the next row in the cursor making it the current row
- extracts the current row contents into the program host variables that are specified on the INTO clause of FETCH
- This statement is part of PROCEDURE DIVISION
- Format :

```
FETCH <cursor name> INTO <host variable list>
```

- The host variables should map properly to the data types of columns in SELECT statement of the cursor
- SQLCODE can be tested for the success of FETCH
- SQLCODE value 100 indicates end of cursor

Cursors - statements

CLOSE

- Closes the cursor and releases the resources assigned to the cursor
- Data will not available for FETCH after CLOSE
- This statement is part of PROCEDURE DIVISION
- Format :

```
CLOSE <cursor name>
```

Cursors - Example

```
EXEC SQL  
    DECLARE EMPCUR CURSOR FOR SELECT EMPNO, NAME FROM EMPLOYEE  
END-EXEC
```

In Data Division

```
EXEC SQL  
    OPEN EMPCUR  
END-EXEC  
EXEC SQL  
    FETCH EMPCUR INTO :EEMPNO, :ENAME  
END-EXEC  
PERFORM UNTIL SQLCODE = 100  
    DISPLAY EEMPNO, ENAME-TEXT( 1 : ENAME-LEN)  
    EXEC SQL  
        FETCH EMPCUR INTO :EEMPNO, :ENAME  
    END-EXEC  
END-PERFORM  
EXEC SQL  
    CLOSE EMPCUR  
END-EXEC
```

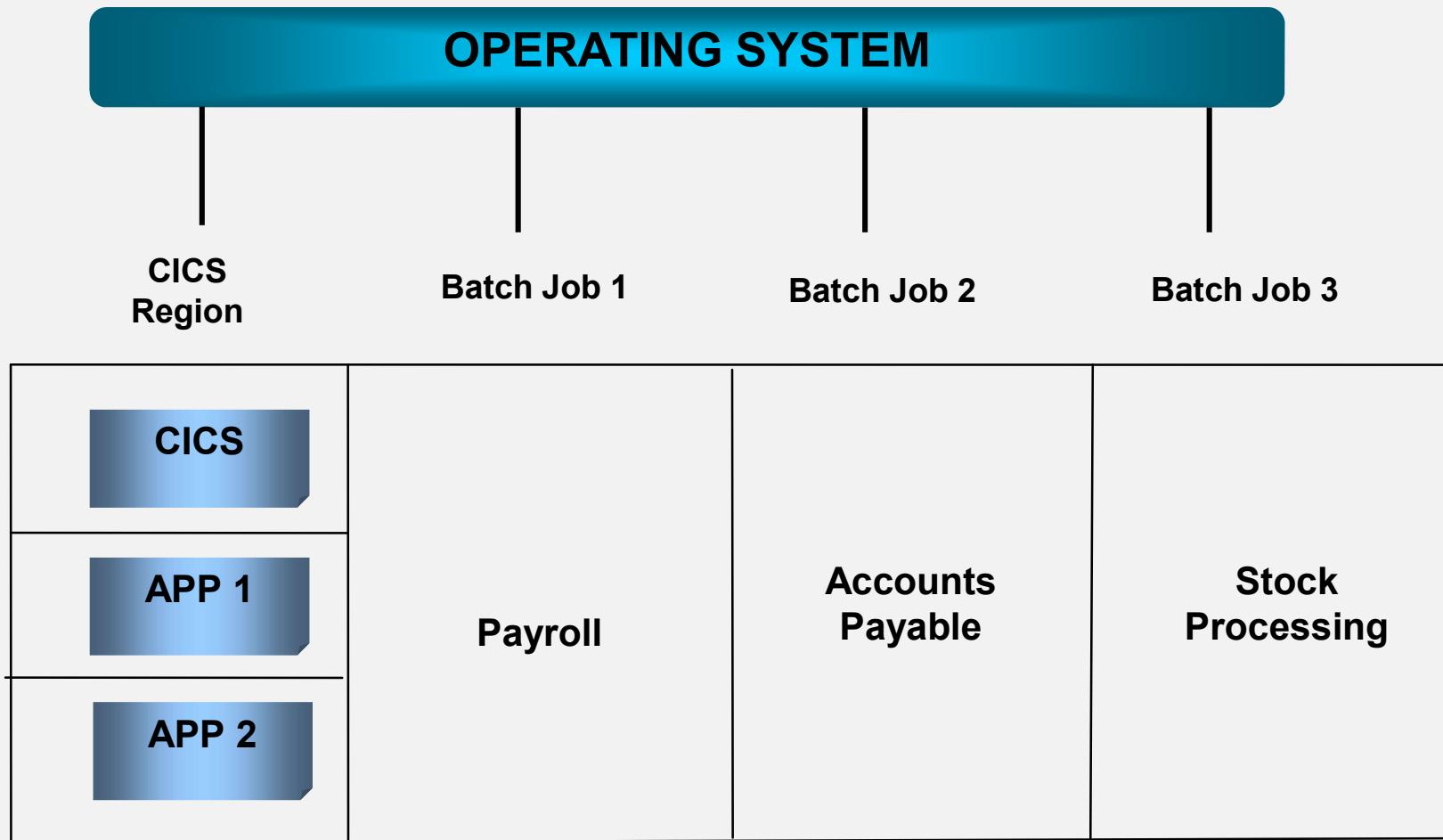
In Procedure Division

Customer Information Control System (CICS)

Introduction to CICS

- Stands for *Customer Information Control System*
- **Availability** - Mainframe, AS/400, RISC/6000 & OS/2
-
- CICS handles file management itself and so there is no need for statements like READ ,WRITE,OPEN
- Allows hundreds of concurrent users, many of whom sharing the files
- Acts as an interface between application programs and OS services
- CICS is an OS under the OS
- As far as OS is concerned CICS is an application program. It has its own JCL

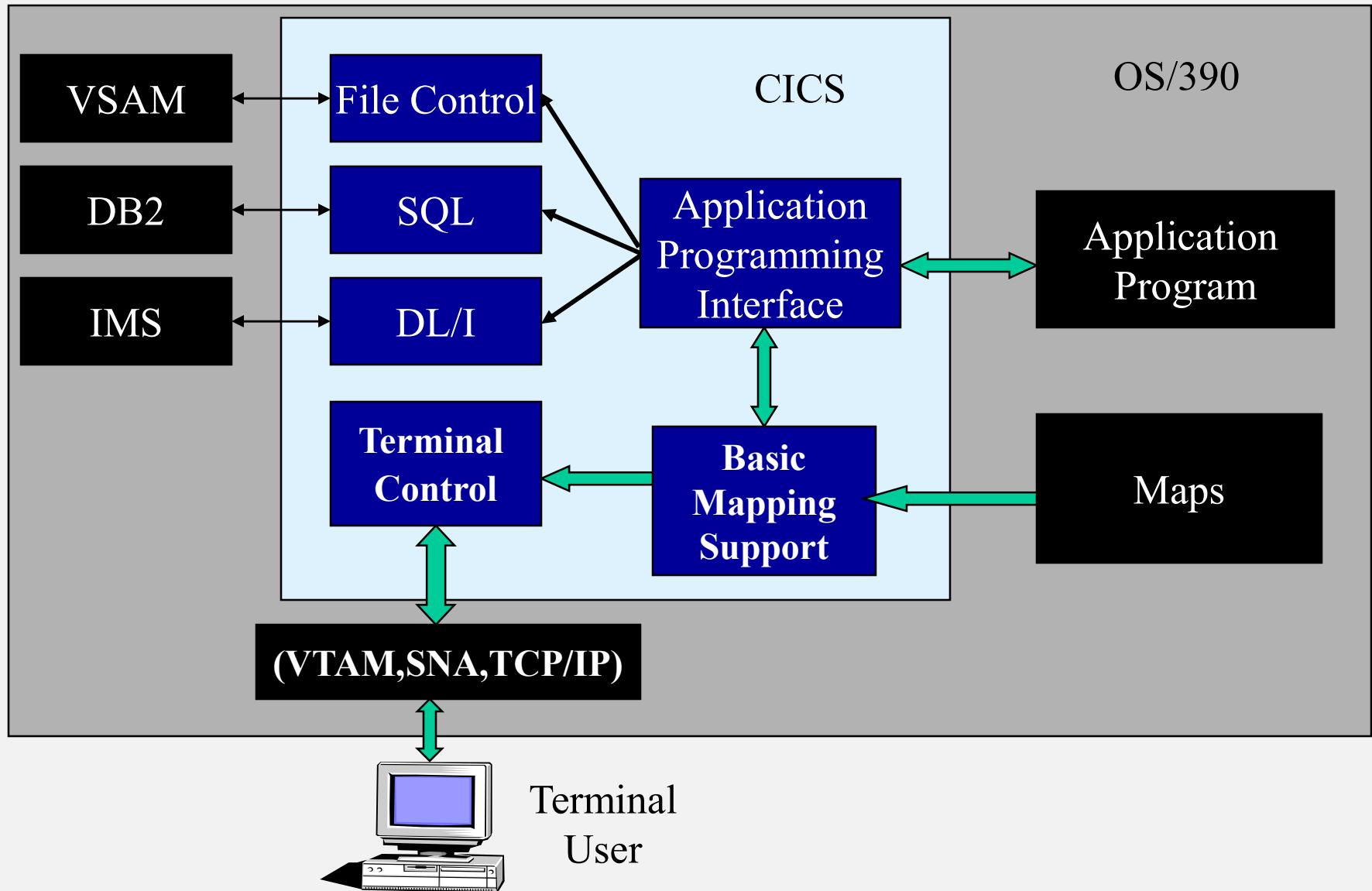
CICS under O S



CICS Services

- CICS provides following services/functions:
 - **Data Communication Functions**
 - VTAM,TCAM,BDAM
 - BMS
 - Multi region communication
 - **Data Handling**
 - Interface with VSAM
 - Interface with DB2, SQL/DS, DLI
 - Data Integrity Application program services
 - Interface with COBOL, PLI, HLASM
 - Command translator
 - **System Services**
 - Program loading, Storage Control, Task Control

CICS Services Diagram



Batch vs Online System

Comparison between Online and Batch Systems

	Batch System	On-line System
Input	Sequential, submitted as group, edited and verified	Random, Edited immediately
Master file updation	Periodically (weekly, monthly etc) with the batched transactions	Updated instantly as the transaction occurs
Processing	Sequential. File used by one program at a time	Random, Many Users can access a file at one time
Output	Usually printed reports	Displayed on terminal
Response Time	could be scheduled to be in hours or days	seconds or minutes

Multi tasking

- Multitasking means that the operating system allows more than one task to be executed concurrently, regardless of whether the tasks use the same program or different programs.
- CICS provides multitasking environment where more than one CICS task run concurrently.
- CICS provides multitasking from within the single address space provided by the OS overriding its multitasking function.
- CICS manages multitasking of CICS tasks within its own region i.e. more than one CICS task run concurrently

Multi threading

- Multithreading is the system environment where the tasks share the same program under the multitasking environment.
- Multithreading is a subset of multitasking since it consists of tasks that use the same program.
- Multiple users running the same program have access to the same storage location
- The area of storage containing a program is not allocated to a specific user
- CICS manages its tasks within its own region
- For the multi threading to work, the program must be **re-entrant**

Re-entrant Programs

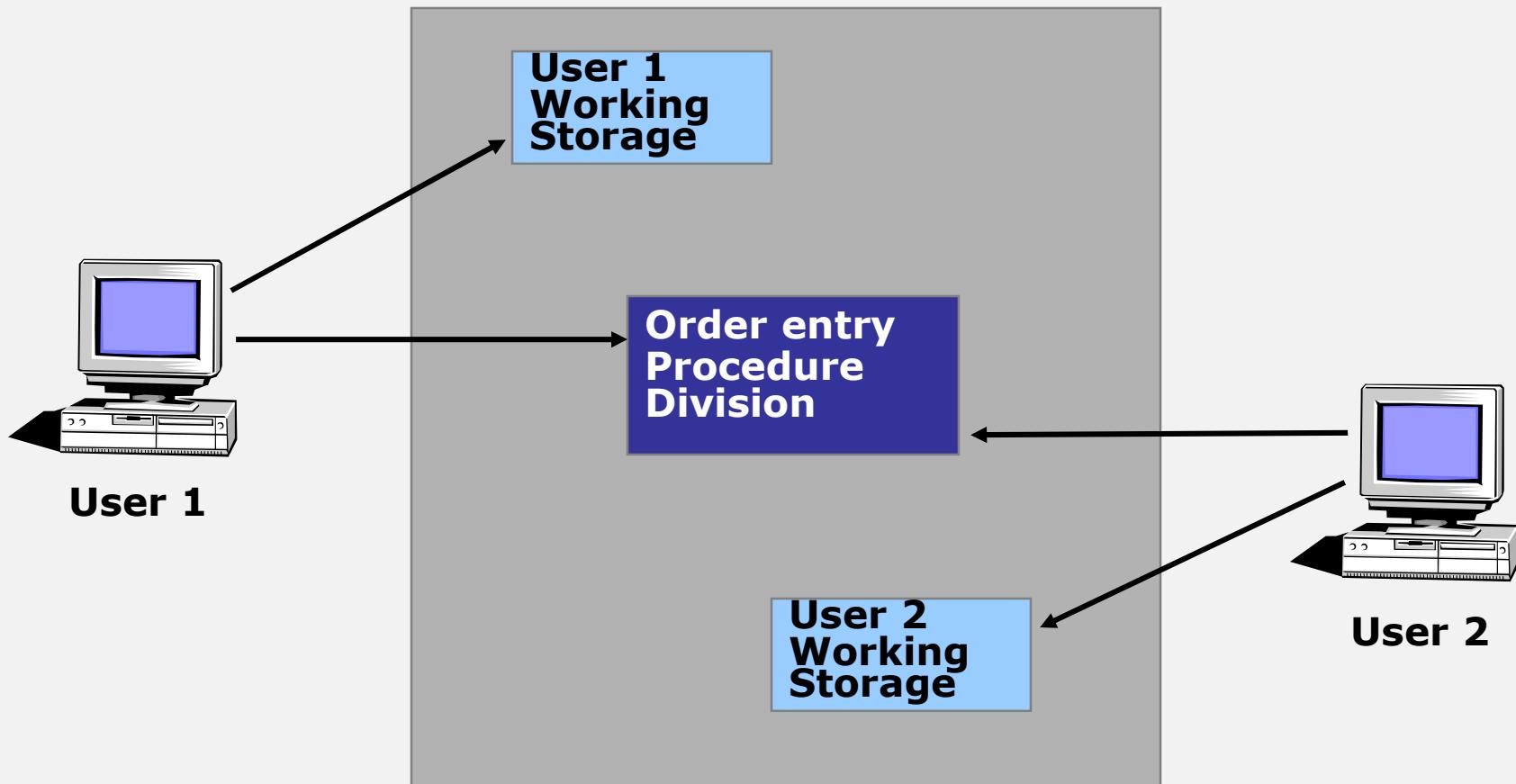
Re-entrant Program

- A program that does not modify itself in any way during execution
- Re-entrant program cannot modify data in working storage
- Is able to reenter itself and continue processing, after an interruption by the operating system

Quasi Re-entrant Program

- A quasi re-entrant program is a re-entrant program under the CICS environment.
- CICS makes this possible by providing separate copy of working storage for each user
- Users share same copy of program but separate working storage area
- All CICS command level COBOL application programs are quasi re-entrant

Re-entrant Programs



CICS Internals

- The core portion (Nucleus) of CICS consists of IBM supplied control programs and corresponding user defined CICS control tables
- CICS control programs perform their primary task based on the entries in the corresponding user defined CICS control tables
- Programmers can modify the control table entries depending on the requirement
- This makes CICS system flexible and easy to maintain

Control Programs and tables

Control Programs	Control Tables
SIP (System Initialization Program)	SIT (System Initialization Table)
TCP (Terminal Control Program)	TCT (Terminal Control table)
SCP (Storage Control Program)	
KCP (Task Control Program)	PCT (Program Control Table)
PCP (Program Control Program)	PPT (Processing Program Table)
FCP (File Control Program)	FCT (File Control Table)
TDP (Transient Data Program)	DCT (Destination Control Table)
Many Others	Many Others

Control Programs and tables

PCT - Program Control Table

- A list of all the TRANSIDs available to CICS is found in the PCT
- The PCT is created and maintained by the systems programmers.
- When you write a new transaction, you need to arrange for the systems programmers to add the new TRANSID to the PCT.
- When you type a TRANS ID, it is looked up in the PCT
- An 8 byte program ID is read, naming the first program in the transaction to be executed which is then looked up in the Processing Program Table (PPT)
- For each TRANS ID, the PCT
 - Identifies the first program to be executed for the transaction
 - Identifies the priority for the transaction
 - Contains the security level for the transaction
- TRANS-ID is of four characters
- IBM TRANS-IDs start with 'C'

Trans ID	Prog. Name
TRN1	PROGRAM1
TRN2	PROGRAM2
TRN3	PROGRAM3

Control Programs and tables

PPT - Processing Program Table

- Registers all CICS application programs and BMS maps.
- Used by CICS to determine if in storage.
- IF not in storage, CICS loads NEW COPY from DASD library
- PPT created by programmer with CEDA

TCT - Terminal Control Table

- Defines the terminal environment.
- Identifies the lines, terminals and priorities

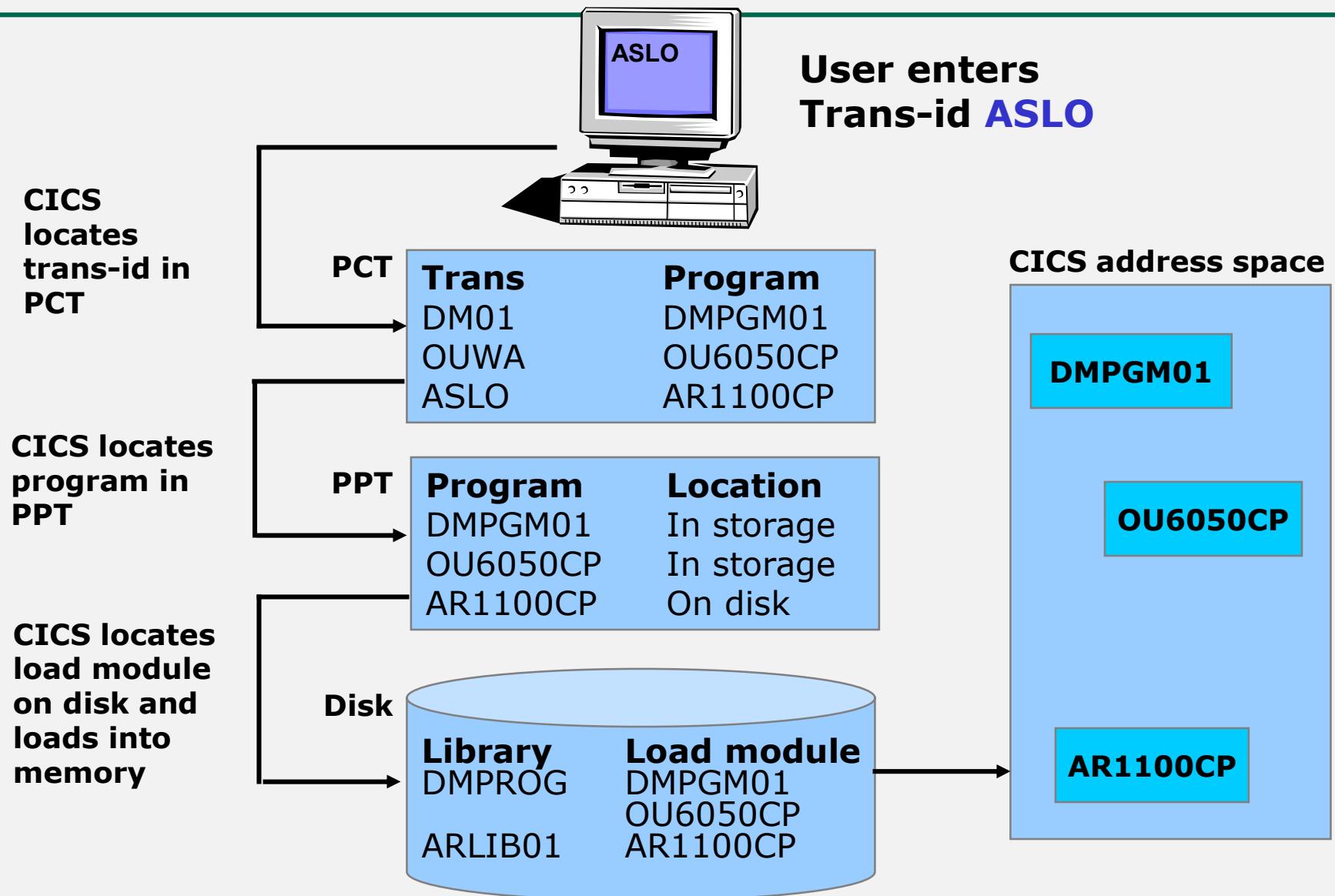
FCT - File Control Table

- Lists and describes user files.
- Contains File characteristics, organization, record formats.

Transaction

- A unit of work done as atomic operation
- Identified by a 4 character ID
- Steps in Transaction processing
 1. You type a TRANSID.
 2. CICS looks up the TRANSID in the PCT to find the name of the initial program.
 3. The initial program is looked up in the PPT (disk and memory information are looked up)
 4. Program is loaded from disk into main memory.
 5. Main memory is assigned space for this program.
 6. Task work space is assigned for CICS's own use.
 7. The task is executed.

Transaction



Pseudo conversational Programming

Conversational Program

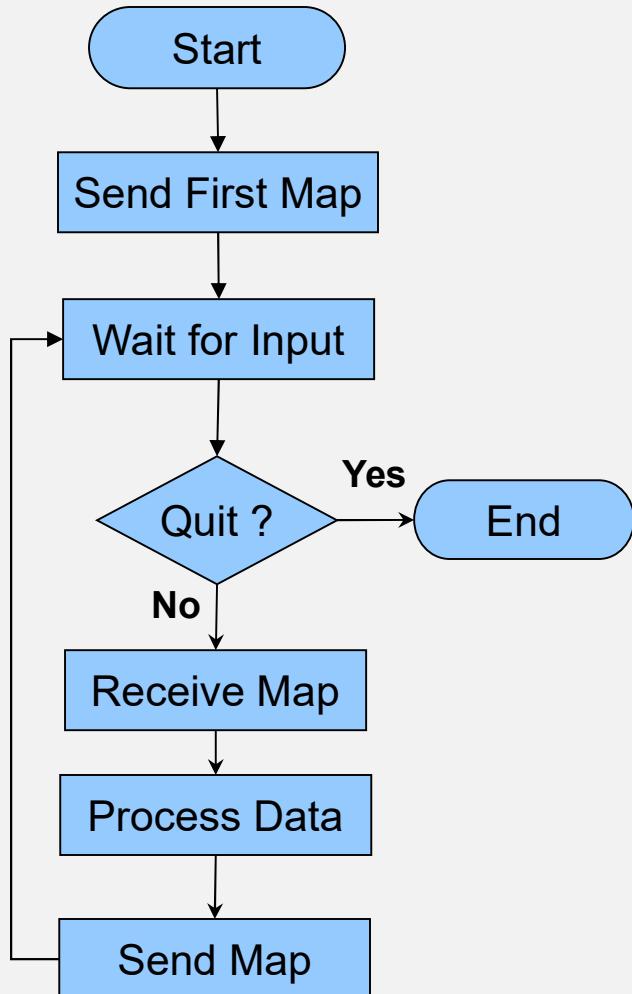
- A mode of dialogue between the program and the terminal
- Program remains idle when waiting for user response
- Program, data areas, control blocks remain in main storage resulting in high virtual storage utilization

Pseudo Conversational Program

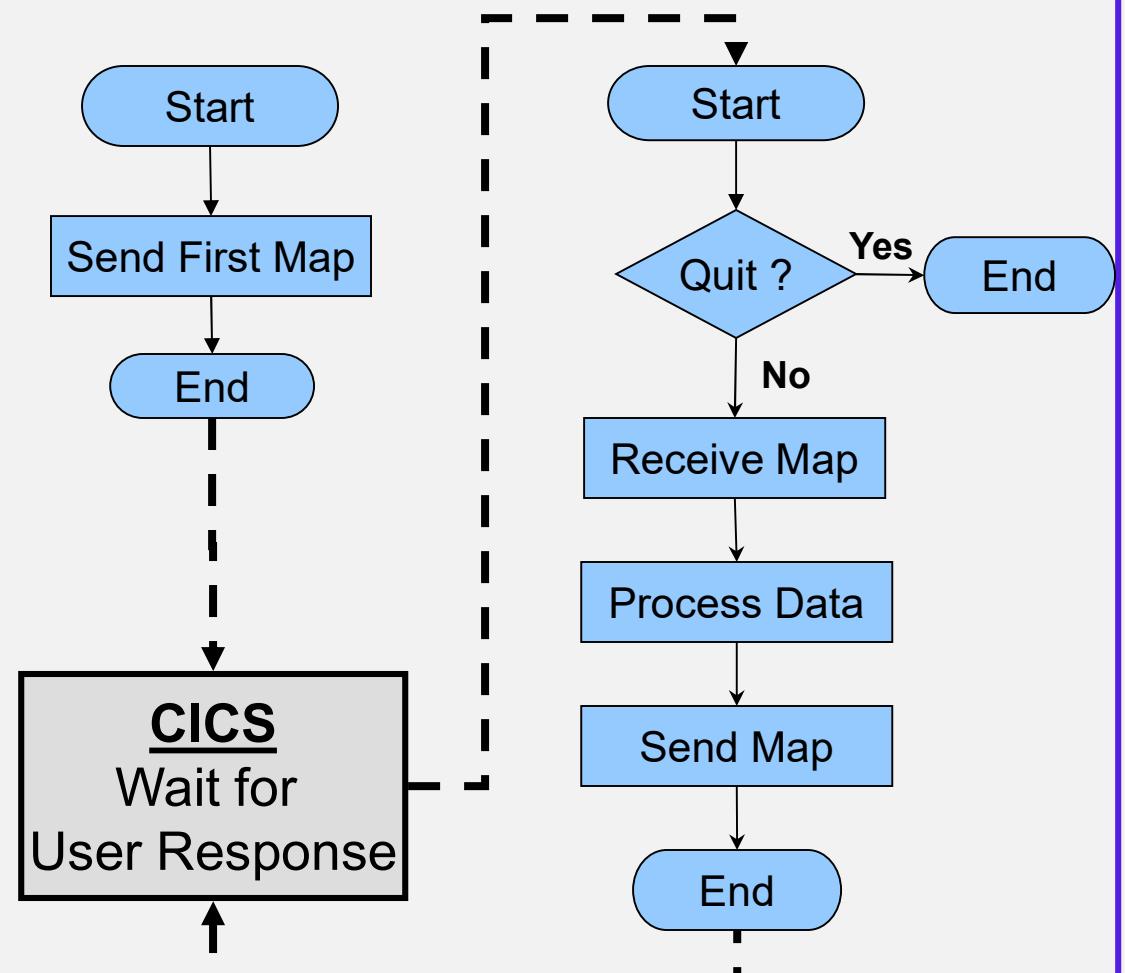
- Program terminates task after sending message to terminal
- When user completes the response, next task is automatically initiated by CICS
- System point of view - Multi task operation
- User point of view - Normal conversation

Pseudo conversational Programming

Conversational Program



Pseudo-conversational Program



Pseudo conversational Programming

- TCP with TCT recognizes incoming data
- SCP gets storage for TIOA (Terminal Input Output Area)
- TCP places data in TIOA & sets pointer to TCT
- TCP passes control to KCP
- KCP realizes the transaction identifier in TIOA
- SCP acquires storage for TCA and puts control data in this
- KCP looks up PCT / PPT to get the program
- If PPT does not have resident address, KCP passes control to PCP which fetches program from loadlib
- KCP passes control to program

CICS Programming - COBOL

- **ENVIRONMENT DIVISION**
 - Only header is allowed, no other sections needed
 - Can be omitted altogether
- **DATA DIVISION**
 - FILE SECTION not allowed
 - WORKING-STORAGE SECTION is required
 - LINKAGE SECTION is optional
- **PROCEDURE DIVISION**
 - Code is mixture of standard COBOL statements and special CICS commands
 - The following statements are not allowed
 - ✓ ACCEPT, DISPLAY, OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, START, EXHIBIT, TRACE
 - ✓ REPORT WRITER feature
 - ✓ SORT feature
 - CICS commands are used in EXEC CICS - END-EXEC block

CICS Commands in the program

General Structure :

```
EXEC CICS  
    Command [option (value) ] ....  
END-EXEC
```

command : a CICS service request

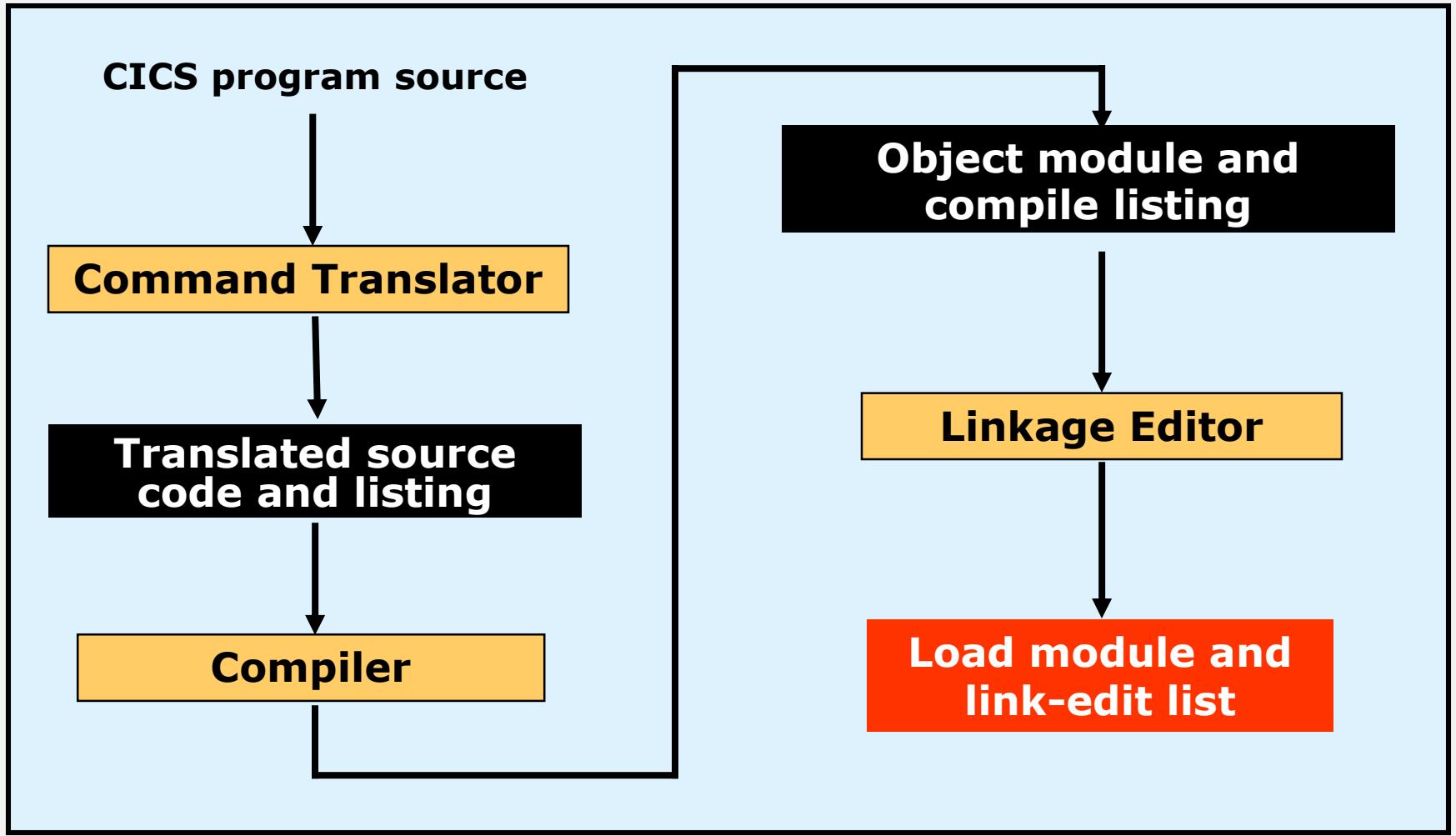
option : one of the options available to command

value : determines the characteristics of the value to be placed for the option as detailed information

Example :

```
EXEC CICS  
    READ DATASET (empfil)  
        INTO (emprec)  
        RIDFLD (empno)  
END-EXEC
```

Program Compilation



Program Development

- Plan the transaction and design the program
- Prepare the BMS mapset
- Code the program
- Compile the BMS mapset
- Translate, compile and link edit the program to create Load module
- Define Program in PPT
- Define transaction in PCT with program name
- Define the mapset in PPT
- Define the files, if required, in FCT
- Sign on to CICS
- Enter the transaction identifier

Basic Mapping Support

Basic Mapping Support (BMS)

- For the formatted screen some device dependent terminal control information has to be added to the data.
- If the application program has to deal with formatted screen, it has to handle both data and terminal control information.
- If the format/device has to be changed then the application program needs to be changed.
- BMS avoids this extra burden of an application program by making the formatted screen definition independent of the program and by taking care of handling terminal control information
- BMS is used to develop formatted screens, which handle the terminal control information while sending or receiving formatted screens.

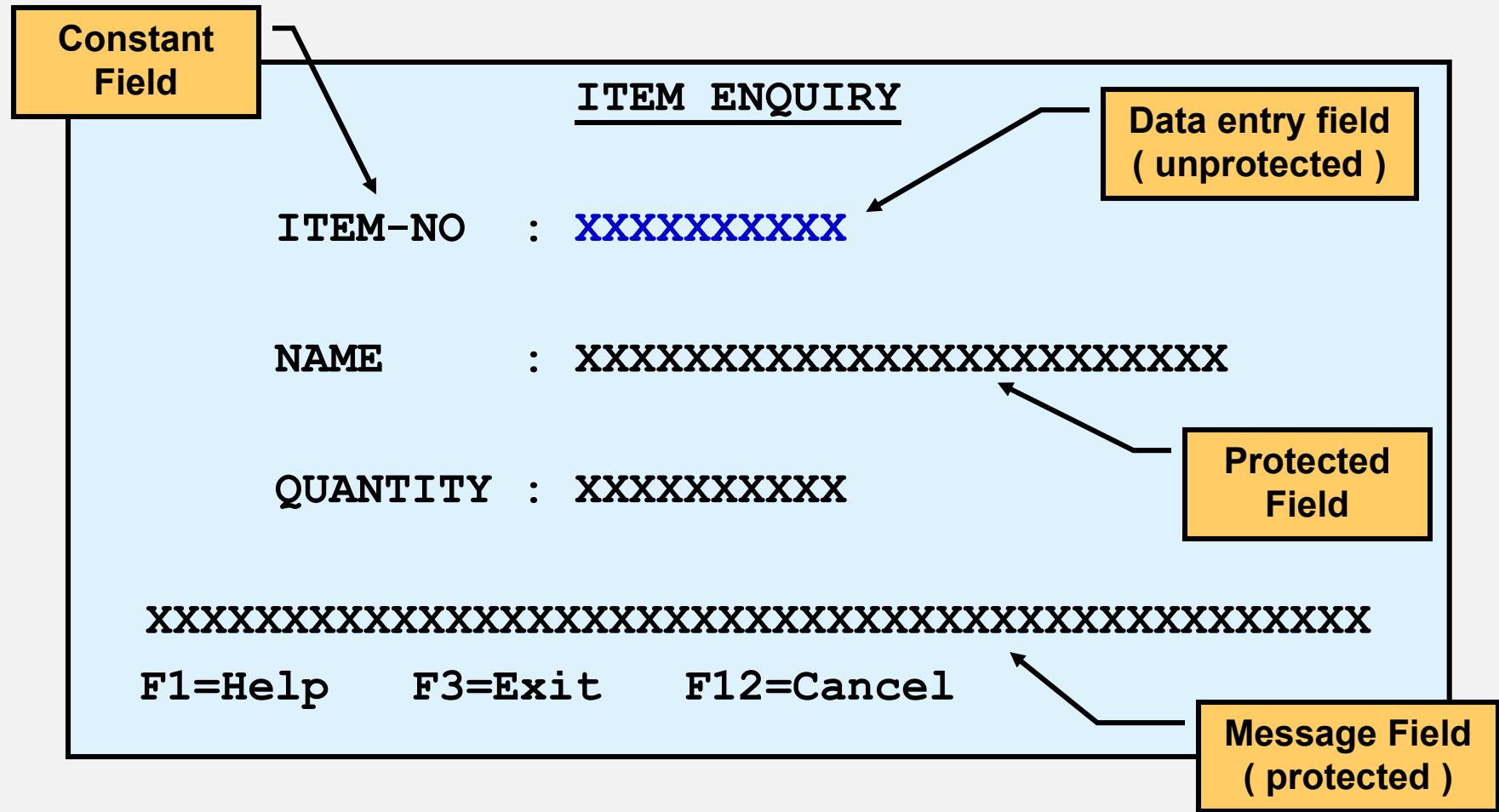
Primary functions of BMS

- To remove device dependent code from an application program
- To remove constant information (headers, field labels etc) from an application program
- To construct the data & control information character stream to produce formatted output screen.
- To interpret the control information and provide access to the data fields of input stream received from the terminal
- To allow repositioning of fields of the screen without modifying the application program
- To provide Terminal Paging Facility (Combination of small mapped data areas into pages of output)
- To provide Message Routing Facility (Sending of messages to terminals).

Map and MapSet

- A formatted screen defined through BMS is called Map
- Map contains constant and input fields along with their properties like position, size, attributes etc
- One or more maps, link edited together, makes a Mapset (load module) which is a member of a PDS
- Mapset must have a entry in PPT
 - BMS provides a set of assembler macros to define Maps and Mapsets
 - These macros are to be assembled and link-edited
 - **BMS Macros**
 - DFHMSD to define a Mapset
 - DFHMDI to define a Map
 - DFHMDF to define the Fields in a map

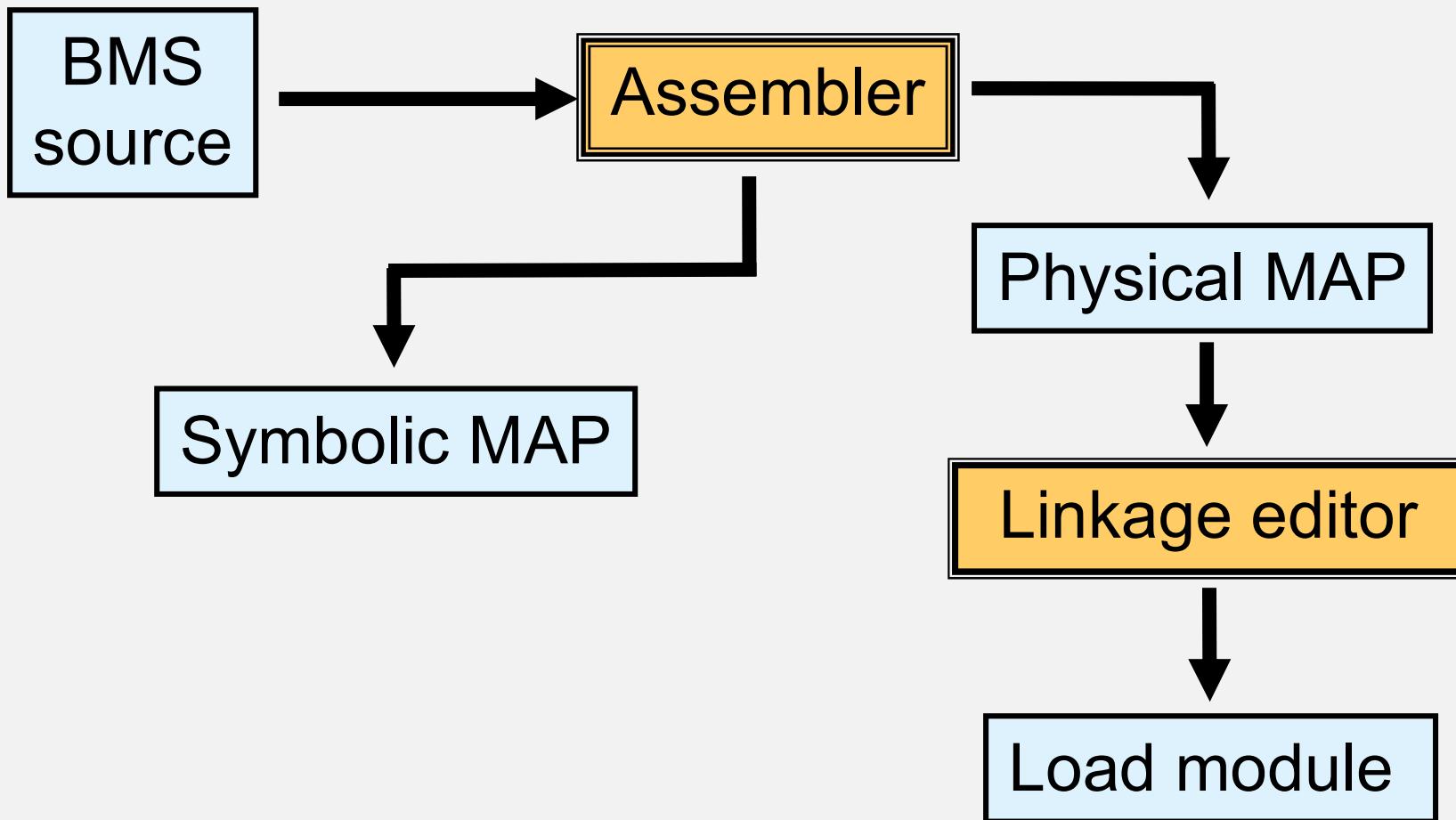
BMS – Sample screen



Types of Maps

- **Physical Map**
 - Physical map contains the information about field length, position and attribute and is used by the BMS to construct the data stream to be sent to the terminal and to interpret the data stream from the terminal
 - It is in the form of a load module stored in Load Library
- **Symbolic Map**
 - Symbolic map is in the form of symbolic variables and is used by the application program
 - Helps to keep screen format independent of the program
 - Application program uses the symbolic map fields to receive the data entered by the user or to send data to the screen
 - Symbolic map is generated as a COPYLIB member when the BMS macros are assembled
 - The symbolic map is to be copied into the working-storage section of the COBOL program

Types of Maps



Sample Symbolic Map

Single Field map

01 EMPRECI.

02 FILLER PIC X(12).

02 EMPNAMEL PIC S9(4) COMP.

02 EMPNAMEF PIC X.

02 FILLER REDEFINES EMPNAMEF.

03 EMPNAMEA PIC X.

02 EMPNAMEI PIC X(20).

01 EMPRECO REDEFINES EMPRECI.

02 FILLER PIC X(12).

02 FILLER PIC X(03).

02 EMPNAMEO PIC X(20).

Variables used
for INPUT

Variables used
for OUTPUT

BMS Macro format

Col.	Col.	Col.	Col.
1	9	16	72
#-----#-----#-----			#
LABEL	Op-Code	Parameters separated by commas(parm1=..)	continuation char Col.72

Example :

**EMPSET1 DFHMSD TYPE=MAP,MODE=INOUT, X
LANG=COBOL,TIOAPFX=YES, X
CTRL=(FREEKB,ALARM)**

BMS Macro format (contd)

- **Label**
 - supplies a symbolic name for the statement
 - Can be up to 7 characters long
 - Required for DFHMSD, DFHMDI.
 - Not required for constant fields in DFHMDF
 - CICS adds one character suffix to some of the labels
- **Op-code**
 - Specifies one of the BMS macros
- **Parameters**
 - Provide information the instruction requires
 - They are separated by commas with no intervening spaces
 - Specify parameter value using equal (=) sign
 - If a parameter has more than one value, separate the values with comma and enclose in parentheses

BMS Macro format (contd)

- **Continuation lines**
 - To continue parameter list in next line, place any non-blank character in column 72 and continue the next parameter in column 16 of next line
- **Comment lines**
 - Any line starting with asterisk (*) in column 1 is comment line

BMS Macros - Template

DFHMSD	Mapset Definition – Start	
DFHMDI	Map Definition - Map 1	Map 1
DFHMDF	Field Definition in a Map - Field 1	
DFHMDF	Field Definition in a Map - Field 2	
.....		
.....		
DFHMDI	Map Definition - Map 2	Map 2
DFHMDF	Field Definition in a Map - Field 1	
DFHMDF	Field Definition in a Map - Field 2	
.....		
.....		
DFHMSD	Mapset Definition - End	

Map Definition options

- **Mapset consisting of single map**
 - Simple to design
 - Assemble and link edit only one map/mapset when changed
 - Increases number of entries in PPT
- **Mapset consisting of several maps**
 - Used in multimap panel
 - Need to assemble and link edit the entire mapset when one of the maps is changed
 - Reduces number of PPT entries

DFHMSD macro – MapSet definition

```
setname DFHMSD      TYPE={ &SYSPARM | DSECT | MAP },
                      MODE={ IN | OUT | INOUT },
                      LANG={ASM | COBOL | C | PLI },
                      TERM=terminal type
                      STORAGE=AUTO,
                      TIOAPFX={ YES | NO },
                      CTRL=(ctrl1,ctrl2...)
```

Underlined items
are default values

TYPE

- Specifies whether a physical map (TYPE=MAP), symbolic map (TYPE=DSECT) or both (TYPE=&SYSPARM) will be generated
- TYPE=FINAL marks the end of mapset

MODE

- Specifies whether the mapset is used for input (IN), output (OUT) or both (INOUT)

LANG

- Specifies the programming language: assembler (ASM), COBOL or PL/I

DFHMSD macro

TERM

- Specifies the terminal supported by the physical map
- 3270, 3270-1, 3270-2 etc

STORAGE

- AUTO – symbolic maps will occupy separate storage locations
- Otherwise they occupy same storage locations

TIOAPFX

- YES – generates a 12 byte FILLER in symbolic map and should be always specified for COBOL
- NO – Does not generate this area

CTRL

- Specifies list of control options. Some common options are
 - FREEKB – unlock keyboard on map display
 - ALARM - sound the audio alarm on map display
 - FRSET - Reset MDT bit (discussed later)

DFHMSD macro - example

```
EMPSET1 DFHMSD TYPE=&SYSPARM, X
          CTRL=(FREEKB,FRSET), X
          TIOAPFX=YES, X
          LANG=COBOL, X
          STORAGE=AUTO, X
          MODE=INOUT, X
          TERM=3270
```

**Map Definition macros
(DFHMDI)
here**

```
DFHMSD TYPE=FINAL
END
```

DFHMDI macro – Map definition

mapname	DFHMDI	SIZE=(lines,columns), LINE=line number, COLUMN=column number, CTRL=(ctrl1,ctrl2...)
---------	--------	--

SIZE

- Specifies size of the map

LINE

- Specifies the starting line number. Usually LINE=1

COLUMN

- Specifies the starting column number. Usually COLUMN=1

CTRL

- Same as CTRL in DFHMSD macro
- Cannot be specified in both DFHMSD and DFHMDI

DFHMDI macro - example

EMPMAP1 DFHMDI	SIZE=(24,80),	X
	LINE=1,	X
	COLUMN=1,	X
	CTRL=(FREEKB,ALARM,FRSET)	

Extended attributes

- Attributes used to allow Color and hilight options for the fields in DFHMDF macro
- Can be specified either in DFHMSD or DFHMDI
- **MAPATTS = (COLOR,HIGHLIGHT)**
 - Specifies whether extended attributes like color, hilight supported for physical map
- **DSATTS = (COLOR,HIGHLIGHT)**
 - Specifies whether extended attributes like color, hilight supported for symbolic map
 - Results in additional variables for each attribute in symbolic map

DFHMDF macro – Field definition

name DFHMDF	POS=(line,column), LENGTH=field length, ATTRB=({ BRT <u>NORM</u> } , { PROT UNPROT } , DRK <u>SKIP</u> },NUM,IC,FSET), INITIAL='literal', PICIN='picture string', PICOUT='picture string', COLOR=color, HIGHLIGHT=hilight options
-------------	--

POS

- Specifies the position of the field

LENGTH

- Specifies the length of the field

DFHMDF macro

ATTRB

- Specifies one or more attributes for the field

BRT field displayed with high intensity

NORM field displayed with normal intensity

DRK field not displayed (black letter on black background)

PROT field is protected. Data entry not allowed

UNPROT field is unprotected. Data may be keyed into it

ASKIP protected skip field

cursor will automatically skip over it to next unprotected field

NUM numeric field. Data is right justified and zero filled on left.

only numeric values allowed in data entry

IC initial cursor position

cursor positioned in this field on map display

FSET Set MDT bit (discussed later)

DFHMDF macro – Field definition

INITIAL

- Specifies initial value. If omitted, defaults to hexadecimal zeroes (LOW-VALUE)

PICIN

- Specifies COBOL picture string to define input data format
- Length of the picture string should match the length of the field

PICOUT

- Specifies COBOL picture string to define output data format
- Length of the picture string should match the length of the field

COLOR

- Specifies field color (BLUE,RED,PINK,GREEN,TURQUOISE,YELLOW,NEUTRAL)
- Allowed only if DSATTS or MSATTS used in map or mapset

HIGHLIGHT

- Specifies hilight options (BLINK,REVERSE,UNDERLINE)
- Allowed only if DSATTS or MSATTS used in map or mapset

DFHMDF macro - example

	DFHMDF POS=(1,33), LENGTH=13, INITIAL='Employee Data', ATTRB=(PROT,NORM)	X
	DFHMDF POS=(3,5), LENGTH=8, ATTRB=(PROT,NORM), INITIAL='Emp No :'	X
EMPNO	DFHMDF POS=(3,15), LENGTH=5, ATTRB=(UNPROT,NORM)	X
	DFHMDF POS=(3,21), ATTRB=ASKIP, X LENGTH=1	
	DFHMDF POS=(4,5), LENGTH=8, ATTRB=(PROT,NORM), INITIAL='Salary :'	X
SALARY	DFHMDF POS=(4,15), LENGTH=9, ATTRB=(PROT,NORM), PICOUT='ZZ,ZZZ.99'	X

Attribute Byte

- Invisible one byte character prefixed to the field
- Defines the characteristics of a field specified in ATTRB parameter in DFHMDF macro
- Attribute byte takes up one position on the screen to the left of the field
- Field starts immediately after the attribute byte and ends at the attribute byte of the next field
- If there is no next field, the field continues till the end of the screen
- Attribute byte should be considered to decide on the position of the next field
- For example, if field is defined with POS=(2,6) and LENGTH=6, position for the next field should be defined as POS=(2,13)

Modified Data Tag (MDT)

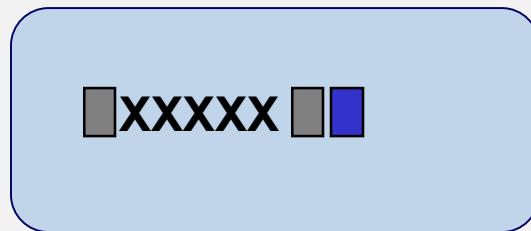
- MDT is one bit in the attribute byte which indicates whether the field is modified by the terminal user
- MDT = 0 indicates that the field has not been modified
- MDT = 1 indicates that the field has been modified
- MDT will be set to '1' automatically when the terminal user modifies the field
- To save transmission time, the 3270 sends a field over TC line only if its MDT is on. Otherwise the field is not transmitted
- When CTRL=FRSET is specified on DFHMSD or DFHMDI, MDT is set '0' for all fields while sending the map
- When FSET is specified on the ATTRB parameter of the field definition, the MDT is set to '1' for that field regardless of whether the field is modified or not

Skip Field

- It is a good practice to skip the cursor to the next unprotected field after user fills an unprotected field
- Skip field is used for this purpose
- Skip field is an unlabeled one byte field with the autoskip (ASKIP) attribute
- Example :

```
EMPNAME DFHMDF POS=(10,10),LENGTH =5,ATTRB=(IC,UNPROT)
DFHMDF POS=(10,16),LENGTH =1,ATTRB=ASKIP
```

- Screen layout :



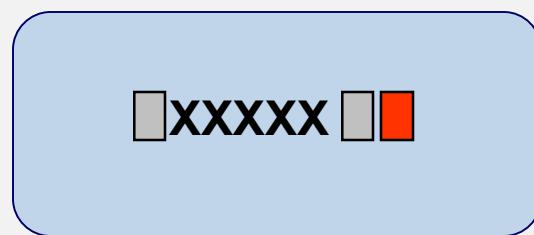
- Attribute Bytes
- Skip field

Stop Field

- Stop field is used to stop the cursor to prevent erroneous field overflow by the terminal operator
- Stop field is an unlabeled one byte protected field
- If Stop field is used to mark the end of unprotected field, user has to press TAB key to go to next data entry field
- Example :

```
EMPNAME DFHMDF POS=(10,10),LENGTH =5,ATTRB=(IC,UNPROT)
DFHMDF POS=(10,16),LENGTH =1,ATTRB=PROT
```

- Screen layout :

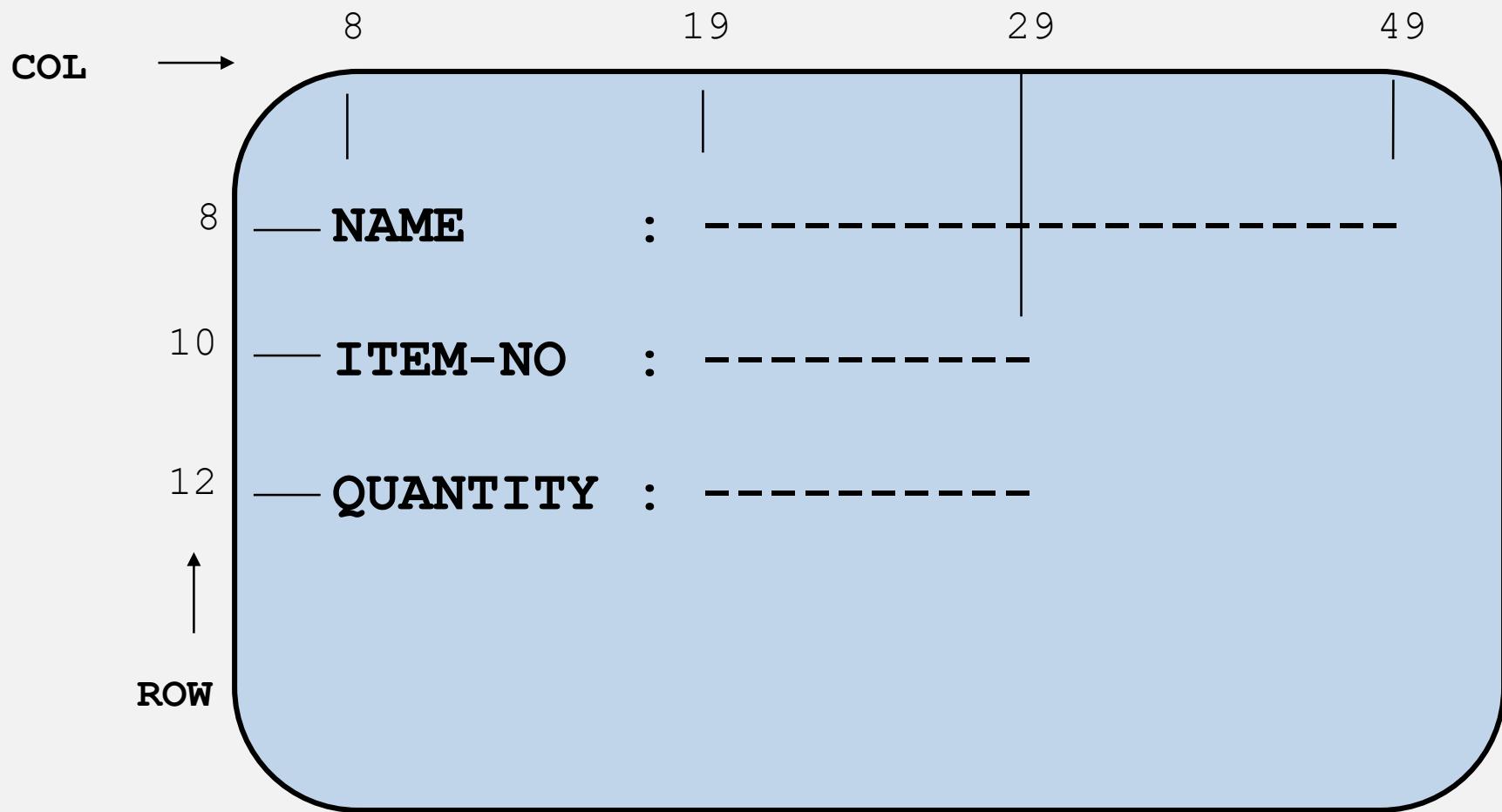


■ Attribute Bytes
■ Stop field

Symbolic Map format

- The symbolic map starts with the 01 level definition of mapname with the suffix 'I' for input map and 'O' for output map.
- Next is the definition of the filler PIC X(12) which is a TIOA prefix.
- For each field name specified on the BMS macro, BMS creates the following fields
 - **name+L**
 - Half word binary (S9(4) comp) field
 - For input operation, it contains actual number of characters entered
 - Used for cursor positioning during output operation
 - **name+F**
 - Flag byte, indicates whether the data field was cleared
 - Contains value X'80' if cleared, otherwise X'00'
 - **name+A**
 - Field attribute byte
 - **name+I / name+O**
 - Input / Output data fields

Sample Map Layout



BMS Code for Sample Map Layout

```
MAPSETS  DFHMSD  TYPE=&SYSPARM,MODE=INOUT,TERM=ALL,          X
          LANG=COBOL,TIOAPFX=YES,STORAGE=AUTO

MAPSMP   DFHMDI   SIZE=(24,80),LINE=1,COLUMN=1
          DFHMDF   POS=(8,7),LENGTH=10,INITIAL='NAME      : ',    X
                      ATTRIB=PROT

NAME     DFHMDF   POS=(8,18),LENGTH=20,ATTRB=(UNPROT,IC)
          DFHMDF   POS=(8,39),LENGTH=1,ATTRB=ASKIP
          DFHMDF   POS=(10,7),LENGTH=10,INITIAL='ITEM-NO  : ',    X
                      ATTRIB=(PROT,NORMAL)

ITEM     DFHMDF   POS=(10,18),LENGTH=6,ATTRB=(NUM,UNPROT)
          DFHMDF   POS=(10,25),LENGTH=1,ATTRB=ASKIP
          DFHMDF   POS=(12,7),LENGTH=10,INITIAL='QUANTITY : ',    X
                      ATTRIB=PROT

QTY      DFHMDF   POS=(12,18),LENGTH=6,ATTRB=(UNPROT,NUM)
          DFHMDF   POS=(12,25),LENGTH=1,ATTRB=ASKIP
          DFHMSD   TYPE=FINAL
          END
```

Symbolic Map for Sample Map Layout

01 MAPSMPI

02 FILLER PIC X(12).
 02 NAMEL PIC S9(4) COMP.
 02 NAMEF PIC X
 02 FILLER REDEFINES NAMEF.
 03 NAMEA PIC X.
 02 NAMEI PIC X(20).
 02 ITEMEL S9(4) COMP.
 02 ITEMF PIC X
 02 FILLER REDEFINES ITEMF.
 03 ITEMA PIC X.
 02 ITEMI PIC X(06).
 02 QTYL S9(4) COMP.
 02 QTYF PIC X
 02 FILLER REDEFINES QTYF.
 03 QTYA PIC X.
 02 QTYI PIC X(06).

01 MAPSMPO REDEFINES MAPSMPI.

02 FILLER PIC X(12).
 02 FILLER PIC X(3).
 02 NAMEO PIC X(20).
 02 FILLER PIC X(3).
 02 ITEM0 PIC X(06).
 02 FILLER PIC X(3).
 02 QTYO PIC X(06).

Error Handling

Error Handling

- When CICS command encounters unusual situation, it generates an exceptional condition
- For example, when READ command cannot find a record, the NOTFND condition is raised
- If these conditions are not handled by the program, the task is abnormally terminated
- Then CICS displays an abend code that identifies the exceptional condition
- Exceptions can be handled by
 - HANDLE CONDITION command
 - NOHANDLE option in any CICS command
 - IGNORE CONDITION command
 - Using RESP option in any CICS command
 - PUSH / POP

HANDLE CONDITION Command

- HANDLE CONDITION specifies a procedure name (Paragraph or Section) that is given control for the exception
 - ERROR matches any condition
 - Format :

```
EXEC CICS  
      HANDLE CONDITION conditionname ( procedure name )  
              [ ERROR (procedure name) ]  
END-EXEC
```

- Example :

NOHANDLE Option

- NOHANDLE option can be used in any CICS command to ignore any HANDLE CONDITION in effect
- It skips to default action in case of any error while executing the command
- Example :

```
EXEC CICS
    READ      DATASET (name) | FILE (name)
              INTO (data-area)
              RIDFLD (Key)
              [ LENGTH (record size) ]
              NOHANDLE
END-EXEC
```

IGNORE CONDITION Command

- IGNORE CONDITION specifies that no action is to be taken when named conditions occur.
- No more than 16 conditions are allowed in the list
- If one of the named conditions occurs, control returns to the instruction that follows the command that has failed to complete
- Format :

```
EXEC CICS  
    IGNORE CONDITION conditionname .....  
END-EXEC
```

- Example :

```
EXEC CICS  
    IGNORE CONDITION      NOTFND  
                      MAPFAIL  
                      LENGERR  
END-EXEC
```

RESP Option

- RESP option can be used in any CICS command to capture the response code
- When RESP option is specified CICS ignores any exceptional condition and NOHANDLE is automatically applied
- RESP option eliminates the need for HANDLE CONDITION
- Define a half word binary variable (PIC S9(4) COMP) to capture response code
- Name the variable in the RESP option
- Check the variable value for possible exception conditions using DFHRESP keyword
- CICS translates DFHRESP specification to a numeric value that corresponds to the exception

RESP Option (contd)

- Example :

```
EXEC CICS
```

```
    READ      DATASET (name) | FILE (name)  
            INTO (data-area)  
            RIDFLD (Key)  
            RESP ( RCODE )
```

```
END-EXEC
```

```
IF RCODE = DFHRESP(NOTFND)  
    PERFORM NO-RECORD-PARA  
END-IF
```

Working-Storage Section.
01 RCODE PIC S9(4) COMP.

PUSH / POP HANDLE

- When CICS program links to a subroutine, the program or routine that receives control inherits the current HANDLE commands
- These commands might not be appropriate within the called program
- The called program can use PUSH HANDLE to suspend existing HANDLE commands
- Before returning control to its caller, the called program can restore the original HANDLE commands by using POP HANDLE
- Format :

EXEC CICS

PUSH HANDLE

END-EXEC

EXEC CICS

POP HANDLE

END-EXEC

Execute Interface Block (EIB)

- CICS provides system related information through EIB fields which can be used by the program
- Some commonly used EIB fields
 - **EIBAID**
 - Contains current AID (Attention Identifier) Key pressed by the user
 - DFHAID copy book contains the following constants to identify keys
 - ✓ DFHENTER Enter key
 - ✓ DFHPF1, DFHPF2 ... Function keys
 - ✓ DFHCLEAR Clear key
 - ✓ DFHPA1 – DFHPA3 Attention keys
 - Use COBOL statement ‘COPY DFHAID’ in working-storage section
 - Example :

```
IF EIBAID = DFHENTER
    PERFORM READ-PARA
END-IF
```

Execute Interface Block (EIB) (contd)

- **EIBCALEN**
 - Length of Communication Area
 - Value '0' (zero) if CommArea is not sent
- **EIBCPOSN**
 - Position of CURSOR
- **EIBDATE**
 - Task DATE (00YYDDD)
- **EIBTIME**
 - Task Time (0HHMMSS)
- **EIBTRNID**
 - Transaction ID of Task
- **EIBFN**
 - Contains code corresponding to last CICS command issued
- **EIBRCODE**
 - Contains Response code of the last CICS command

MAP commands

CICS command - SEND MAP

- Used to send Map to the screen
- Format :

```
EXEC CICS
    SEND    MAP(name)
            MAPSET(name)
            [ FROM(data-area) ]
            [ LENGTH (data length) ]
            [ MAPONLY | DATAONLY ]
            [ ERASE | ERASEAUP ]
            [ CURSOR [ (value) ] ]
            [ ALARM ]
            [ FRSET ]
END-EXEC
```

SEND MAP Options

- **ERASE**
 - To erase screen before displaying the map
- **ERASEAUP**
 - To erase all unprotected fields
- **FREEKB**
 - To free the keyboard after displaying the map
- **ALARM**
 - To make an alarm beep after the map is displayed
- **FRSET**
 - To reset MDT to zero
- **DATAONLY**
 - Only application program data in the symbolic map is sent to the terminal
- **MAPONLY**
 - Only the default data from the physical map is sent to the terminal
 - FROM option should not be specified while using MAPONLY option
- **CURSOR**
 - To specify cursor position dynamically

Cursor positioning techniques

Cursor positioning while issuing SEND MAP command

- Static Positioning
- Dynamic / Symbolic Positioning
- Dynamic / Direct Positioning

Static Positioning (through MAP definition)

- Place 'IC' in the ATTRB parameter of DFHMDF macro of a particular field
- When map is sent, cursor will appear in this field
- If more than one field has 'IC' attribute, last one will be effective

Symbolic Positioning (through application program)

- move -1 into the filed length field (fieldname+L) for the field in which cursor is required
- SEND MAP command to be issued must have CURSOR option without value
- MAPSET must be coded with MODE=INOUT in DFHMSD
- This is the easiest and convenient method

Cursor positioning techniques

Direct Positioning (through application program)

- Use cursor (value) operand with SEND MAP
- The value specified is relative to the top-left position which is 0 .
- To position the cursor at (row, col), code CURSOR (value) where
value = (row -1) * 80 + (col -1)
- This is not a very convenient method for maps
- Example :

To position the cursor on a field at (11,21) while sending the map :

```
EXEC CICS SEND
  MAP('EMPMAP')
  MAPSET('EMPSET')
  CURSOR ( 820 )
  ERASE
END-EXEC
```

CICS command - RECEIVE MAP

- Used to receive data from the screen Map to the screen
- Format :

```
EXEC CICS  
    RECEIVE MAP(name)  
        MAPSET(name)  
        [ INTO(data-area) ]  
END-EXEC
```

- Fields received with RECEIVE Command
 - Fieldname +L - Contains actual number of characters typed
 - Fieldname +F
 - ✓ Flag byte field. Usually contains X'00'
 - ✓ Contains X'80' if user modifies the field but does not enter data
 - Fieldname +I
 - ✓ Contains the input data. Filled with x'00' if no data is entered

MAP Commands Exceptions

SEND MAP

- **INVMPsz**
 - Occurs if the specified map is too wide or too long for the terminal
- **LENGERR**
 - Occurs when the LENGTH option is zero or negative

RECEIVE MAP

- **INVMPsz**
 - Occurs if the specified map is too wide or too long for the terminal
- **INVREQ**
 - Occurs when a RECEIVE MAP command is issued in a task that is not associated with a terminal
- **MAPFAIL**
 - Occurs when program issues a RECEIVE MAP command to which the terminal operator responds by pressing one of the following:
 - ✓ The CLEAR key
 - ✓ Any PA key
 - ✓ ENTER without entering data
 - ✓ Any PF key without entering data.

HANDLE AID Command

- Used to specify paragraph name to which control is passed based on AID key pressed by the user
- Using EIBAID is better option than using this command
- Syntax :

```
EXEC CICS  
      HANDLE AID  
      option (procedure name)...  
END-EXEC
```

HANDLE AID Options

- | | |
|------------|--|
| • PA1-PA3 | Program Attention Keys |
| • PF1-PF24 | Program Function Keys |
| • ENTER | The ENTER Key |
| • CLEAR | The CLEAR Key |
| • ANYKEY | Any key not Specified (Except the ENTER key) |

CICS Command - RETURN

- Used to terminate the transaction and return control to CICS
- Format :

```
EXEC CICS  
    RETURN [ TRANSID(name) ]  
    [ COMMAREA(data-area) ]  
    [ LENGTH(data-value) ]  
END-EXEC
```

- **TRANSID**
 - Specifies transaction id to be invoked when user presses an attention key
- **COMMAREA**
 - Specifies a data area that is passed to the next execution of pseudo-conversational program
 - The next program execution accesses this data via its DFHCOMMAREA field in Linkage section
- **LENGTH**
 - Length of communication area specified by a half word binary (S9(4) COMP) or literal

CICS Command - ABEND

- Terminates the program abnormally
- Causes the message to be displayed on the terminal
- To continue after abend, user has to clear the screen and enter Trans-ID
- Format :

```
EXEC CICS  
    ABEND [ ABCODE ( name ) ]  
END-EXEC
```

- ABCODE is a 4 character code
- If ABCODE is specified, CICS will produce storage dump of the program
- ABCODE value will be used to identify the dump

File Handling

File Handling

- CICS File Control Program (FCP) provide application programs with services to read, write, update, add and delete records in a file
- It also manages exclusive control over records in order to maintain data integrity
- CICS supports VSAM files - KSDS, ESDS and RRDS
- CICS application program should not open/close any file. The file must be opened by CICS at the CICS startup time
- The characteristics of files and also their initial status must be defined in FCT
- The application program should not include any of the file related COBOL statements / clauses

File Handling - Services of File Control

Data Independence

- Program is independent of the structure of the file or data access methods
- Data Access Method is transparent to the programmer

File Open/Close

- Opening and closing of the files automatically performed by CICS

Exclusive Control during updates

- CICS ensures that if a task is updating a record, other tasks are excluded from doing so

CICS command – READ FILE

- Read a specific record specified by the key value specified in RIDFLD
- Data content will be moved into the record area defined in Working Storage Section
- File name defined in FCT should be used
- LENGTH indicates record size
- If RRN or RBA is specified, RIDFLD is interpreted as Relative Record Number (RRDS) or Relative Byte Address (ESDS)
- Format :

```
EXEC CICS
  READ      DATASET (name) | FILE (name)
            INTO (data-area)
            RIDFLD (Key)
            [ RRN / RBA ]
            [ LENGTH (record size) ]
END-EXEC
```

READ FILE - Exceptions

- **DUPKEY**
 - Occurs if a record to be retrieved by way of an alternative index has multiple occurrences
- **FILENOTFOUND**
 - Occurs if the file cannot be located
- **INVREQ**
 - Occurs if request is invalid due to number of reasons
- **LENGERR**
 - Occurs when the length of a record read with the INTO option specified does not match with LENGTH option
- **NOTAUTH**
 - Occurs when a resource security check is unsuccessful on the file
- **NOTFND**
 - Occurs if the record is not found
- **NOTOPEN**
 - Occurs if the file is closed or un-enabled.

CICS command – WRITE FILE

- Adds a record to the file
- Format is similar to READ

Format :

```
EXEC CICS
    WRITE      DATASET (name) | FILE (name)
              FROM (data-area)
              RIDFLD (Key)
              [ RRN / RBA ]
              [ LENGTH (record size) ]
END-EXEC
```

WRITE FILE - Exceptions

- **DUPREC**
 - Occurs if a record with the key already exists
- **FILENOTFOUND**
 - Occurs if the file cannot be located
- **INVREQ**
 - Occurs if request is invalid due to number of reasons
- **LENGERR**
 - Occurs when the length of a record does not match with LENGTH option
- **NOSPACE**
 - Occurs if no space available on the DASD for adding the record to the file
- **NOTAUTH**
 - Occurs when a resource security check is unsuccessful on the file
- **NOTOPEN**
 - Occurs if the file is closed or un-enabled.

Programming Approaches

CICS Programming Techniques

Techniques to develop Pseudo-Conversational Transaction

- Multiple Transaction Identifiers & Multiple programs
- Multiple Transaction Identifiers & 1 program
- 1 Transaction Identifier & 1 Program

CICS Programming Techniques

Multiple Transaction Identifiers & Multiple programs

Approach

- Uses multiple transaction identifiers (PCT) and multiple programs (PPT).
- Conversational Program logically & physically divided into separate programs
- Each program is assigned unique ID
- Before terminating, each program issues RETURN command with next transaction identifiers
- Last RETURN does not have transaction identifier.

Advantage

- Easy to develop

Disadvantage

- Increases the number of PCT and PPT entries
- Redundancy in programming

CICS Programming Techniques

Multiple Transaction Identifiers & 1 program

Approach

- Uses Multiple PCT entries and 1 PPT entry
- All programs are combined into 1 physical program
- Physical program has separate functional routines which can be identified by the transaction ID (EIBTRNID)

Advantage

- Reduces number of programs (PPT entries)

Disadvantage

- Creates many transaction identifiers (PCT entries).

CICS Programming Techniques

1 Transaction Identifier & 1 Program

Approach

- Uses 1 PCT entry & 1 PPT entry
- Physical program consists of multiple logical programs
- Each logical program takes care of 1 conversation
- When logical program terminates, it issues RETURN commands
- RETURN command is accompanied by transaction id and a piece of data through COMMAREA.
- COMMAREA indicates next logical path on returning

Advantage

- Efficiency increased - Requires only 1 PCT & 1 PPT

Program Control

CICS Commands - LINK

- Used to pass control from one program to a program at next lower level
- Expects control to be returned the previous program
- Data can be passed through COMMAREA
- Has more overhead than XCTL
- Syntax :

```
EXEC CICS  
  LINK      PROGRAM (program name)  
            [ COMMAREA(data-area) ]  
            [ LENGTH(data-value) ]  
END-EXEC
```

CICS Commands - XCTL

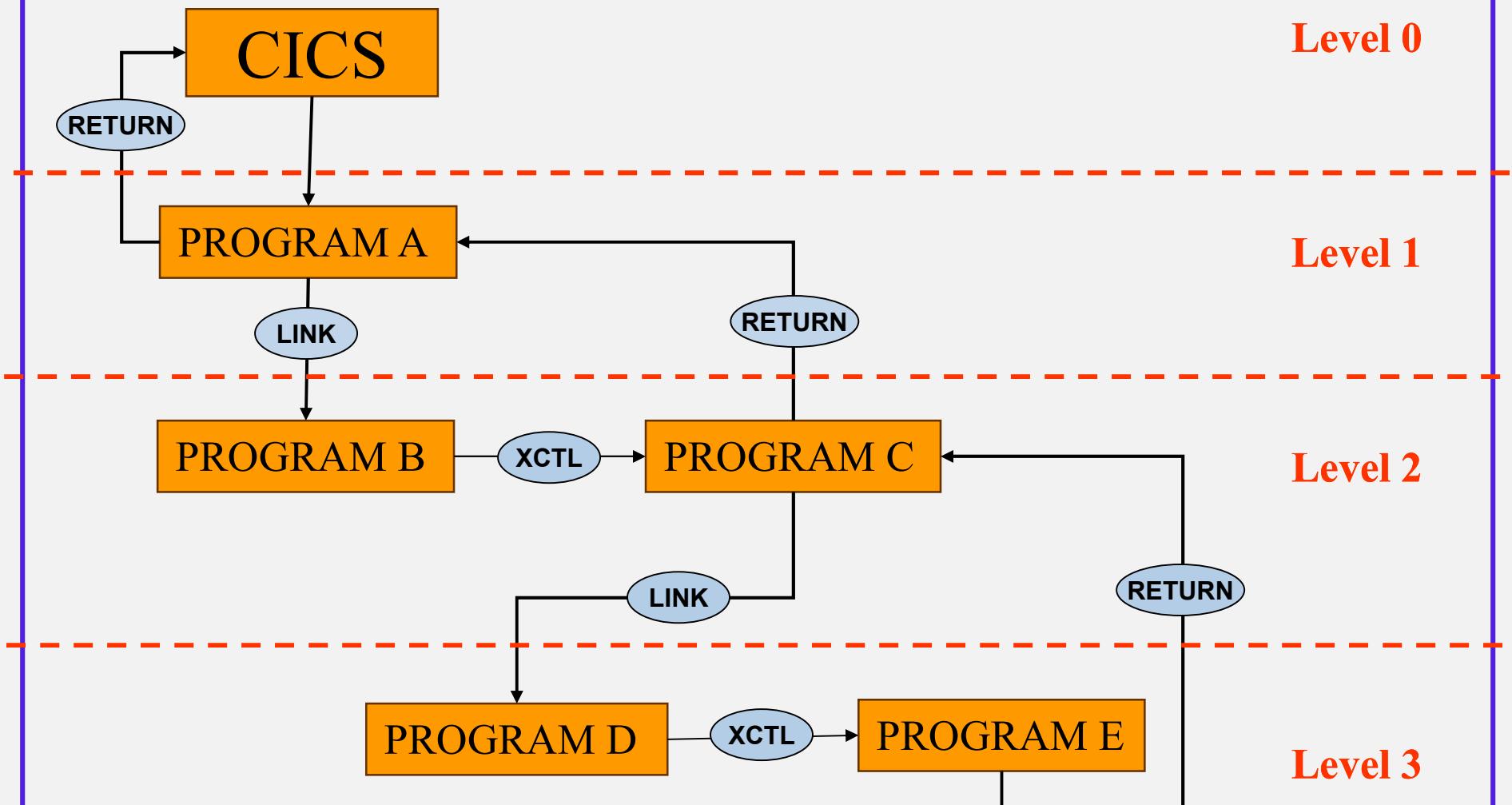
- Used to pass control from one program to the other at the same logical level
- Does not expect control to be returned
- The program from which the control is passed is released
- Data can be passed through COMMAREA
- Has little overhead
- Syntax :

```
EXEC CICS  
  XCTL    PROGRAM (program name)  
          [ COMMAREA(data-area) ]  
          [ LENGTH(data-value) ]  
  END-EXEC
```

LINK vs XCTL

XCTL	LINK
does not expect control back.	expects control back.
Handle conditions in the main program are not available to the called program. Establish new handle conditions.	Handle conditions in the main program will be available to the called program. Called program may use PUSH and POP statements to suspend and restore them
less overhead on system resources like memory etc	more overhead in this case on memory etc.
Similar to GO TO.	Similar to PERFORM

Application Program Logical Levels



Handling Input data

Handling Input Data

- In actual business applications, dealing with numeric data in the fields is quite complex as the data entered by the terminal user cannot be stored in the file as it is due to some of the reasons --
 - presence of numeric sign (COBOL does not recognize sign as part of numeric value)
 - presence of decimal point (Decimal point is not allowed as part of numeric data in COBOL)
 - invalid entry of numeric data (example : decimal point entered twice)
- Output can be easily formatted with PICOUT with editing characters
- Some special techniques are required to handle numeric sign and decimal point while receiving input

Handling Input Data

Handling Numeric Sign

- Separate Amount Field Approach
- CR / DB Approach

Handling Decimal Point

- Virtual Decimal Point Approach
- Separate Dollar / Cents Field Approach

Handling Both

- Money Edit Subroutine Approach

Separate Amount Field Approach

- Create separate Credit and Debit fields
- Define the fields with appropriate PICIN parameter (example : PIC 999V99)

BALANCE :	CREDIT (+)	OR	DEBIT (-)
	XXXXXXX		XXXXXXX

- Instruct users to enter only absolute values in one of two fields, depending on the sign
- In the program, move credit field to a numeric field or move debit field to the same field and invert the sign
-

Example :

```
IF BALCRI > ZERO  
    MOVE BALCRI TO BALAMT  
END-IF  
IF BALDBI > ZERO  
    MOVE BALDBI TO BALAMT  
    COMPUTE BALAMT = BALAMT * -1  
END-IF
```

CR / DB Approach

- Create CR / DB field near the numeric field
- Instruct users to enter CR / DB after entering numeric value

	AMOUNT	CR(+) / DB(-)
BALANCE :	XXXXXXX	XX

- In the program check for DB in the sign field

Example :

```
MOVE BALI TO BALAMT  
IF SIGNI = 'DB'  
    COMPUTE BALAMT = BALAMT * -1  
END-IF
```

Virtual Decimal Point Approach

- Create a numeric field with virtual decimal point
- Define appropriate PICIN parameter
- Instruct the users to enter the absolute data (without decimal point) by providing help text
- Optionally, a couple of examples can be provided

BALANCE (format : 9999V99) : XXXXXXXX

(Example : For 34.56 Enter 0003456)

Separate Dollar / Cents Field Approach

- Create the dollar and cents fields separately
- Instruct the users to enter both separately

BALANCE :	DOLLARS	CENTS
	XXXXX	XX

- In the program, merge both the fields as single amount
- Example :

COMPUTE BALAMT = DOLLI + CENTSI / 100

Money Edit Subroutine Approach

- In this approach, develop a subroutine to deal with sign and decimal point in a character string field
- Find numeric sign starting from right to left
- If ‘-‘ is found, move to sign indicator (say SIGNF)
- Find decimal point backwards (right to left)
- 2 numeric character right of the decimal is cents. Move this to CentsAmt
- Numeric characters left of the decimal is dollar amount. Move this to DollAmt
- Merge these two values (DollAmt and CentsAmt) into a single value as shown in the previous slide
- Use sign indicator (SIGNF) to decide the sign and convert the value into negative if required

Data Validity Checking

- Performed after RECEIVE MAP command
- **Check the fieldname +L**
 - If zero, no data has been entered
 - If positive, data has been entered, so validate
- **Check the fieldname +I**
 - If LOW-VALUES or SPACE no data has been entered
 - If not LOW-VALUES or SPACE data has been entered, so validate
- Perform validation for all the fields
- Send the error messages to the terminal
- **Check for**
 - Valid data type such as integer, character
 - Valid range of acceptable values such as hhmmss
 - Valid number of digits or characters allowed
 - Valid logical relationship to other data items

Attribute Byte Manipulation

- Attribute character is one byte of data with the following Bit pattern

BITS	Function
0, 1	Value determined by the contents of 2 to 7
2	0 : Unprotected , 1 : Protected
3	0 : Alphanumeric, 1 : Numeric
2,3	11 Autoskip
4, 5	0 0 Normal intensity 1 0 High intensity 1 1 Dark
6	Always 0
7	MDT Bit (0 : OFF 1: ON)

- Attribute character (field+A) can be modified in the program with the necessary bit pattern to change attributes dynamically

Attribute Byte Manipulation

- CICS provides standard attribute character list in the form of COPYLIB member DFHBMSCA
- The list covers most of the combinations of the attributes
- Some examples from the character list :
 - **DFHBMFSE** Unprotected MDT on
 - **DFHUNNUM** Unprotected , MDT on, Numeric
 - **DFHUNIMD** Unprotected, High intensity, MDT on
- These characters can be used to change the value of Attribute byte of the field after copying DFHBMSCA into the program

Advanced File Handling

File Handling: READ with GENERIC option

- Used to read a non-specific record based on higher part of the key (generic)
- Used when the full information of the key is not available
- Useful to access records when full key is not known
- Reads the first record with matching partial key

Format :

- READ command used with GENERIC keyword
- Length of the generic key mentioned in the KEYLENGTH option
- Generic key data mentioned in the RIDFLD field

File Handling: READ with GENERIC option

Example :

```
MOVE 35 to WK-LEN  
MOVE 'NY' TO REC-A-KEY  
EXEC CICS  
  READ    DATASET ('EMPFIL')  
        INTO ( EMPREC )  
        RIDFLD ( REC-A-KEY )  
        KEYLENGTH( 2 )  
        GENERIC  
        LENGTH ( WK-LEN )  
END-EXEC
```

Reads the first record
with key value starting
with 'NY'

File Handling: READ with GTEQ option

READ command with GTEQ option

- Used to read a non-specific record whose key is equal to or greater than the key value specified
- Useful when the full key is known but not sure if the record with that key exists or not
- Full key is required to read the record

Format

- READ command used with GTEQ keyword
- Specify the key value in RIDFLD

File Handling: READ with GTEQ option

Example :

```
MOVE 35 TO WK-LEN  
MOVE 'NY003' TO REC-A-KEY  
EXEC CICS  
  READ    DATASET ('EMPFIL')  
        INTO ( EMPREC )  
        RIDFLD ( REC-A-KEY )  
        GTEQ  
        LENGTH ( WK-LEN )  
END-EXEC
```

Reads the record with key value equal to or greater than 'NY003'

File Handling: UPDATE and REWRITE

- A Combination of READ command with UPDATE option and REWRITE command is used to update a record
- Between these commands, exclusive control over the record is maintained for this task so that other tasks cannot access this record for updates
- Due to this, interval between these two commands should be as short as possible

Format :

- Use UPDATE option in READ command
- Format of REWRITE command similar to WRITE command
- DATASET in REWRITE should be same as the one in the READ command

File Handling: UPDATE and REWRITE

Example :

```
EXEC CICS
```

```
    READ     DATASET ('MPFIL')
            INTO (EMPREC)
            RIDFLD (REC-A-KEY)
            LENGTH ( WK-LEN )
```

```
    UPDATE
```

```
END-EXEC
```

```
.....
```

```
.....
```

```
EXEC CICS
```

```
    REWRITE DATASET ('MPFIL')
              FROM (EMPREC)
              LENGTH (WK-LEN)
```

```
END-EXEC
```

Make changes to
the record here

RIDFLD not required
as same record is to
be rewritten

File Handling: UNLOCK command

- READ with UPDATE option maintains exclusive control until
 - the record is updated by REWRITE command
 - the transaction is normally or abnormally completed
- At times the update may not be required for the record retrieved with READ with UPDATE option
- In such case, UNLOCK command is used to release exclusive control
- Format :

```
EXEC CICS  
    UNLOCK DATASET (name)  
END-EXEC.
```

File Handling: DELETE command

- Used to delete one record or group of records
- There are three approaches to using DELETE command
 - DELETE after READ/UPDATE Approach
 - Direct DELETE Approach
 - Group Record DELETE Approach

File Handling: DELETE after READ / UPDATE

- Issued without the RIDFLD field
- Performed after READ command with UPDATE option
- Record read by READ with UPDATE option will be deleted
- Format :

```
EXEC CICS  
    DELETE    DATASET (name)  
END-EXEC
```

File Handling: Direct DELETE

- Issued with full key information in the RIDFLD field
- Record specified will be directly deleted from file
- No need to read in advance
- RIDFLD is required
- Format :

```
EXEC CICS  
      DELETE  DATASET ( name )  
              RIDFLD ( key value )  
END-EXEC
```

File Handling: Group DELETE

- DELETE command is issued with GENERIC option
- All records satisfying the GENERIC (partial) key are deleted
- GENERIC and KEYLENGTH options are required in DELETE command
- NUMREC option makes the number of records deleted available in the variable which is declared as half word binary
- Format :

```
EXEC CICS  
    DELETE DATASET (name)  
          RIDFLD (Rec-Key)  
          KEYLENGTH (key-length)  
          GENERIC  
          NUMREC(rec-count)  
END-EXEC
```

CICS Programming Techniques:

Double Updating during Pseudo-Conversation

- Wrong update of resource
- No exclusive control on the resource
- Results in loss of data integrity

Cause

- Between 2 tasks exclusive control is terminated
- If another update attempt is made, double update occurs

CICS Programming Techniques:

Double Updating during Pseudo-Conversation

Solution

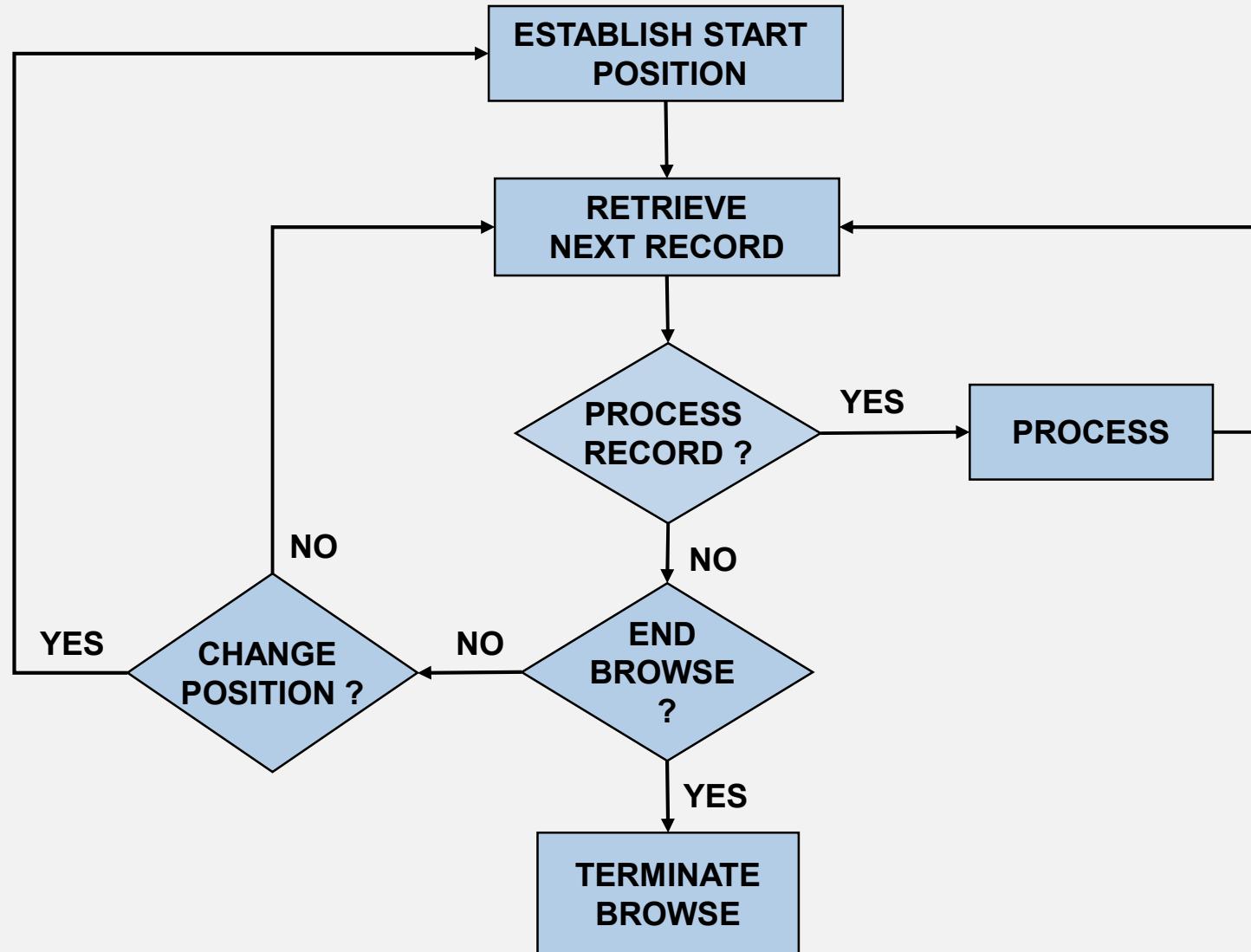
- By Programming Convention
- By update identifier
- Update Identifier
 - Half Word Binary (S9(4)COMP)
 - Contains update counter
 - Included in every record to be updated
 - To update, take increments update identifier by 1
 - Passed to next task through COMMAREA
 - Before updating in next task update identifier of record and that passed through COMMAREA are compared
 - ✓ Same - No other task is accessing the resource
 - ✓ Different - Another task is accessing the resource so task cancels itself.

File Handling: Browsing

- Sequential access of VSAM file under CICS is called 'Browsing'
- Updation or deletion not allowed. Only browsing is allowed
- Commands :

STARTBR	To establish the starting position for the browse
READNEXT	To read the Next record
READPREV	To read the previous record
RESETBR	To reestablish another position for a new browse
ENDBR	To end browsing operation

File Handling: Browsing



File Handling: STARTBR Command

- Establishes a browse starting position for a file
- Used for establishing the position only
- Actual record is read by READNEXT or READPREV command
- Format :

```
EXEC CICS  
    STARTBR  DATASET(name) | FILE (name)  
            RIDFLD (data-area)  
            GTEQ | EQUAL | GENERIC  
END-EXEC.
```

- **GTEQ** : Position the first record whose key is greater than or equal to RIDFLD
- **EQUAL** : Position the record whose key value is equal to the value in RIDFLD
- **GENERIC** : Similar to the option in READ command

- Some Common Exceptions : DISABLED, NOTFND, INVREQ, NOTOPEN, NOTAUTH

File Handling: READNEXT Command

- Used to read a record the next record
- STARTBR must be successfully completed prior to READNEXT
- Format :

```
EXEC CICS
```

```
    READNEXT    DATASET(name) | FILE (name)  
              INTO (data-area)  
              LENGTH (data-value)  
              RIDFLD (data-area)
```

```
END-EXEC
```

- Some Common Exceptions : INVREQ, DUPKEY, ENDFILE, LENGERR, NOTAUTH, NOTFND

File Handling: READPREV Command

- Used to read a record of a file backwards
- STARTBR must be successfully completed prior to READPREV
- Format :

```
EXEC CICS
```

```
    READPREV   DATASET(name) | FILE (name)  
              INTO (data-area)  
              LENGTH (data-value)  
              RIDFLD (data-area)
```

```
END-EXEC
```

- Some Common Exceptions : INVREQ, DUPKEY, ENDFILE, LENGERR, NOTAUTH, NOTFND

File Handling: RESETBR Command

- To reestablish another starting point within the same browse operation
- Only used for positioning
- Reading will be performed by READNEXT / READPREV command
- Can change the characteristics of browse (GTEQ, EQUAL, GENERIC)
- Format :

```
EXEC CICS  
    RESETBR  DATASET (name) | FILE (name)  
              RIDFLD (data-area)  
              GTEQ | EQUAL | GENERIC  
END-EXEC.
```

- Some Common Exceptions : INVREQ, NOTOPEN, NOTAUTH

File Handling: ENDBR Command

- To terminate the browse operation initiated by STARTBR command
- Format :

```
EXEC CICS  
    ENDBR      DATASET (name) | FILE (name)  
END-EXEC.
```

- Some Common Exceptions : INVREQ, NOTOPEN, NOTAUTH

File Handling: Update during Browse

- Browsing and updating are mutually exclusive
 - Updating during Browsing requires special programming technique
-
- After READNEXT issue ENDBR (to temporarily suspend the browse operation)
 - Issue READ with the UPDATE option
 - Issue REWRITE for updating the record
 - Issue STARTBR for resuming browser operation
 - Issue READNEXT to read the record that has been updated
 - Issue another READNEXT to skip the record that has been updated and continue browsing

Text Handling

Text Handling

- BMS offers function to send text to the screen without the need for a formatted screen
- SEND TEXT command is used for this purpose
- Features :
 - Doesn't require a formatted screen definition
 - The text to be sent is stored in a working-storage variable
 - Helpful in sending large amount of unformatted information onto the screen
 - Header and trailer can be added to the text
 - Several text blocks can be sent as a single message each block being stored in a different working-storage variable
 - BMS takes care of newline character and word alignment

SEND TEXT Command

- Used to send a single stream text to the terminal
- BMS maps not required
- Text line automatically broken at word boundary
- Format :

```
EXEC CICS
```

```
    SEND TEXT    FROM (data-value)
                  LENGTH (data-value)
                  [HEADER (data-value)]
                  [TRAILER (data-value)]
                  [ERASE]
```

```
END-EXEC.
```

SEND TEXT Command

Where

- FROM - Defines the text body field from the Working Storage Section
- LENGTH - Half Word Binary(S9(4) COMP) indicates length of the text excluding HEADER and TRAILER
- HEADER - Name of header data
- TRAILER - Name of trailer data
- ERASE - Indicates the screen be cleared before text is displayed

SEND TEXT Command

Header

- Appears at the top of the screen followed by the text
- Includes
 - 2 byte : length of the text body
 - 1 byte : character denoting the page number position in the text body
 - 1 byte : control field used by BMS
 - Text body
- Example :

```
01 TEXT-HEADER.  
 05 FILLER PIC S9(4) COMP VALUE 16.  
 05 FILLER PIC X      VALUE '#'.  
 05 FILLER PIC X.  
 05 FILLER PIC X(10) VALUE 'HEADER TEXT : ##'.
```

Footer

- Appears at the bottom of the screen
- Has same format as Header

SEND TEXT Command

MULTIPLE TEXT BLOCKS

- Used to send several paragraphs of text as a single logical text
- Requires two commands
 - Use SEND TEXT with ACCUM option for accumulating text blocks
 - Issue SEND PAGE command to send the entire text
- Format :

```
EXEC CICS
    SEND TEXT      FROM (data-value)
                  LENGTH (data-value)
                  [ HEADER (data-value) ]
                  ACCUM
                  [ERASE]
END-EXEC.
```

```
EXEC CICS
    SEND PAGE      [ TRAILER (data-value) ]
END-EXEC
```

SEND CONTROL Command

- Send device control instructions to the screen
- No map is sent
- Format :

```
EXEC CICS  
  SEND CONTROL [ERASE]  
    [ERASEAUP]  
    [ALARM]  
    [FREEKB]  
    [FRSET]  
END-EXEC.
```

Queues

Queues

Temporary Storage Queue (TSQ)

Transient Data Queue (TDQ)

Temporary Storage Queues

- Each TSQ contains one or more records referred as 'items'
- TSQ is identified by a unique 8 characters identifier
- TSQs are created dynamically
- When a program tries to write to a TSQ that does not exist, it is automatically created
- Queue IDs Need not be defined in advance
- Items in TSQ can be accessed Random or Sequentially
- Items can be added or existing items can be updated
- Items accessed through item numbers
- Although you can store many items in TSQ, it is uncommon to have more than one item
- TSQ remains until it is deleted

TSQ – WRITEQ TS Command

- Used to write new item to a TSQ or update existing item
- ITEM option is required for REWRITE
- If the TSQ does not exist, it will be created
- Format :

```
EXEC CICS
  WRITEQ TS    QUEUE ( TSQid )
               FROM (tsq-data)
               LENGTH (tsq-length)
               ITEM (TSQ-item)
               [ REWRITE ]
               [ NOSUSPEND ]
               [ MAIN | AUXILLARY ]
END-EXEC
```

- Common Exceptions : ITEMERR, LENGERR, NOSPACE, QIDERR

TSQ – WRITEQ TS Command options

- **FROM**
 - Working-storage area that contains record to be written
- **LENGTH**
 - Length of data to be written
- **ITEM**
 - Item number through half word binary variable for REWRITE option
 - Without REWRITE option CICS places the item number in the variable
- **REWRITE**
 - To update existing record
- **NOSUSPEND**
 - Not to suspend the task even if NOSPACE condition occurs
- **MAIN / AUXILIARY**
 - Where to create TSQ (main storage or external VSAM file)

TSQ – READQ TS Command

- Used to read an item from a TSQ
- Format :

```
EXEC CICS
    READQ TS      QUEUE ( TSQid )
                  INTO ( tsq-data )
                  [ LENGTH ( tsq-length ) ]
                  [ ITEM ( TSQ-item ) | NEXT ]
                  [ NUMITEMS ( data-area ) ]
END-EXEC
```

- Common Exceptions : ITEMERR, LENGTHERR, QIDERR

TSQ – READQ TS Command options

- **INTO**
 - Working-storage area that will contain record read from TSQ
- **LENGTH**
 - Maximum length of data to be read
 - If the length of the item is longer, LENGERR occurs
 - If the length of the item is smaller, CICS places the actual length in the variable
- **ITEM**
 - Item number through half word binary variable for random read
- **NEXT**
 - To read next item following last item read by any task
- **NUMITEMS**
 - CICS places number of items in the TSQ in this variable

TSQ – DELETEQ TS Command

- Used to delete the entire TSQ
- Format :

```
EXEC CICS  
    DELETEQ TS  QUEUE ( TSQid )  
END-EXEC
```

- Common Exceptions : QIDERR

Transient Data Queues

- A record passes into a TDQ when it is written. That record will be deleted when it is read
- TDQs are processed sequentially
- TDQ is identified by a unique 4 characters identifier
- TDQs are often called Destinations
- Unlike TSQs, TDQs should be defined in Destination Control Table (DCT)
- There are two types of TDQs
 - Intra partition TDQ – can be accessed only within CICS
 - Extrapartition TDQ – Can be accessed by programs outside CICS
- Intrapartition TDQ can be configured for Automatic Task Initiation (ATI)
- With the ATI, a task can be initiated automatically when the number of records reach a specified level (trigger level)

TDQ – WRITEQ TD Command

- Used to write new item to a TDQ
- Format :

```
EXEC CICS  
    WRITEQ TD    QUEUE ( TDQid )  
                  FROM (tdq-data)  
                  LENGTH (tdq-length)  
END-EXEC
```

- Common Exceptions : LENGERR, NOSPACE, QIDERR

TDQ – READQ TD Command

- Used to read an item from a TDQ
- The item is deleted from the queue automatically after READ
- Format :

```
EXEC CICS
  READQ TD    QUEUE ( TDQid )
              INTO ( tdq-data )
              [ LENGTH ( tdq-length ) ]
END-EXEC
```

- Common Exceptions : LENGERR, QIDERR, QZERO

TSQ – DELETEQ TD Command

- Used to delete all the records from TDQ
- TDQ itself is not deleted
- Format :

```
EXEC CICS  
  DELETEQ TD  QUEUE ( TDQid )  
END-EXEC
```

- Common Exceptions : QIDERR

TSQ vs TDQ

Temporary Storage	Transient Data
Data is read randomly.	Data must be read sequentially.
A data item can be read many times and it remains in the queue until the entire queue is purged.	A data item can be read only once.
Data can be changed.	Data cannot be changed
Temporary storage can be written to auxiliary or main storage.	Transient data items are always written to disk. Transient Data is ALWAYS associated with a destination

Interval Control

Interval Control

- **Interval Control** provides a variety of time-related features
- Used for the purpose of
 - Time synchronization
 - Multi-threading
 - Auto-Initiation

Interval Control - ASKTIME

- To request the current date and time
- ASKTIME updates the date (EIBDATE) and CICS time-of-day clock (EIBTIME) fields in the EIB
- These two fields initially contain the date and time of when the task started
- Format :

```
EXEC CICS  
  ASKTIME [ ABSTIME (data-area) ]  
END-EXEC
```

- ABSTIME
 - specifies the 15 digit packed-decimal field (PIC S9(15) COMP-3) where CICS places an absolute time value representing number of milliseconds elapsed from Jan 1 1900

Interval Control - FORMATTIME

- To retrieve information of date and time in various formats
- Format :

EXEC CICS

```
  FORMATTIME ABSTIME(abs-date)
              [ DATE(data-area) ]
              [ MMDDYY(data-area) ]
              [ DDMMYY(data-area) ]
              [ YYMMDD(date-area) ]
              [ YYDDMM(date-area) ]
              [ YYDDD(date-area) ]
              :
              [ DATESEP[(date-separator-char)] ]
              [ TIME(data-area) ]
              [ TIMESEP[(time-separator-char)] ]
```

END-EXEC

Default format

- data-area is 8 byte storage

Interval Control - FORMATTIME

- DATESEP
 - single character used as separator in date
 - If not specified no separator is used
 - If separator-char is not provided (/) is used
- TIMESEP
 - single character used as separator in time
 - If not specified no separator is used
 - If separator-char is not provided (:) is used

Interval Control - START

- Schedules a task in a local or a remote system
- Format :

EXEC CICS

```
    START  TRANSID (trans-id)
           [ INTERVAL ( hhmmss ) ]
           [ TIME ( hhmmss ) ]
           [ AFTER [ HOURS(hrs) ] [ MINUTES(min) ] [ SECONDS(sec) ] ]
           [ AT [ HOURS(hrs) ] [ MINUTES(min) ] [ SECONDS(sec) ] ]
           [ TERMID (terminal-id) ]
           [ REQID (req-id) ]
           [ FROM (data-area) ]
           [ LENGTH ( data-length) ]
```

END-EXEC

Interval Control - START

- INTERVAL / TIME
 - Specified as 6 digit constant. If specified as data area, it must be 4 byte packed-decimal (PIC S9(7) COMP-3)
- AFTER
 - Specifies that task should start after interval specified
- AT
 - Specifies that the task must start at the time specified
- TERMID
 - Specifies that the task should start on the terminal mentioned
 - If not specified, the task will run without a terminal
- REQID
 - Specifies 8 byte value that identifies the task
 - This can be used to issue CANCEL command
 - If omitted, CICS generates a request-id and returns it is EIBREQID
- FROM / LENGTH
 - Specify data and length of data to be passed to the task

Interval Control - RETRIEVE

- To retrieve data passed to it by the starting task
- Format :

```
EXEC CICS  
  RETRIEVE      INTO (data-area)  
                  [ LENGTH ( data-area-length ) ]  
END-EXEC
```

- If data-area-length is smaller than the data retrieved, LENGERR condition is raised

Interval Control - CANCEL

- To cancel the task scheduled earlier
- CANCEL command cannot be issued once the task is started
- Format :

```
EXEC CICS  
      CANCEL      REQID ( req-id)  
      END-EXEC
```

Task Control

Task Control

- Task Control refers to the CICS functions that manage the execution of tasks
- Dispatcher is one of the major components which keeps track of all current tasks and decides which of several waiting tasks should be given the control
-
- Application program can influence the dispatcher's operation through
 - SUSPEND Command
 - ENQ and DEQ Commands

Task Control - SUSPEND

- Normally, a program gives up control whenever it issues a CICS command while CICS fulfills the request
- In the meantime, dispatcher gives control to another task
- This way, several tasks are operational at the same time though only one of them is actually executing
- If a task is taking long time without executing a CICS command, it can suspend itself so that other tasks can get control
- Format :

```
EXEC CICS  
      SUSPEND  
END-EXEC
```

- It simply returns control to CICS and the current task is placed at the end of the list of tasks waiting to gain control

Task Control – ENQ / DEQ

- To gain exclusive control over a resource like TSQ, TDQ or a printer
 - ENQ is issued to gain exclusive control over a resource
 - Once ENQ is issued for a resource, other tasks cannot need that resource or suspended
 - DEQ is issued to release the resource
-
- Format :

```
EXEC CICS  
    ENQ    RESOURCE (name)  
END-EXEC
```

```
EXEC CICS  
    DEQ    RESOURCE (name)  
END-EXEC
```

CICS Transactions

CICS supplied transactions

- **CESN**
 - To sign on to CICS system
- **CESF**
 - To sign off from CICS
- **CEDA**
 - Resource Definition Online
 - DEFINE – to specify resource definition
 - INSTALL – to transfer the new definition to active CICS system
- **CECI**
 - Command interpreter
 - CICS commands can be issued directly to test them

CICS supplied transactions

- **CEMT**
 - Displays the status of CICS & system resources
 - Alter the status of CICS & system resources
 - Remove the installed resource definitions
 - Perform few functions that are not related to resources
- **CEDF**
 - Execution Diagnostic facility
 - Controlled through CEDF ON / OFF
 - Starts execution of each CICS command in the program
 - Status can be monitored at after execution of each CICS command

Here we come to the end of Training

Thank You