SAHASRA INFOTECH

IBM – Mainframes Material

SAHASRA INFOTECH & CONSULTING SERVICES

#301, Swaathi Manor's, Beside Aditya Trade Center Lane, Ameerpet,
Hyderabad – 500016, Andhra Pradesh, India.
Ph: 040-40037065, Mobile: 7799020208
info@sahasrainfotech.co.in, www.sahasrainfotech.co.in

SAHASRA INFOTECH
&
CONSULTING SERVICES

COBOL
(Common Business Oriented Language)

www.sahasrainfotech.co.in

IBM - MAINFRAMES

- A Mainframe is a Large Computer or Server which was designed in early 1950's by IBM.
- Mainframe was designed mainly to meet the business requirements. Almost 70% of the business applications are currently on mainframe platform.

Features or Characteristics of Mainframes:

- To store bulk amount of data
- High processing speed of 80 MIBS.
- Low maintenance.

Application where Mainframe is used:

Banking, Railways, Insurance, Airlines, Retail Marts, Telecom and Breakage (Stock Market).

Technical skills required to learn Mainframes:

- COBOL Programming Language (HLL) / User Friendly.
- JCL Compile and Execute the Program.
- VSAM Files.
- DB2 Tables.
- CICS Front End in MIF Environment.

COBOL (Common Business Oriented Language)

- COBOL was designed in 1960 by a team called CODASYL (Conference on Data Systems Language).
- COBOL is used as a programming language in mainframe environment.

JCL (Job Control Language)**:** JCL is used to compile and execute batch programs.

VSAM (Virtual Storage Access Method)**:** In VSAM, the data will be stored in files.

DB2 (Data Base 2)**:**

- DB2 is a second release of database by IBM. They have designed DB2 in the year 1983 with RDBMS. In DB2, the data will be stored in tables.
- COBOL, JCL, VSAM, DB2 are backend modules in mainframe.

CICS (Customer Information Control System)**:** CICS acts as front end in mainframe platform.

COBOL

COBOL was developed initially in the year 1959 by team called conference on data system language (CODASYL), it was mainly developed to meet business requirement.

HISTORY OF COBOL:

- When they developed in the year 1959 there was no compiler available, then CODASYL team has approached ANSI to develop COBOL compiler.
- ANSI has developed the COBOL compiler and the COBOL compiler was available in the year 1968.
  - COBOL – 1968 (CODASYL)
  - COBOL – 1974 (ANSI released 1st Version VS-COBOL-I)
  - COBOL – 1985 (ANSI released 1st Version VS-COBOL-II)

FEATURES OF COBOL:

- HLL / very much English like language.
- Capable of strong large amount of data.
- Platform independent (it can in any other OS).
- Structured language, Case Insensitive.

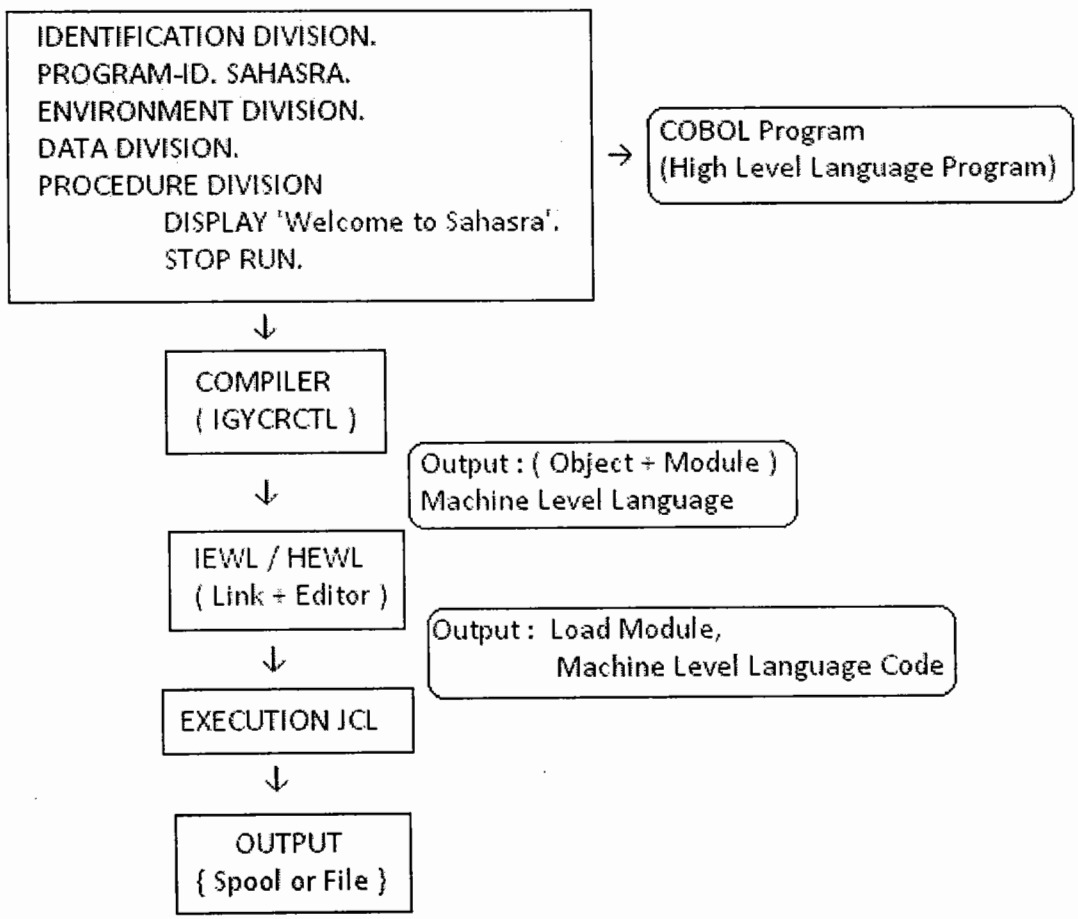
STRUCTURE OF COBOL LANGUAGE:

IDENTIFICATRION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.

DIVISIONS → SECTIONS → PARAGRAPHS → SENTENCES → STATEMENTS → WORDS
↓
CHARACTERS (A-Z, 0-9, Special characters)

Program Preparation for COBOL:

- Write a COBOL Program.
- Compile and Execute the Program.



IDENTIFICATION DIVISION:

- It should be the 1st statement in any COBOL program. It is mainly used for doc purpose. It is further divided into paragraphs.

Syntax:

IDENTIFICATION DIVISION.
PROGRAM-ID. PGMNAME.
[AUTHOR: USERNAME
DATE-WRITTEN DD/MM/YYYY
DATE-COMPILED DD/MM/YYYY]

Only program-id is mandatory paragraph in identification division.

ENVIRONMENT DIVISION:

- It is used to identify the machine names and file names used in the program.

Syntax:

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE COMPUTER. IBM-3270 →(It is for Compilation)
OBJECT COMPUTER. IBM-3270 →(It is for Execution)
INPUT-OUTPUT SECTION
FILE-CONTROL →(Define files)

- Source computer specifies the machine on which the program is compiled.
- Object computer specifies the machine by which the program is extended.
- File control is used to identify all the files used in the program.

DATA DIVISION:

- It is mainly used for declaration purpose.
- This is divided into three sections.

1. File Section: File section is used for declaring the file variables.

2. Working-Storage Section: Working-storage section is used to declare all temporary variables used in program.

3. Linkage Section: Linkage section is used to declare all the sub-program variables.

PROCEDURE DIVISION:

- This is the division where application developer spends 90% office time and the programming language is coded in this division.

COBOL Coding Sheet: It is a 24 / 80 Sheet.

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

Page Numbers 1 to 3) Line Numbers (4 to 6)	* / D –	Area - A / Margin - A Divisions Sections Paragraphs Level Numbers	Area - B / Margin - B Procedure Division Statements	Identification Fields **OR** Small Comments (Characters will be ignored by the computer at these Level)

P1. Write a program to display a message on the screen.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ADDPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
        DISPLAY 'WELCOME TO SAHASRA'.
        STOP RUN.
```

Note: STOP RUN should be the last statement in any COBOL program. Once the control comes across 'STOP RUN' statement it passes control to OS for future processing.

COMPILE:

- For every program we need not write compile JCL. The compile JCL will be standard for the entire program's still VSAM module.
- Just we need to copy the compile JCL from the existing PDS and make the necessary changes.

JCL (JOB CONTROL LANGUAGE):

- JCL is used to compile and execute batch programs, to write any JCL we require JOB Statement, EXEC Statement, DD Statement.

JCL Coding Sheet:

123--------COLUMN NUMBERS----------72	73----------80
//JOBNAME JOB PARAMETER1, PARAMETER2.....	**COMMENTS**

(Where // → Identification Field
JOBname → Naming Field
JOB, EXEC, DD → Statement / Operation
// EXEC
// DD
//* → Comment
// → End of JCL

JOB STATEMENT:

- JOB statement is used to identify the JOB Name and JOB's related parameters (accounting info, username, class, notify etc).

Accounting Information:

- Accounting information is mainly used for billing purpose.

Note: Suppose we submit any JOB in real time it is going to take some CPU time. Based on CPU time some '$' amount involved where this amount has to go will be decided by 'accounting info' parameter.

User name:

- This is used to identify who has written the JCL.

Class:

- Class parameter is used to assign priorities to the JOBs.

Note: Assume we have submitted 10 JOBs at the same time which JOB runs first and which runs 2nd will be decided by class parameter.

Notify:

- To which user id the JOB has to be notify after successful or unsuccessful compilation will be decided by notify parameter.

EXEC Statement:

- This statement is used to identify the step name and program name.

Note: maximum we can write 255 statements in a single JOB.

Steplib:

- This is used to identify the path of the load module (ML language code).

Sysprint:

- This is used to print the error messages in the spool.

Sysout:

- If the program is successful the o/p will be routed to sysout of spool.

// → it is used to indicate the end of the JCL.

```
//DISPGM     JOB    123,'SAHASRA',CLASS=A-Z/0-9,NOTIFY=&SYSUID/MF ID
//STEP1      EXEC   PGM=DISPGM
//STEPLIB    DD     DSN=USERID.NAME.LOADLIB,DISP=SHR
//SYSPRINT   DD     SYSOUT=*
//SYSOUT     DD     SYSOUT=*
//
```

LAB MODULE:

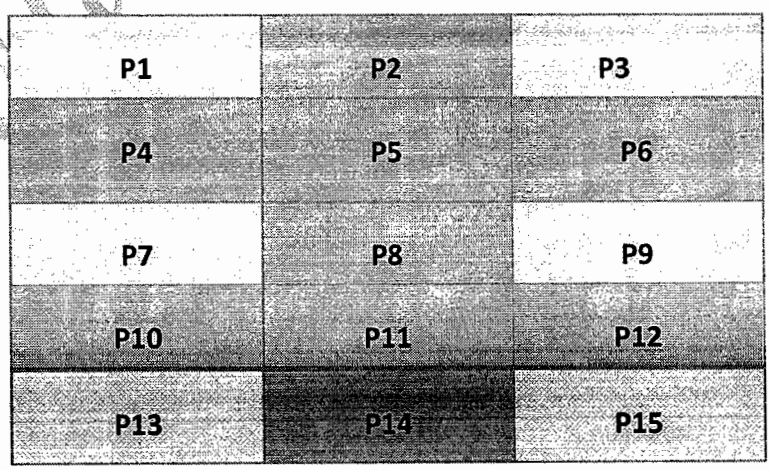
- We will connect to mainframe server using IP address
- To connect to the mainframes we should have emulation software in our system.
- Emulation Software is free software which we can download from internet.
- Different Emulation software's are VISTA, HUMMING BIRD, RUMBA, CITRIX, TELNET, etc,

Step1:

Allocate 2 partitions 'DATASET' namely
USERID.NAME.COBOL (to write/store program/JCLs)
USERID.NAME.LOADLIB (to store load module)

Partition Data Set:

- PDS is further divided into different partitions based on the directory blocks. Assume we are given DB as 15 the PDS will be divided into 15 partitions.



- In each partition we can write 6 members. The member can be COBOL programs (or) JCL or copy books or control cards.

Different Commands:

E → Edit (in edit mode we can make changes to the members and save them.
V → View (in view mode we can make changes but we can't save)
B → Browse (we can't make any changes)
I → It is used to insert a new line in the coding sheet.
D → It is used to delete a line.
DD → It is used to delete a block of lines.
R → Repeats a line.
RR → Repeat a group of lines.
RRN → Where 'n' is number of times. Ex: rr4.
M → Move a statement from one place to another place.
A → After
B → Before
C → Copy the statement.
CC → To copy a group of statements after or before the line.
A/B → To copy the statements from one member to another member we have to make use of 2 command line commands.
CUT → It will Copy all the Statements in the Program.
PASTE → It will Paste all the Statements in the Program.

Functional Case:

F3 → Exit / Back
F7 → Up (to go to page up)
F8 → Down (to go to next page)
F10 → Left Side of the screen.
F11 → Right Side of the screen.
F9 → It is used for swapping the screens.
We can open a New Screen by **start.*** (At a time we can open Maximum 8 Screens).
LIST+F9 → it is used to show / display all the screens we have opened.
COLS → Displaying the columns.
SAVE → It is used to make changes permanently.
SUB → It is used to submit the JOBs.
X → It is used to close the screen.

ISPF OPTIONS: (Interactive System Productivity Facility)

3.2 → Allocating the DATASET.
3.4 → List the DATASET.

Variable Declaration in COBOL:

Syntax:

LEVEL-NO VAR-NAME PIC DATATYPE&SIZE VALUE LITERAL.

Variable Name Rules:

1. The variable-name should not exceed more than 30 characters.
2. Only the allowable special char is hyphen.
3. Hyphen should not be present at the starting or else at the ending.

   E.g.: WS-A (correct declaration)
   WSA- (wrong declaration)
   -WSA (wrong declaration)
4. We should not give any reserved words.

   E.g.: Display, Division, Section etc...

PICTURE CLAUSE (PIC):

- PIC is used to identify the data type and data size.

DATATYPE:

- There are five different data types in COBOL Namely

Data Type	Cobol Equivalent	Max Data Size
Numeric (0-9)	9	18
Alphabetic (a-z)	A	160 → Cobol-85
Alpha Numeric (a-z)/(0-9)	X	120 → Cobol-74
Signed (+Ve, -Ve)	S	Combination with Numeric 9 → +Ve S9 → (+Ve, -Ve)
Assumed decimal point	V	9(3) v 9(2)

DATA SIZE:

- Data size is used to allocate the memory internally. Numeric, Alphanumeric and Alphabetic datatypes are going to take 1 byte of memory internally.

VALUE CLAUSE:

- This is used to assign values to the variable.

LITERAL:

- Value can be a literal with numeric data, alphabetic data and alphanumeric data.
- Numeric value should not be enclosed with in single codes.
- Alphabetic and alphanumeric should be mandatory enclosed with ' '.

E.g.: → NUMERIC

```
01    WS-A  PIC   9(3)    VALUE 800.
```

E.g.: → ALPHABETIC

```
01    WS-B  PIC   A (10)  VALUE 'SHASRA'.
```

E.g.: → ALPHANUMERIC

```
01    WS-C  PIC   X (6)   VALUE '00121'.
```

DEFAULT VALUES:

- For numeric the default values will be ZERO / ZEROS and for alphabetic, alphanumeric will be SPACE / SPACES.

E.g.:

```
01    WS-D  PIC   9(4)    VALUE ZERO /ZEROS.
01    WS-E  PIC   X(5)    VALUE SPACE/ SPACES.
```

P2. Write a program to ADD two no's and store the result in Third variable.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ADDPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01      WS-A  PIC     9(3)    VALUE 800.
01      WS-B  PIC     9(3)    VALUE 700.
01      WS-C  PIC     9(4)    VALUE ZEROS.
PROCEDURE DIVISION.
        COMPUTE     WS-C  =  WS-A  +  WS-B.
        DISPLAY 'WS-C:' WS-C.
        STOP RUN.
```

EXECUTION JCL FOR ADDPGM:

```
//FSS234EX    JOB    123,'SAHASRA',CLASS=A,NOTIFY=&USERID
//STEP1       EXEC   PGM=ADDPGM
//STEPLIB     DD     DSN=LOADLIB PDS,DISP=SHR
//SYSPRINT    DD     SYSOUT=*
//SYSOUT      DD     SYSOUT=*
//
```

DATA ITEMS (VARIABLES):

- In COBOL we have two types of data items (variables).
  1. Group Item.
  2. Elementary Data Item.

Note: A group item doesn't contain the PIC and should be declared only by 01 level number.

LEVEL NUMBERS:

- Level Numbers specify the level; of the data item in the program. We have Different Level Numbers, 01, 02...to...49, 66, 77 and 88

01	Group Data Item and elementary data item.
02 to **49**	Group Data Item or Sub-Group Item.
66	Rename (Grouping the different Groups).
77	Elementary Data Item.
88	Condition name Condition.

Group Data Item:

E.g.:

```
01      EMP-DET.
        05      EMP-ID          PIC     9(4).
        05      EMP-NAME        PIC     X(10).
        05      EMP-LOC.    → SUB GROUP
                10      CITY    PIC     X(10).
                10      STATE PIC       X(10).
        05      EMP-DOB         PIC     X(8).
```

- Level no's should be in increasing order.

- The difference between 01 and 77 is '01' is used for group item as well for elementary but '77' is used for only elementary but not for group item.

Elementary Data Item:

```
01      WS-A  PIC     9(4).
77      WS-B  PIC     X(5).
```

P3. Write a program to display a single employee details.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EMPDET.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 EMP-DET.
        05 EMP-ID PIC 9(5) VALUE 1111.
        05 EMP-NAME PIC X(10) VALUE 'SAHASRA'.
        05 EMP-LOC.
                10 CITY PIC X(10) VALUE 'HYD'.
                10 STATE PIC X(10) VALUE 'AP'.
         05 EMP-SAL PIC 9(5) VALUE 10000.
PROCEDURE DIVISION.
        DISPLAY EMP-DET.
        STOP RUN.
```

EXECUTION JCL:

```
//FSS234EX    JOB    123,'SAHASRA',CLASS=A,NOTIFY=&SYSUID
//STEP1       EXEC   PGM=EMPDET
//STEPLIB     DD     DSN=FSS234.SAHASRA.LOADLIB,DISP=SHR
//SYSPRINT    DD     SYSOUT=*
//SYSOUT      DD     SYSOUT=*
//
```

- Data Descriptor (DD) statements specify the input and output statements of our program.

Different ways of assigning values to the variable:

- Value Clause
- Move Verb
- Accept Verb

VALUE CLAUSE SYNTAX:

```
01 WS-A PIC 9(3) VALUE 800.
01 WS-B PIC X(10) VALUE 'Sahasra'. (It should be enclosed in ' ')
```

MOVE VERB SYNTAX:

- MOVE sending field value TO receiving field (or) MOVE Literal TO receiving field
  - Numeric without ' ' **E.g.**: 800.
  - Alpha Numeric with ' ' **E.g.**: 'abc'.

P4. Program to ADD TWO no's and store the result in third variable using move verb?

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ADDPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
      01    WS-A  PIC   9(3)   VALUE 800.
      01    WS-B  PIC   9(3)   VALUE ZEROS.
      01    WS-C  PIC   9(4)   VALUE ZEROS.
PROCEDURE DIVISION.
      MOVE WS-A TO WS-B.
      COMPUTE WS-C = WS-A + WS-B.
      DISPLAY WS-C.
      STOP RUN.
```

EXICUTION JCL:

```
//FSS234EX    JOB    123,'SAHASRA',CLASS=A,NOTIFY=&SYSUID
//STEP1       EXEC   PGM=ADDPGM
//STEPLIB     DD     DSN=FSS234.SAHASRA.LOADLIB,DISP=SHR
//SYSPRINT    DD     SYSOUT=*
//SYSOUT      DD     SYSOUT=*
//
```

P5. Write a program ASSIGNMENT

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MOVEPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-A PIC X(7) VALUE 'SAHASRA'.
01 WS-B PIC X(3) VALUE SPACES.
PROCEDURE DIVISION.
      MOVE WS-A TO WS-B.
       DISPLAY WS-A.
      DISPLAY WS-B.
STOP RUN.
```

Output: WS-A → SAHASRA
WS-B → SAH (For alpha numeric RHS truncation takes place.)

ACCEPT VERB:

- Value clause and move verb are used to assign the values Statically.
- Accept verb is used to assign the values to the variable Dynamically.
- Static → Assigning values within the program.
- Dynamic → Accepting values at the run time or at Execution time.

Note: It is always good to write dynamic program rather than static, because if we write the program dynamically the same copy of program used for multiple times.

Syntax: ACCEPT WS-VAR.

INSTREAM:

- Instream data is used for passing the values from JCL to COBOL.
- To accept the values in COBOL program we should have equivalent accept verbs.

SYNTAX:

```
//SYSIN DD * → starting of instream
  Value1
   ----      → (instream data)
  Value n
/*  → end of instream
```

P6. Write a program to ADD 2 numbers by using accept verb.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ADDPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-A PIC 9(3) VALUE ZEROS.
01 WS-B PIC 9(3) VALUE ZEROS.
01 WS-C PIC 9(4) VALUE ZEROS.
PROCEDURE DIVISION.
        ACCEPT WS-A.
        ACCEPT WS-B.
        COMPUTE WS-C = WS-A + WS-B.
        DISPLAY WS-C.
        STOP RUN.
```

EXICUTION JCL:

```
//FSS234EX   JOB    123,'SAHASRA',CLASS=A,NOTIFY=&SYSUID
//STEP1      EXEC   PGM=ADDPGM
//STEPLIB    DD     DSN=FSS234.SAHASRA.LOADLIB,DISP=SHR
//SYSPRINT   DD     SYSOUT=*
//SYSOUT     DD     SYSOUT=*
//SYSIN      DD     *
700
800
/*
//
```

Output: 1500

Note:

- In numeric data type the truncation happen from LHS.
- In non-numeric data type the truncation happens from RHS.
- We should take care when we pass values through instream data.

CONDITIONAL STATEMENTS IN COBOL:

- There are two conditional statements in COBOL
  1. IF → (IF, IF-ELSE, Nested-IF)
  2. Evaluate (In Cobol-85) →(EVALUATE TRUE, EVALUATE ALSO)

A. IF: Syntax: IF (Condition)
ST1
ST2 (There is no period in between IF and END-IF)
--
STn
END-IF.

B. IF-ELSE Syntax: IF (Condition)
ST1
ST2
ELSE
ST3

STn
END-IF.

C. Nested-IF Syntax: IF (Condition)
ST1
ST2
ELSE IF (Condition)
ST3
ST4
ELSE IF (Condition)
ST5
ST6
ELSE
ST7
ST8
END-IF
END-IF
END-IF.

P7. Write a program to find greater among two numbers.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CMPRPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-A PIC 9(3) VALUE ZEROS.
01 WS-B PIC 9(3) VALUE ZEROS.
PROCEDURE DIVISION.
    ACCEPT WS-A.
    ACCEPT WS-B.
    IF WS-A > WS-B
        DISPLAY 'WS-A IS GREATER'
    ELSE
        DISPLAY 'WS-B IS GREATER'
    END-IF.                    STOP RUN.
```

P8. Write a program to find greater among three numbers.

- LOGICAL OPERATORS: AND, OR, NOT

Note: Else is always a true condition hence else should be coded as the last condition.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BIG1PGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01      WS-A  PIC     9(3)    VALUE ZEROS.
01      WS-B  PIC     9(3)    VALUE ZEROS.
01      WS-C  PIC     9(4)    VALUE ZEROS.
PROCEDURE DIVISION.
        ACCEPT WS-A.
        ACCEPT WS-B.
        ACCEPT WS-C.
        IF WS-A > WS-B AND WS-A > WS-C
                DISPLAY 'WS-A IS GREATER'
        ELSE IF
        WS-B > WS-A AND WS-B > WS-C
                DISPLAY 'WS-B IS GREATER'
      ELSE
              DISPLAY 'WS-C IS GREATER'
        END-IF
        END-IF.
        STOP RUN.
```

P9. Assignment: Write a program to display the grades as follows.

80-100 → Distinction
60-79 → 1st class
50-59 → 2nd class
40-49 → 3rd class
00-39 → Fail

```
IDENTIFICATION DIVISION.
PROGRAM-ID. GRADEPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01      WS-M  PIC     9(3)    VALUE ZEROS.
PROCEDURE DIVISION.
        ACCEPT WS-M.
        IF WS-M >= 80 AND WS-M <= 100
                DISPLAY 'DISTINCTION'
        ELSE IF
        WS-M >= 60 AND WS-M <= 79
                DISPLAY '1ST CLASS'
```

```
ELSE IF
  WS-M >= 50 AND WS-M <= 59
        DISPLAY '2ND CLASS'
  ELSE IF
  WS-M >= 40 AND WS-M <= 49
        DISPLAY '3RD CLASS'
ELSE IF
  WS-M >= 00 AND WS-M <= 39
        DISPLAY 'FAIL'
  ELSE
        DISPLAY 'INVALID MARKS'
  END-IF
  END-IF
  END-IF
  END-IF
  END-IF.
  STOP RUN.
```

2. EVALUATE:

- Evaluate is also used for conditional checking similar to if statement.
- Evaluate was designed in Cobol-85.
- Evaluate efficiently and effectively replaces Nested-If statements.

Syntax: EVALUATE VARIABLE-NAME

```
        WHEN COND1
              STATEMENT1
              STATEMENT2
        WHEN COND2
              STATEMENT3
              STATEMENT4
        WHEN OTHER
              STATEMENT5
              STATEMENT6
    END-EVALUATE.
```

- Maximum we can check 255 when conditions using evaluate.
- WHEN OTHER is an optional statement but it is highly recommendable to code "WHEN OTHER" statement.
- WHEN OTHER is always a true statement, hence should be coded as last condition.
- When any of the condition is true the control will automatically pass to next statement after END-EVALUATE.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EVALPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01      WS-MONTH-NUM      PIC      9(2)      VALUE ZEROS.
```

```
PROCEDURE DIVISION.
        ACCEPT WS-MONTH-NUM.
        EVALUATE WS-MONTH-NUM
                WHEN        01
                        DISPLAY 'JAN'
                WHEN        02
                        DISPLAY 'FEB'
                        ----------
                        ----------
                WHEN        OTHER
                        DISPLAY 'ENTER VALID NUMBER'
        END-EVALUATE.
        STOP RUN.
```

Note: There is no period between EVALUATE and END-EVALUATE.

- "THRU" is a key word which is used to maintain a range of values in COBOL.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. THRUPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01      WS-SUB1     PIC     9(3)    VALUE ZEROS.
01      WS-SUB2     PIC     9(3)    VALUE ZEROS.
01      WS-SUB3     PIC     9(3)    VALUE ZEROS.
01      WS-SUM      PIC     9(3)    VALUE ZEROS.
01      WS-AVG      PIC     9(3) V 9(2)     VALUE ZEROS.
PROCEDURE DIVISION.
        ACCEPT      WS-SUB1.
        ACCEPT      WS-SUB2.
        ACCEPT      WS-SUB3.
        COMPUTE     WS-SUM=WS-SUB1+WS-SUB2+WS-SUB3.
        COMPUTE     WS-AVG=WS-SUM/3.
            DISPLAY 'WS-SUM:' WS-SUM.
            DISPLAY 'WS-AVG:' WS-AVG.
        EVALUATE    WS-AVG
            WHEN        80      THRU  100
                    DISPLAY 'DISTINCTION.'
            WHEN        60      THRU  79
                    DISPLAY 'FIRST DIVISION.'
            WHEN        50      THRU  59
                    DISPLAY 'SECOND DIVISION.'
            WHEN        40      THRU  49
                    DISPLAY 'THIRD DIVISION.'
            WHEN        00      THRU  39
                    DISPLAY 'FAIL.'
```

```
                WHEN OTHER
                DISPLAY 'ENTER VALID MARKS.'
            END-EVALUATE.
            STOP RUN.
```

EVALUATE TRUE.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. EVALPGM.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01     WS-GENDER          PIC    X(2)    VALUE ZEROS.
              88     MALE        VALUE 'M'.
              88     FEMALE      VALUE 'F'.
       PROCEDURE DIVISION.
              ACCEPT WS-GENDER.
              EVALUATE TRUE
                     WHEN        M
                            DISPLAY 'MALE'
                     WHEN        F
                            DISPLAY 'FEMALE'
                     WHEN        OTHER
                            DISPLAY 'ENTER VALID OPTION'
              END-EVALUATE.
              STOP RUN.
```

EVALUATE ALSO:

```
       IDENTIFICATION DIVISION.

       PROGRAM-ID. EALSOPGM.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
        01 SALARY PIC 9(9)  VALUE ZEROS.
        01 TAX     PIC 9(9)  VALUE ZEROS.
       01 SEX      PIC X     VALUE SPACES.
        PROCEDURE DIVISION.
        10000-MAIN-PARA.
              MOVE 10000 TO SALARY.
             MOVE 'M' TO SEX.
             EVALUATE TRUE ALSO TRUE
             WHEN SALARY <=10000 ALSO SEX='M'
                     DISPLAY 'MALE WORKER'
                      COMPUTE TAX= SALARY*20/100
                     DISPLAY 'TAX APPLICABLE IS =' TAX
            WHEN SALARY >10000   ALSO SEX='M'
                     DISPLAY 'MALE SUPERVISOR'
                     COMPUTE TAX= SALARY*30/100
```

```
            DISPLAY 'TAX APPLICABLE IS = TAX
WHEN SALARY <=10000 ALSO SEX='F'
            DISPLAY 'FEMALE WORKER'
            COMPUTE TAX= SALARY*05/100
            DISPLAY 'TAX APPLICABLE IS = ' TAX
WHEN SALARY >10000 ALSO SEX='F'
            DISPLAY 'FEMALE SUPERVISOR'
            COMPUTE TAX= SALARY*10/100
            DISPLAY 'TAX APPLICABLE IS = ' TAX
END-EVALUATE.
STOP RUN.
```

MEMORY SAVING TECHNIQUES IN COBOL:

1. REDEFINES
2. USAGE ITEMS
3. RENAMES (66 LEVEL NO)
4. CONDITION NAME CONDITION (88 LEVEL NO)

1. REDEFINES:

Redefines allow different variables to use the same computer storage area or memory space.

Syntax:

```
01 WS-A PIC X(5) VALUE AB123.
01 WS-B REDEFINES WS-A PIC 9(3).
```

Output: WS-A: AB123 & WS-B: AB1

Rules for Redefines:

1. Redefining variable and redefined variable should be of same level number.

```
01 WS-A PIC 9(5).
01 WS-B REDEFINES WS-A PIC 9(5).
```

2. Redefining variable should immediately follow the redefined variable.

```
01 WS-A PIC 9(5).
01 WS-B PIC 9(5).
01 WS-C REDEFINES WS-B PIC 9(5).
```

3. The PIC of redefining variable and redefined variable need not be same.

```
01 WS-A PIC 9(3).
01 WS-B REDEFINES WS-A PIC X(5).
```

Note: When we use redefines we can't use value clause.

P10. Write a program on REDEFINES.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. REDFIN1.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-A PIC 9(5) VALUES ZEROS.
```

```
01 WS-B REDEFINES WS-A PIC 9(5).
PROCEDURE DIVISION.
        ACCEPT WS-A.
        DISPLAY WS-A.
        DISPLAY WS-B.
        STOP RUN.
```

P11. PROGRAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID. REDFIN2.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-A PIC 9(5) VALUES ZEROS.
01 WS-B REDEFINES WS-A PIC 9(5).
PROCEDURE DIVISION.
        ACCEPT WS-A.
        DISPLAY WS-A.
        ACCEPT WS-B.
        DISPLAY WS-A.
        DISPLAY WS-B.
        STOP RUN.
```

P12. PROGRAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID. REDFIN3.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-A PIC 9(5) VALUES ZEROS.
01 WS-B REDEFINES WS-A PIC 9(3).
PROCEDURE DIVISION.
        ACCEPT WS-A.
        DISPLAY WS-A.
        DISPLAY WS-B.
        STOP RUN.
```

P13. PROGRAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID. REDFIN4.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-A PIC 9(5) VALUES ZEROS.
01 WS-B REDEFINES WS-A PIC 9(7).
PROCEDURE DIVISION.
        ACCEPT WS-A.
        ACCEPT WS-B.
```

DISPLAY WS-A.
DISPLAY WS-B.
STOP RUN.

P14. PROGRAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID. REDFIN5.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-A PIC 9(5) VALUES ZEROS.
01 WS-B REDEFINES WS-A PIC X(5).
PROCEDURE DIVISION.
    ACCEPT WS-A.
    DISPLAY WS-A.
    ACCEPT WS-B.
    DISPLAY WS-A.
    DISPLAY WS-B.
    STOP RUN.
```

2. USAGE ITEMS:

- There are five usage items in COBOL, Namely

Display → Default
comp → Binary Representation of Data.
comp-1 → Single precision Floating Point.
comp-2 → Double precision Floating Point.
comp-3 → Hexa Decimal Representation of Data.

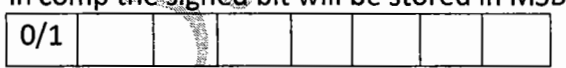
comp: It is a binary representation of data.

Syntax: 01 WS-A PIC S9(8) USAGE IS COMP.
(OR)
01 WS-A PIC S9(8) COMP.

Memory calculation in comp:

Lower Boundaries		Higher Boundaries		Memory Used
S9(01)	→	S9(04)	→	2 BYTES (HALF WORD).
S9(05)	→	S9(09)	→	4 BYTES (WORD).
S 9(10)	→	S9(18)	→	8 BYTES (DOUBLE WORD).

In comp the signed bit will be stored in MSB (Most Significant Bit).



Msb { 0 → +Ve Value, 1 → -Ve Value }

Comp-1 and Comp-2:

- Comp-1 and Comp-2 are used for storing the decimal point.
- In comp-1 we can store up to 0-22 decimal points.
- In comp-2 we can store up to 22-55 decimal points.

Note: For comp1 and comp-2 we should not mention any PIC clause. By default OS will be allocating 4 bytes of memory for comp-1 and 8 bytes for comp-2.

Syntax:
01 WS-A comp-1/comp-2.

comp-3:

- It is packed decimal or hexa decimal representation of data.

Syntax: 01 WS-A PIC S9(04) COMP-3.
(OR)
01 WS-A PIC S9(04) USAGE IS COMP-3.

Memory Calculation in comp-3:

If n (data size) is ODD → (n+1)/2 Bytes.
If n is EVEN → n/2 + 1 bytes.
Signed bit in comp-3.
Signed bit will be stored in last nibble (nibble = 4 bits).
C: Signed + ve.
D: Signed - ve.
F: Un-Signed + ve.

Memory representation for colmp-3 variable:

E.g.: if n is odd, 98765

9	7	5
8	6	c/d/f

E.g.: If n is even, 1234

0	2	4
1	3	c/d/f

Q) Which one is best either comp or comp-3?
Ans) It depends on the data size. If the data size is at lower boundaries comp-3 is best.
If data size is at higher boundaries com p is best.
Q) Where you can see values for comp and comp-3 data items?
Ans) File – aid tool.

3. RENAMES (66 LEVEL NUMBER):

- Renames is used for regrouping of different variables into single variable.

Syntax: 66 WS-VAR1 Renames <starting variable name> thru ending <variable name>.
E.g.: 66 WS-VAR1 Renames ws-var2 thru ws-var3.

- Renames is used generally for backup purpose. In renames we don't use any PIC clause.

P15. Write a program Pay Roll Of a company.

```
IDENTIFICATION  DIVISION.
PROGRAM-ID. REDFIN5.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01      PAY-ROLL-INFO.
```

```
03  EMP-PAY.
    05 EMP-BASIC PIC 9(5) VALUE 9000.
    05 EMP-HRA   PIC 9(4) VALUE 3000.
       10 EMP-PAY-OTHERS.
    05 EMP-DA    PIC 9(3) VALUE 800.
    05 EMP-INCENTIVES PIC 9(03) VALUE 500.
       10 EMP-DEDUCTIONS.
    05 EMP-PF-TX     PIC 9(03) VALUE 900.
    05 EMP-PROF-TX PIC 9(03) VALUE  200.
66 WS-VAR1 RENAMES EMP-BASIC THRU EMP-DA.
66 WS-VAR2 RENAMES EMP-PF-TX THRU EMP-PROF-TX.
PROCEDURE  DIVISION.
       DISPLAY EMP-PAY.
       DISPLAY WS-VAR1.
       DISPLAY WS-VAR2.
       STOP RUN.
```

- We should not renames 66, 77, 88 level numbers.

4. CONDITION NAME CONDITION (88 LEVELNUMBERS):

- When we have multiple conditions that need to be given to a single variable we go for condition name condition.

***Note*:** We can associate 77 Level Number in combination with 88 Level Number

P16. Write a program using CONDITION NAME CONDITION.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CONDPGM1.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01      MARITAL-STATUS PIC X.
        88      SINGLE VALUE        'S'.
        88      MARRIED VALUE       'M'.
        88      WIDOWED VALUE       'W'.
        88      DIVOREED VALUE      'D'.
01      WS-SAL          PIC 9(5)        VALUE 10000.
PROCEDURE DIVISION.
        ACCEPT   MARITAL-STATUS.
        IF SINGLE
                COMPUTE WS-SAL = WS-SAL + 1000
        ELSE IF MARRIED
                COMPUTE WS-SAL = WS-SAL + 2000
        ELSE IF WIDOWED
                COMPUTE WS-SAL = WS-SAL + 3000
        ELSE IF DIVORCED
                COMPUTE WS-SAL = WS-SAL + 4000
```

```
            ELSE
                DISPLAY 'ENTER CORRECT VALUE FOR MARITAL-STATUS'
            END-IF
            END-IF
            END-IF
            END-IF.
                DISPLAY MARITAL-STATUS.
                DISPLAY WS-SAL.
            STOP RUN.
```

SEQUENCE CONTROL VERBS:-

1. PERFORM
2. GO TO

PERFORM:

- Perform is an Iteration in COBOL program. We have different types of performs.
  1. Simple Perform
  2. Perform n Times
  3. Perform Until Condition.
  4. Perform Varying Condition.
  5. Perform Para-X thru Para-Y.
- This Performs can be coded as 2 types.
  1. Inline Perform.
  2. Outline Performs.
- Generally the program will be executed from top to bottom & left to right.
- If we want to break the control in between, we have to make use of outline perform.

Inline Perform

Syntax:

```
Perform (Condition)
        Statement—1
        Statement—2
        .
        .
        Statement—n
End-perform.
```

Outline Perform

Syntax:

```
        Perform Para-name
        Statement—3.
Para-name.
        Statement—1.
        Statement—2.
```

- End-perform is mandatory for inline perform
- No end performs for outline performs.

P17. Write a simple perform (Outline Perform) program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SIMPER1.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01    WS-A  PIC   9(3)   VALUE ZERO.
01    WS-B  PIC   9(3)   VALUE ZERO.
```

```
PROCEDURE DIVISION.
        PERFORM  ACCEPT-PARA.
        PERFORM  DISPLAY-PARA.
        STOP RUN.
ACCEPT- PARA.
        ACCEPT  WS-A.
        ACCEPT  WS-B.
DISPAY-PARA.
        DISPLAY WS-A.
        DISPLAY WS-B.
```

P18. Write a simple perform (Inline Perform) program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SIMPER1.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01      WS-A  PIC     9(3)    VALUE ZERO.
01      WS-B  PIC     9(3)    VALUE ZERO.
PROCEDURE DIVISION.
        PERFORM
                ACCEPT  WS-A
                ACCEPT  WS-B
                DISPLAY WS-A
                DISPLAY WS-B
        END-PERFORM.
        STOP RUN.
```

- Outline passes control to some other Para expects back the control after perform.

P19. Write a program to perform n times.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  NPGM.
ENVIRONMENT DIVISION.
DATA DISIVION.
WORKING-STROGE SECTION.
01      WS-A  PIC     9(3)    VALUE ZERO.
01      WS-B  PIC     9(3)    VALUE ZERO.
PROCEDURE DIVISION.
       PERFORM ACCEPT-PARA 5 TIMES.
       STOP RUN.
ACCEPT-PARA.
        ACCEPT WS-A.
        ACCEPT WS-B.
        PERFORM  DISPLAY-PARA.
DISPLAY-PARA.
        DISPLAY WS-A.
        DISPLAY  WS-B.
```

P20. Write a program to perform until condition.

- In perform until condition the Para will be executed when the condition is false and it comes out of Para when the condition is true.

```
IDENTIFICATON DIVISION.
PROGRAM-ID.  UNTLPGM.
ENIVIRONMENT DIVISION.
DATA  DIVISION.
WORKING-STORAGE SECTION.
01      WS-A  PIC     9(2)    VALUE ZERO.
01      WS-B  PIC     9(2)    VALUE ZERO.
01      K     PIC      9      VALUE ZERO.
PROEDURE DIVISION.
        MOVE 1 TO K.
        PERFORM ACCEPT-PARA UNTIL K > 5.
        STOP RUN.
ACCEPT-PARA.
        ACCEPT WS-A.
        ACCEPT WS-B.
        PERFORM DISPLAY-PARA.
        DISPLAY-PARA.
        DISPLAY WS-A.
        DISPLAY WS-B.
        COMPUTE K = K + 1.
```

Execution JCL:

```
// SYSIN DD *
1 2 3 4 5 6 7 8 9
/*
```

Output: 1 2 3 4 5 6 7 8 9 8

Perform Varying Condition:

Syntax: PERFORM PARA-NAME VARYING WS-VAR FROM STARTING - VALUE BY INCREMENTING VALUE UNTIL SOME CONDITION.

Note: For the very 1st time from clause will be Executed and 2nd time onwards by clause will be executed.

E.g.: PERFORM PARA-NAME VARYING K FROM 5 BY 10 UNTIL K > 50.
K = 5, K = 15, K = 25, K = 35, K = 45, K = 55

P21. Write a program using PERFORM VARYING CONDITION.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  VARYPGM.
ENIVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SETION.
01      WS-A  PIC     9(3)    VALUE ZERO.
01      WS-B  PIC     9(3)    VALUE ZERO.
```

```
01      K       PIC     9       VALUE ZERO.
PROCEDURE DIVISION.
        PERFORM ACCEPT-PARA VARYING K FROM 1 BY 1 UNTIL K > 5.
        STOP RUN.
ACCEPT-PARA.
        ACCEPT WS-A.
        ACCEPT WS-B.
        PERFORM DISPLAY-PARA.
DISPLAY-PARA.
        DISPLAY WS-A.
        DISPLAY WS-B.
```

P22. Write a program using PERFORM PARA-X THRU PARA-Y.

```
IDENTIFICATION DIVISION
PROGRAM-ID. THRUPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WOKING –STORAGE SECTION.
PROCEDURE DIVISION.
        PERFORM  PARA-5 THRU PARA-9.
        STOP RUN.
PARA-3.
        DISPLAY '3'.
PARA-5.
        DISPLAY '5'.
PARA-4.
        DISPLAY '4'.
PARA-10.
        DISPLAY '10'.
PARA-8.
        DISPLAY '8'.
PARA-9.
        DISPLAY '9'.
PARA-2.
        DISPLAY '2'.
```

Output: 5 4 10 8 9

PERFORM AND GO TO:

- Perform passes control to some other Para and expects back the control to the next statement after perform. Go to passes control to some other Para and doesn't expect back the control.

P23. Write a program using PERFORM AND GO TO.

E.g.:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  GOPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
        PERFORM PARA-5 THRU PARA-9.
        STOP RUN.
PARA-3.
        DISPLAY '3'.
PARA-5
      DISPLAY '5'.
      GO TO PARA-8.
PARA-6
        DISPLAY '6'.
PARA-7
        DISPLAY '7'.
PARA-8
        DISPLAY '8'.
PARA-9
        DISPLAY'9'.
```

Output: 5 8 9

OCCURS (ARRAYS IN 'C'):

- Occurs is table handling function in COBOL.
- When we want to repeat a group of data items or variable which are of same data type and data size.

Note: Occurs can't be coded at 01,66,77,88 level numbers.
To work on occurs we have to follow the following steps.

1. DECLARE THE OCCURS.

E.g.:
```
01   STU-REC.
     03 STU-DETAILS OCCURS 5 TIMES.
        05 STU-NO PIC 9(5).
        05 STU-NAME PIC X(10).
        05 STU-ADD   PIC X(10).
```

STU-NO(1)	STU-NAME(1)	STU-ADD(1)
STU-NO(2)	STU-NAME(2)	STU-ADD(2)
STU-NO(3)	STU-NAME(3)	STU-ADD(3)
STU-NO(4)	STU-NAME(4)	STU-ADD(4)
STU-NO(5)	STU-NAME(5)	STU-ADD(5)

STU-DETAILS(1) (25 Bytes)
STU-DETAILS(2) (25 Bytes)
STU-DETAILS(3) (25 Bytes)
STU-DETAILS(4) (25 Bytes)
STU-DETAILS(5) (25 Bytes)

2. LOAD VALUE INTO THE OCCURS/TABLE.

E.g.:
```
ACCEPT    STU-NO(K).
ACCEPT    STU-NAME(K).
ACCEPT    STU-ADDR(K).
```

3. RETRIEVE VALUES FROM THE TABLE.

E.g.:
```
DISPLAY  STU-NO(K).
DISPLAY  STU-NAME(K).
DISPLAY  STU-ADDR(K).
```

P24. Write a Program to load value into single dimensional table and retrieve all the values from the table.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. OCCPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01      STU-REC.
        03      STU-DETAILS   OCCURS        5       TIMES.
                05      STU-NO          PIC     9(3).
                05      STU-NAME        PIC     X(10).
                05      STU-ADDR        PIC     X(10).
01      K       PIC     9       VALUE ZERO.
PROCEDURE DIVISION.
        PERFORM  LOAD-PARA    VARYING K FROM1 BY 1 UNTIL K > 9.
        PERFORM  DISPLAY-PARA VARYING K FROM 1BY 1 UNTIL K > 5.
        STOP RUN.
LOAD-PARA.
        ACCEPT  STU-NO(K).
        ACCEPT  STU-NAME(K).
        ACCEPT  STU-ADDR(K).
DISPLAY-PARA.
    DISPLAY  STU-NO(K).
    DISPLAY  STU-NAME(K).
    DISPLAY  STU-ADDR(K).
```

- To display particular student details perform display-para.

```
DISPLAY-PARA.
    DISPLAY  STU-NO(3).
    DISPLAY  STU-NAME(3).
    DISPLAY  STU-ADDR(3).
             (OR)
    DISPLAY  STU-DETAILS(3).
```

P25. Write a program to load value into Two Dimensional tables and retrieve all the values from the table.

TWO-DIMENSIONAL OCCURS

```
IDENTIFICATION DIVISION.
PROGRAM-ID. D2PGM.
ENVIRONMENT DIVISION.
```

```
DATA   DIVISION.
WORKING –STORAGE SECTION.
01      STU-REC.
        03      STU-DETAILS   OCCURS        5 TIMES.
                05      STU-NO          PIC     9(3).
                05      STU-NAME        PIC     X(10).
                05      STU-ADDR        PIC     X(10).
                05      STU-MARKS       OCCURS          3 TIMES.
                    10          MARKS           PIC     9(3).
01      K       PIC     9.
01      J       PIC     9.
PROCEDURE DIVISION.
        PERFORM LOAD-PARA    VARYING K FROM 1 BY 1 UNTIL K > 5.
        PERFORM DISPLAY-PARA VARYING K FROM 1 BY 1 UNTIL K > 5.
        STOP RUN.
LOAD-PARA.
        ACCEPT          STU-NO(K).
        ACCEPT          STU-NAME(K).
        ACCEPT          STU-ADDR(K).
        PERFORM MARKS-PARA VARYING J FROM 1 BY 1 UNTIL J > 3.
MARKS-PARA.
      ACCEPT    MARKS(K,J).
DISPLAY-PARA.
        DISPLAY         STU-NO(K).
        DISPLAY         STU-NAME(K).
        DISPLAY         STU-ADDR(K).
        PERFORM DISPLAY-MARKS-PARA VARYING J FROM 1 BY 1 UNTIL J > 3.
DISPLAY-MARKS-PARA.
        DISPLAY MARKS(K, J).
```

SEARCH TECHNIQUES:

- Search is used or find on element from the table.
- There are 2types of search techniques.
  1. SEARCH
  2. SEARCH ALL

SEARCH:

- In Search the records will be checked in a linear or sequential fashion until a match is found or until end of the table (EOT).
- In search records need not be in the sorted order.

SEARCH ALL:

- In Search all it divides the table into 2 equals half and it checks for the element either in first half or second half based on element.
- It keeps on dividing the table into 2 equal half until a match is found or until table lookup is completed.

DIFFERENCE BETWEEN SEARCH AND SEARCH ALL

SEARCH	SEARCH ALL
Linear / Sequential Search.	Binary Search.
In search records need not be in sorted order.	Records should be in sorted order.
We have to set index variable to 1 before performing search.	No need to set index variable to 1, as the records are in sorted order.
Any logical operation can be performed. (>, <, >=, <=, NOT=)	Only = Operator is possible.
Multiple 'when' conditions can be coded.	Only single 'when' condition can be coded.
Search is preferable when table is small. We don't code asc-key OR dsc-key.	Preferable when table is large. We should code asc-key OR dsc-key.

DIFFERENCE BETWEEN SUBSCRIPT AND INDEX

SUBSCRIPT	INDEX
Occurrence: it internally converts to displacement and fetches the value.	Displacement: it holds the address of the value and directly fetches the value.
Slower.	Faster.
Subscript is working-storage variable.	Index is an internal item.
It should be declared in working-storage section.	No need to declare index variable.
It can be Incremented/Decremented with compute or perform. COMPUTE K=K+1 (OR) COMPUTE K=K-1 (OR) PERFORM VARYING.	To increment index we have to use SET verb. SET INDEX-VAR TO 1 OR SET INDEX-VAR UP BY 1/(K=K+1) OR SET INDEX-VAR DOWN BY 1/(K=K-1)

Search Syntax:

```
SET INDEX-VARIABLE TO 1.
SEARCH OCCURS DATA ITEM
AT END
        ST1
        ST2
WHEN WS-VARIABLE(INDEX)=SOME VALUE
        ST3
        ST4
END-SEARCH.
```

P26. Write a program on SEARCH.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SEARCHPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
01      STU-REC
        03      STU-DETAILS OCCURS 5 TIMES INDEXED BY A1.
                05      STU-ID          PIC     9(5).
                05      STU-NAME        PIC     X(10).
                05      STU-ADDR        PIC     X(10).
01      K               PIC     9  VALUE    ZERO.
01      WS-STU-ID       PIC     9(5)  VALUE   ZEROS.
PROCEDURE DIVISION.
        PERFORM LOAD-PARA VARYING K FROM 1 BY 1 UNTIL K > 5.
        PERFORM SEARCH-PARA.
        STOP RUN.
LOAD-PARA.
        ACCEPT STU-ID(K).
        ACCEPT STU-NAME(K).
        ACCEPT STU-ADDR(K).
SEARCH-PARA.
        ACCEPT WS-STU-ID.
        SET A1 TO 1.
        SEARCH STU-DETAILS
        AT END
                DISPLAY "RECORD NOT FOUND"
        WHEN STU-ID(A1)=WS-STU-ID
                DISPLAY STU-DETAILS(A1)
        END-SEARCH.
```

P27. Write a SEARCH ALL PROGRAM.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SEARCHPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 STU-REC.
        03 STU-DETAILS OCCURS 5 TIMES ASCENDING KEY IS STU-ID INDEXED BY A1.
                05      STU-ID          PIC     9(5).
                05      STU-NAME        PIC     X(10).
                05      STU-ADDR        PIC     X(10).
01      K               PIC     9  VALUE   ZERO.
01      WS-STU-ID       PIC     9(5)  VALUE   ZEROS.
PROCEDURE DIVISION.
        PERFORM LOAD-PARA VARYING K FROM 1 BY 1 UNTIL K > 5.
        PERFORM SEARCH-PARA.
        STOP RUN.
LOAD-PARA.
        ACCEPT STU-ID(K).
```

```
        ACCEPT STU-NAME(K).
        ACCEPT STU-ADDR(K).
SEARCH-PARA.
        ACCEPT WS-STU-ID.
        SEARCH ALL STU-DETAILS
        AT END
                DISPLAY "RECORD NOT FOUND"
         WHEN STU-ID(A1)=WS-STU-ID
                DISPLAY STU-DETAILS(A1)
        END-SEARCH.
```

Note: If we give duplicate STU-IDS with different names, then the 1st record will be displayed.

P28 Write a program to increment the index-value

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SEARCH1.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 STU-REC.
        05 STU-DETAILS OCCURS 5 TIMES INDEXED BY A1.
                03      STU-NO          PIC     9(3).
                03      STU-NAME        PIC     X(3).
                03      STU-ADDR        PIC     X(3).
01      K               PIC 9  VALUE  ZERO.
01      EOT             PIC X(3)  VALUE  'NO'.
01      WS-STU-ID       PIC  9(5)  VALUE   ZEROS.
PROCEDURE DIVISION.
        PERFORM LOAD-PARA VARYING K FROM 1 BY 1 UNTIL K > 5.
        SET A1 TO 1.
        PERFORM SEARCH-PARA UNTIL EOT = 'YES'. STOP RUN.
LOAD-PARA.
        ACCEPT          STU-NO(K).
        ACCEPT          STU-NAME(K).
        ACCEPT          STU-ADDR(K).
SEARCH-PARA.
        ACCEPT          WS-STU-ID.
        SEARCH          STU-DETAILS
        AT END
                MOVE 'YES' TO EOT
                DISPLAY 'RECORD NOT FOUND'
        WHEN STU-NO(A1) = WS-STU-ID
                DISPLAY STU-DETAILS(A1)
        SET A1 UP BY  1
        END-SEARCH.
```

FILES

- Generally any project deals with collection of Data. The data can be permanently stored in either file or database.
- Spool is a temporary dataset which stores job information for max 48 hours.
- There are 2 types of datasets
    1. Partition Datasets (PDS).
    2. Physical Sequential Dataset (PS) / FLAT FILE.

PDS:

- PDS is further divided into members based on the Directory Blocks.
- For PDS the Directory Blocks should be mandatory greater than '0'.

E.g.: Assume we have given Directory Blocks = 9 then PDS will be divided as follows.

P1	P2	P3
P4	P5	P6
P7	P8	P9

- Members can be a Cobol Program, JCL, Copy Books or Control Cards.

PS FILE:

- PS file is a collection of Records.
- Records are a collection of Fields.
- For PS files the Directory Blocks should be mandatorily zero.

E.g.:

CUST-ID	CUST-NAME	CUST-ADDR	CUST-AMT

Different operations that can be operated on a File:

1. Open
2. Write
3. Read
4. Update
5. Append
6. Close

Different Modes of File:

Mode	***File Operation***
OUTPUT	Write
INPUT	Read
INPUT-OUTPUT	Update
EXTEND	Append

Steps to be followed to work on File Program:

1. Define the file.
2. Declare the file.
3. Open the file.
4. Process the file.
5. Close the file.

1) **Define:**
   ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
   FILE-CONTROL.
2) **Declare:**
   DATA DIVISION.
   FILE SECTION.
   FD FileName. (**FD** → FILE DESCRIPTION)
3) **Open:**
   OPEN MODENAME FILENAME.
4) **Process:**
   Varies from Program to Program
5) **Close:** CLOSE FILENAME.

Define Syntax:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT FILENAME ASSIGN TO DDNAME
        ORGANIZATION IS SEQUENTIAL / INDEXED / RELATIVE
        ACCESS MODE IS SEQUENTIAL / RANDOM / DYNAMIC.
```

Declare Syntax:

```
DATA DIVISION.
FILE SECTION.
FD FILENAME
RECORD MODE IS FIXED / VARIABLE
BLOCK CONTAINS 32 CHARACTERS
LABEL RECORDS ARE STANDARD / OMITTED.
01 CUST-REC.
        05    CUST-ID      PIC    9(5).
        05    CUST-NAME    PIC    X(10).
        05    CUST-ADDR    PIC    X(10).
        05    CUST-AMT     PIC    9(5)V99.
```

The records will be internally stored in blocks. Block's helps us to increase the speed or access.

P29. Write a program to write records into a File.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. WRIFIL.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT CUSTFILE ASSIGN TO CUSTDD
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD CUSTFILE.
```

```
01 CUST-REC.
        05    CUST-ID      PIC   9(5).
        05    CUST-NAME    PIC   X(10).
        05    CUST-ADDR    PIC   X(10).
        05    CUST-AMT     PIC   9(5)V9(2).
WORKING-STORAGE SECTION.
01  WS-EOF       PIC    X(3)    VALUE 'NO'.
PROCEDURE DIVISION.
A001-MAIN-PARA.
        PERFORM     B001-OPEN-PARA.
        PERFORM     B002-PROCESS-PARA  UNTIL  WS-EOF ='YES'.
        PERFORM     C001-CLOSE-PARA.
        STOP RUN.
B001-OPEN-PARA.
        OPEN OUTPUT CUSTFILE.
B002-PROCESS-PARA.
        ACCEPT CUST-ID.
        ACCEPT CUST-NAME.
        ACCEPT CUST-ADDR.
        ACCEPT CUST-AMT.
        WRITE CUST-REC.
        ACCEPT WS-EOF.
C001-CLOSE-PARA.
        CLOSE CUSTFILE.
```

EXECUTION JCL FOR WRIFILE PROGRAME

```
// FSS244EX   JOB     123,'SAHASRA',CLASS=A,NOTIFY=&SYSUID
//STEP1       EXEC    PGM=WRIFIL
//STEPLIB     DD      DSN=FSS244.SAHASRA.LOADLIB,DISP=SHR
//CUSTDD      DD      DSN=FSS244.SAHASRA.CUSTFILE,
//                    DISP=(NEW,CATLG,DELETE),
//                    SPACE=(TRK,(1,1),RLSE),
//                    UNIT=SYSDA,
//                    DCB=(DSORG=PS,LRECL=32,BLKSIZE=320,RECFM=FB)
//SYSPRINT    DD      SYSOUT=*
//SYSOUT      DD      SYSOUT=*
//SYSIN       DD      *
******INPUT VALUES******
/*
//
```

Note: We can add the records only at the end of PS files.

- In previous program for extending file we need to change in the program

```
B001-OPEN-PARA.
        OPEN     EXTEND      CUSTFILE.
```

- IN JCL

```
DISP=(MOD,CATLG,DELETE)
```

Read Syntax:

```
READ FILENAME
AT END
        STATEMENT—1
        STATEMENT—2
NOT AT END
        STATEMENT—3
        STATEMENT—4
              :     :
        STATEMENT—N
END-READ.
```

Note: There should be no periods between read and end-read

P30. Write a program to display all the records from the file and print them into spool.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. READFIL.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT CUSTFILE ASSIGN TO CUSTDD
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD CUSTFILE.
01 CUST-REC.
        05    CUST-ID      PIC    9(5).
        05    CUST-NAME    PIC    X(10).
        05    CUST-ADDR    PIC    X(10).
        05    CUST-AMT     PIC    9(5)V9(2).
WORKING-STORAGE SECTION.
01  WS-EOF     PIC    X(3)    VALUE 'NO'.
PROCEDURE DIVISION.
A001-MAIN-PARA.
        PERFORM     B001-OPEN-PARA.
        PERFORM     B002-PROCESS-PARA  UNTIL   WS-EOF ='YES'.
        PERFORM     C001-CLOSE-PARA.
        STOP RUN.
B001-OPEN-PARA.
        OPEN INPUT CUSTFILE.
B002-PROCESS-PARA.
        READ CUSTFILE
        AT END
              MOVE 'YES' TO WS-EOF
        NOT AT END
              DISPLAY CUST-REC
```

```
            END-READ.
        C001-CLOSE-PARA.
            CLOSE CUSTFILE.
```

***Note*:** By default organization and access mode will take value as sequential.

EXECUTION JCL:

```
//JOB CARD
//STEP1 EXEC PGM=READFIL
//STEPLIB DD DSN=LOAD-PDS,DISP=SHR
//CUSTDD DD DSN=FSS244.SAHASRA.CUSTFILE,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
/*
//
```

P31. Write a Program to read all the records from the file and then write into another File.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. READFIL1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT CUSTFILI  ASSIGN TO CUSTDDI.
        SELECT CUSTFILO ASSIGN TO CUSTDDO.
DATA DIVISION.
FILE SECTION.
FD CUSTFILI.
01      CUST-REC.
        05      CUST-ID         PIC     9(5).
        05      CUST-NAME       PIC     X(10).
        05      CUST-ADDR       PIC     X(10).
        05      CUST-AMT        PIC     9(5)V9(2).
FD CUSTFILO.
01      CUST-REC-OUT.
        05      CUST-ID-OUT             PIC     9(5).
        05      CUST-NAME-OUT           PIC     X(10).
        05      CUST-ADDR-OUT           PIC     X(10).
        05      CUST-AMT-OUT            PIC     9(5)V9(2).
WORKING-STORAGE SECTION.
01      WS-EOF          PIC     X(3)    VALUE 'NO'.
PROCEDURE DIVISION.
A100-MAIN-PARA.
        PERFORM     B100-OPEN-PARA.
        PERFORM     B200-PROCESS-PARA  UNTIL WS-EOF = 'YES'.
        PERFORM     C100-CLOSE-PARA.
        STOP RUN.
```

```
B100-OPEN-PARA.
        OPEN          INPUT        CUSTFILI.
        OPEN          OUTPUT       CUSTFILO.
B200-PROCESS-PARA.
        READ CUSTFILI
        AT END
              MOVE 'YES' TO WS-EOF
        NOT AT END
              MOVE CUST-REC TO CUST-REC-OUT
        WRITE CUST-REC-OUT
        END-READ.
C100-CLOSE-PARA.
        CLOSE  CUSTFILI.
        CLOSE  CUSTFILO.
```

EXECUTION JCL:

```
//JOB CARD
//STEP1       EXEC   PGM=READFIL1
//STEPLIB     DD     DSN=LOADLIB-PDS,DISP=SHR
//CUSTDDI     DD     DSN=I/P PS-FILE,DISP=SHR
//CUSTDDO     DD     DSN=O/P PS-FILE,
//                   DISP=(NEW,CATLG,DELETE),
//                   SPACE=(TRK,(1,1),RLSE),
//                   DCB=(DSORG=PS,LRECL=32,BLKSIZE=320,RECFM=FB)
//SYSPRINT    DD     SYSOUT=*
// SYSOUT     DD     SYSOUT=*
//
```

P32. Write a program to get list of Customers who's AMT is greater than 50000.

```
B200-PROCESS-PARA.
        READ CUSTFILI
        AT END
              MOVE 'YES' TO WS-EOF
        NOT AT END
              IF CUST-AMT > 50000
              MOVE CUST-REC TO CUST-REC-OUT
              WRITE CUST-REC-OUT
              END-IF
        END-READ.
```

P33. Write a program to update the record in a file.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UPDTFILE.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT CUSTFILE ASSIGN TO CUSTDD.
DATA DIVISION.
```

```
FILE SECTION.
01      CUST-REC.
        05      CUST-ID         PIC     9(5).
        05      CUST-NAME       PIC     X(10).
        05      CUST-ADDR       PIC     X(10).
        05      CUST-AMT        PIC     9(5)V9(2).
WORKING-STORAGE SECTION.
01      WS-EOF                  PIC     9(3)    VALUE 'NO'.
01      WS-CUST-ID              PIC     9(5)    VALUE ZEROS.
01      WS-CUST-ADDR            PIC     X(10)   VALUE SPACES.
PROCEDURE DIVISION.
A101-MAIN-PARA.
        PERFORM     B101-OPEN-PARA.
        PERFORM     B201-UPDATE-PARA   UNTIL WS-EOF = 'YES'.
        PERFORM     C301-CLOSE-PARA.
        STOP RUN.
B101-OPEN-PARA.
                OPEN   INPUT  CUSTFILE.
B201-UPDATE-PARA.
                ACCEPT        WS-CUST-ID.
                ACCEPT        WS-CUST-ADDR.
                READ   CUSTFILE
                AT END
                        MOVE  'YES'   TO      WS-EOF
                        DISPLAY       'RECORD NOT FOUND'
                NOT AT END
                        IF CUST-ID = WS-CUST-ID
                        MOVE WS-CUST-ADDR TO CUST-ADDR
                        REWRITE CUST-REC
                        MOVE 'YES' TO WS-EOF
                END-IF
                END-READ.
C301-CLOSE-PARA.
                CLOSE  CUSTFILE.
```

EXECUTION JCL

```
//JOBCARDS
//STEP1         EXEC    PGM=UPDTFILE
//STEPLIB       DD      DSN=LOADLIB-PDS,DISP=SHR
//CUSTDD        DD      DSN=I/P-PS FILE,DISP=OLD
//SYSPRINT      DD      SYSOUT=*
//SYSOUT        DD      SYSOUT=*
//SYSIN         DD      *
12345
BANGLORE
/*      //
```

P34. Write a program to copy only the even records to the output file

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COPYEVN.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT CUSTFILI   ASSIGN TO CUSTDDI.
        SELECT CUSTFILO ASSIGN TO CUSTDD1.
DATA DIVISION.
FILE SECTION.
01 CUST-REC.
        05 CUST-ID              PIC  9(5).
        05 CUST-NAME            PIC  X(10).
        05 CUST-ADDR            PIC  X(10).
        05 CUST-AMT             PIC  9(5)V9(2).
01 CUST-REC-OUT.
        05 CUST-ID1             PIC  9(5).
        05 CUST-NAME1           PIC  X(10).
        05 CUST-ADDR1           PIC  X(10).
        05 CUST-AMT1            PIC  9(5)V9(2).
WORKING-STORAGE SECTION.
01 WS-EOF      PIC 9(3) VALUE 'NO'.
01 COUNT       PIC 9(2) VALUE ZEROS.
PROCEDURE DIVISION.
A100-MAIN-PARA.
        PERFORM       B100-OPEN-PARA.
        PERFORM       B200-PROCESS-PARA  UNTIL WS-EOF = 'YES'.
        PERFORM       C100-CLOSE-PARA.
        STOP RUN.
B100-OPEN-PARA.
        OPEN   INPUT  CUSTFILI.
        OPEN   OUTPUT CUSTFILO.
B200-PROCESS-PARA.
        READ   CUSTFILI
        AT END
                MOVE 'YES' TO WS-EOF
                NOT AT END
                COMPUTE  COUNT = COUNT  + 1
        IF COUNT = = 2
                MOVE CUST-REC TO CUST-REC-OUT
                WRITE CUST-REC-OUT
                MOVE 0 TO COUNT
        END-IF
        END-READ.
C100-CLOSE-PARA.
```

CLOSE CUSTFILI.
CLOSE CUSTFILO.

FILLER:

- Filler is a reserved word. They are used to separate the fields with spaces.
- Fillers are coded in data division and the data type of filler is alphanumeric.

COPYBOOK:

- When we want to repeat group of statements multiple times it is better to place that statements in COPYLIB member and call the member by following syntax.

Syntax: COPY COPYBOOKNAME

- Copybooks can be used either in data division or procedure division.
- Copybooks are expanded during compilation time.
- We can write copybook with in a copybook.

Note: Generally we code the filler with high value. This filler helps us in adding new fields.

ADVANTAGE OF COPYBOOK:

1. REUSABILITY.
2. REDUCES THE MEMORY SPACE.
3. REDUCES THE MANUAL EFFORT.

FILE- STATUS:

- File-status is used for error handling for files.
- File-status is predefined register; it gets some value automatically when we perform any operation on the file.

IF FILE STATUS = 00 → SUCCESSFUL OPERATION
FILE STATUS = 10 → END OF FILE

- Other than these values file operation is unsuccessful.

E.g.: 22, 23, 39, 41, 97 etc.

- We define the file status in file-control paragraph.
- It is always a good practice to check for file status code after every file operation (open, write, update etc).

P35. Write a program FOR FILE STATUS.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COPYEVN.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT CUSTFILI ASSIGN TO CUSTDD
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUETIAL
        FILE STATUS IS WS-STAT.
DATA DIVISION.
FILE SECTION.
FD CUSTFILE.
COPY MEM1.
```

```
WORKING-STORAGE SECTION.
01      WS-EOF          PIC     X(3)    VALUE 'NO'.
01      WS-STAT         PIC     X(2).
PROCEDURE DIVISION.
A100-MAIN-PARA.
        PERFORM     100-OPEN-PARA       THRU  100-OPEN-EXIT.
        PERFORM     200-READ-PARA       THRU  200-READ-EXIT.
        PERFORM     300-CLOSE-PARA      THRU  300-CLOSE-EXIT.
        STOP RUN.
100-OPEN-PARA.
        OPEN   INPUT CUSTFILI.
        IF WS-STAT = 00
                DISPLAY 'SUCCUSSFULLY OPEN'
        ELSE
                DISPLAY 'FILE STATUS' WS-STAT
        END-IF.
100-OPEN-EXIT.
            EXIT.
200-READ-PARA.
         ---------
200-READ-EXIT.
            EXIT.
300-CLOSE-PARA.
        CLOSE CUSTFILE.
        IF WS-STAT = 00
                DISPLAY 'CLOSE SUCCUSFUL'
        ELSE
                DISPLAY 'ERROR DURING CLOSE' WS-STAT
        END-IF.
300-CLOSE-EXIT.
        EXIT.
```

P36. WRITE A PROGRAM TO READ RECORDS FROM THE FILE AND LOAD INTO TABLE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PGM1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT CUSTFILE ASSIGN TO CUSTDD
        ORGANIZATION IS SEQUETIAL
        ACCESS MODE IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD CUSTFILE.
01 CUST-REC.
        05      CUST-ID         PIC     9(5).
```

```
        05      CUST-NAME   PIC     X(10).
        05      CUST-ADDR   PIC     X(10).
        05      CUST-AMT    PIC     9(5)V9(2).
WORKING-STORAGE SECTION.
01      CUST-REC-OUT.
        02      CUST-DETAILS        OCCURS      3 TIMES.
                05      CUST-ID1    PIC     9(5).
                05      CUST-NAME1  PIC     X(10).
                05      CUST-ADDR1  PIC     X(10).
                05      CUST-AMT1   PIC     9(5)V9(2).
01      WS-EOF      PIC     X(3)    VALUE 'NO'.
01      K           PIC     9(2)    VALUE ZEROS.
PROCEDURE DIVISION.
        PERFORM     OPEN-PARA.
        PERFORM     READ-PARA   UNTIL WS-EOF = 'YES'.
        PERFORM     CLOSE-PARA.
        STOP RUN.
OPEN-PARA.
        OPEN    INPUT  CUSTFILE.
READ-PARA.
        READ    CUSTFILE
        AT END
                MOVE 'YES' TO WS-EOF
                NOT AT END
                PERFORM     PARA1 VARYING K FROM 1 BY 1 UNTIL K > 3
                PERFORM     DISPLAY-PARA VARYING K FROM 1 BY 1 UNTIL K > 3
        END-READ.
PARA1.
        MOVE CUST-REC   TO CUST-DETAILS(K).
DISPLAY-PARA.
        DISPLAY CUST-DETAILS(K).
CLOSE-PARA.
        CLOSE CUSTFILE.
```

P37. Write a MATCHING LOGIC PROGRAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MATPGM.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT INFILE1   ASSIGN TO DD1.
        SELECT INFILE2   ASSIGN TO DD2.
        SELECT OUTFILE ASSIGN TO DD3.
DATA DIVISION.
FILE SECTION.
FD INFILE1.
```

```
01 CUST-REC1.
        05      CUST-ID         PIC     9(5).
        05      CUST-NAME       PIC     X(10).
        05      CUST-ADDR       PIC     X(10).
        05      CUST-AMT        PIC     9(5)V9(2).
FD INFILE2.
01 CUST-REC2.
        05      CUST-ID1        PIC     9(5).
        05      CUST-NAME1      PIC     X(10).
        05      CUST-ADDR1      PIC     X(10).
        05      CUST-AMT1       PIC     9(5)V9(2).
FD OUTFILE.
01      OUT-REC         PIC     X(32).
WORKING-STORAGE SECTION.
01      WS-EOF          PIC X(3)        VALUE 'NO'.
01      WS-EOF1         PIC X(3)        VALUE 'NO'.
PROCEDURE DIVISION.
        PERFORM         OPEN-PARA.
        PERFORM         READ-PARA1.
        PERFORM         READ-PARA2.
        PERFORM         PROCESS-PARA UNTIL WS-EOF = 'YES' OR WS-EOF1 = 'YES'.
        PERFORM         CLOSE-PARA.
        STOP RUN.
OPEN-PARA.
        OPEN    INPUT           INFILE1.
        OPEN    INPUT           INFILE2.
        OPEN    OUTPUT          OUTFILE.
READ-PARA1.
        READ    INFILE1
        AT END
                MOVE 'YES' TO WS-EOF
        END-READ.
READ-PARA2.
        READ INFILE2
        AT END
                MOVE 'YES' TO WS-EOF1
        END-READ.
PROCESS-PARA.
        IF CUST-ID = CUST-ID1
                MOVE CUST-REC1 TO OUT-REC
                WRITE OUT-REC
                PERFORM READ-PARA1
                PERFORM READ-PARA2
        ELSE IF
                CUST-ID > CUST-ID1
```

```
                PERFORM READ-PARA2
            ELSE IF
                CUST-ID < CUST-ID1
                PERFORM READ-PARA1
            END-IF
            END-IF
            END-IF.
        CLOSE-PARA.
            CLOSE  INFILE1.
            CLOSE  INFILE2.
            CLOSE  OUTFILE.
```

CHARACTER HANDLING FUNCTIONS/STRINGS:

- There are three character handling functions in COBOL, namely

1. INSPECT:

- Find the count of occurrence of character in a string.
- Replace one char with other.

2. STRING:

- Used to concatenate 2 or more strings into single string.

3. UNSTRING:

- Used to split single string into multiple strings.

P38. Write a program on 'INSPECT COUNT'.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. INSPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01      WS-NAME     PIC     X(30)   VALUE 'SAHASRA INFOTECH'.
01      WS-COUNT    PIC     9       VALUE  ZEROS.
PROCEDURE DIVISION.
*********************Counting how many A's in Sting**************************
        INSPECT     WS-NAME     TALLYING     WS-COUNT    FOR    'A'.
        DISPLAY WS-COUNT.
        STOP RUN.
*************Counting how many A's  before a character F in a String************
        INSPECT     WS-NAME     TALLYING     WS-COUNT
                    FOR    ALL    'A'     BEFORE      'F'.
        DISPLAY WS-COUNT.
        STOP RUN.
***************Counting how many 'A's' after a character F in a String*************
        INSPECT     WS-NAME     TALLYING     WS-COUNT
                    FOR    ALL    'A'     AFTER   'F'.
        DISPLAY WS-COUNT.
        STOP RUN.
```

**********Counting how many 'A's' after 'F' and before 'T' in a string**********

```
INSPECT     WS-NAME     TALLYING     WS-COUNT
                FOR  ALL  'A'  AFTER  'F'  BEFORE  'T'.
DISPLAY WS-COUNT.
STOP RUN.
```

P39. Write a program on 'INSPECT REPLACE'.

```
WORKING-STORAGE SECTION.
01    WS-NAME    PIC    X(30)  VALUE  'SAHASRA INFOTECH'.
PROCEDURE DIVISION.
      INSPECT     WS-NAME     REPLACING  ALL  'A'  BY  '$'.
      DISPLAY WS-NAME.
      STOP RUN.
```

P40. Write a program on 'STRING'.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. STRPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01    WS-FNAME   PIC    X(10)  VALUE 'SAHASRA'.
01    WS-MNAME   PIC    X(10)  VALUE 'INFOTECH'.
01    WS-LNAME   PIC    X(20)  VALUE 'AND CONSULTING'.
01    WS-NAME    PIC    X(50)  VALUE SPACES.
PROCEDURE DIVISION.
      STRING      WS-FNAME,WS-MNAME,WS-LNAME
                  DELIMITED  BY  SIZE  INTO  WS-NAME.
      DISPLAY WS-NAME.
      STOP RUN.
```

P41. Write a program on 'UNSTRING'.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UNSTRPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01    WS-NAME    PIC    X(30)  VALUE  'SAHASRA INFOTECH AND CONSULTING'.
01    WS-FNAME   PIC    X(10)  VALUE SPACES.
01    WS-MNAME   PIC    X(10)  VALUE SPACES.
01    WS-LNAME   PIC    X(10)  VALUE SPACES.
PROCEDURE DIVISION.
      UNSTRING    WS-NAME
            DELIMITED BY SPACES INTO WS-FNAME, WS-MNAME, WS-LNAME.
      DISPLAY WS-FNAME.
      DISPLAY WS-MNAME.
      DISPLAY WS-LNAME.
      STOP RUN.
```

Different Types of Moves

1. SIMPLE MOVE
2. GROUP MOVE
3. CORRESPONDING MOVE
4. REFERENCE MODIFICATION MOVE

SIMPLE MOVE

MOVE WS-A TO WS-D

GROUP MOVE

MOVE CUST-REC TO CUST-REC-OUT. (OR) MOVE WS-GROUP1 TO WS-GROUP2.

CORRESPONDING MOVE

MOVE CORRESPONDING WS-GRP1 TO WS-GRP2.

***Note*:**

- When we are moving data from one group to another group it will be considered as alphanumeric irrespective of elementary data type and data items.
- In corresponding move it checks for the data name and if the data names are same only that data move from one group to another group.

REFERENCE MODIFICATION

- When we want to move a part of text from one string to another string we make use of reference modification move. With this we can point to a particular char in a string

Syntax:

WS-VAR1(STARTING POSITION:LENGTH)

E.g.:

```
01    S-A    PIC    X(7)    VALUE 'SAHASRA'.
      MOVE   WS-A(3:3)      TO    WS-B.
      MOVE   WS-A(1:5)      TO    WS-B
```

SUBPROGRAMS

- Whenever there is necessity to write very lengthy program to meet different functionalities, it is better to divide the program into different modules or sub programs and then call in the main program. The program which passes the control is called as **MAIN PROGRAM.**
- The program which receives the control is called subprogram.

CALL Syntax:

CALL SUBPROGNAME USING WORKING-STORAGE VARAIBLES

E.g.: CALL SUBPGM1 USING WS-A,WS-B,WS-C BY REFERENCE / CONTENT.

Note: CALL BY REF IS A DEFAULT OPTION.

WRITE PRECONDITION TO WORK ON SUBPROG:

- Declare the actual parameters in Working Storage Section.
- Declare the formal parameters in the linkage section of subprogram.
- The number of variable in working storage section should be equal to number of variable in linkage section.
- The variable names can be different.
- The variable data type and data size should be same.

ACTUAL PARAMETERS (AP)

- The variables coded in main program are called Actual Parameters.

FORMAL PARAMETERS (FP)

- The variables coded in sub program are called as Formal Parameter.

CALL BY REFERENCE

- In call by ref when FP's are changed the AP's are also changed.

CALL BY CONTENT

- When the FP are changed the AP will not be changed because in this AP and FP will be using different memory where as in call by reference both AP and FP will be using same memory, Mostly we will use call by reference.

DIFFERENCE BETWEEN STOPRUN, EXIT PROGRAM, GOBACK

STOP RUN

- When control comes across STOP RUN it passes the control to OS for further processing. It is coded only in main programs.

EXIT PROGRAM

- When control comes across EXIT PROGRAM it passes the control to Main program for further processing. It is coded only in subprograms.

GO BACK

- It can be coded both in main program and subprogram. When coded in main program it acts similar to STOP RUN i.e., it passes control to OS for further processing. When coded in subprogram it acts similar to EXIT PROGRAM i.e., it passes control to main program for further processing.

P42. Write a program on 'STATIC CALL'

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINPGM1.
ENVIRONMENT DIVISION.
DATA DIVISION.
01 WS-A PIC 9(3) VALUE ZEROS.
01 WS-B PIC 9(3) VALUE ZEROS.
01 WS-C PIC 9(4) VALUE ZEROS.
PROCEDURE DIVISION.
        ACCEPT WS-A.
        ACCEPT WS-B.
        CALL 'SUBPGM1' USING WS-A,WS-B,WS-C.
        DISPLAY WS-C.
        STOP RUN. / GO BACK.
```

SP42 SUBPROGRAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUMPGM1.
ENVIRONMENT DIVISION.
DATA DIVISION.
LINKAGE SECTION.
01 LS-A PIC 9(3) VALUE ZEROS.
01 LS-B PIC 9(3) VALUE ZEROS.
01 LS-C PIC 9(4) VALUE ZEROS.
PROCEDURE DIVISION USING LS-A, LS-B, LS-C.
        COMPUTE LS-C = LS-A + LS-B.
        EXIT PROGRAM. / GO BACK.
```

STEPS TO WORK ON SUB PROGRAM:

- Write the main program and sub program.
- Compile the sub program.
- Then compile the main program. (In main program compilation the sub program and main program object module will be link edited to get a single load module in static call)
- In compile JCL of main program we have to mention as follows.
  //STEP1 EXEC IGYWCL, PARM = NODYNAM
- Write the execution JCL and in that give the main program load module in STEPLIB.

We Can Call The Program In Two Ways:

1. Static Call.
2. Dynamic Call.

- In Static call we directly call the Sub Program Name.
- In Dynamic Call we move Sub Program Name to working storage variable and then call working storage variable.

DIFFERENCE BETWEEN STATIC CALL AND DYNAMIC CALL

STATIC CALL / CALL PROGRAM	DYNAMIC CALL / MOVE PROGRAM
Call subprogram	Move subprogram to ws-var. Call ws-var
In Static call, all the main programs object module and sub programs object module will be link edited to get single load module.	In Dynamic call, it picks up the subprogram load module during run/execution time.
Fast	Slow
If we make any changes in subprogram, Then we need to recompile both subprogram and main program.	Recompilation of subprogram is enough.
Parm = nodynam.	Parm = dynam.
Size of load module is large.	Size of load module is small.
STOP RUN. / GO BACK.	EXIT PROGRAM. / GO BACK.

P43. Write a program on 'DYNAMIC CALL'.

MAIN PROGRAM:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MAINPGM1.
ENVIRONMENT DIVISION.
DATA DIVISION.
01 WS-A PIC 9(3) VALUE ZEROS.
01 WS-B PIC 9(3) VALUE ZEROS.
01 WS-C PIC 9(4) VALUE ZEROS.
01 WS-SUBNAME PIC X(8) VALUE SPACE.
PROCEDURE DIVISION.
        ACCEPT WS-A.
        ACCEPT WS-B.
        MOVE SUBPGM1 TO WS-SUBNAME.
        CALL WS- SUBNAME USING WS-A,WS-B,WS-C.
        DISPLAY WS-C.
        STOP RUN.
```

There are two types of scope terminators,

1) **Implicit Scope terminator**
   E.g.: Period (.)
2) **Explicit Scope Terminator**
   E.g.: END-IF, END-EVALUATE, END-PERFORM, END-SEARCH etc.

Continue: Continue passes control to Next Statement after Explicit Scope terminator.
Next Sentence: It Passes control to the Next Statement after Implicit Scope Terminator.

E.g.: Here is an example which will explain use of next sentence & continue.
Consider working-storage variable

```
        01 WS-A PIC 9(02) VALUE 10.
        01 WS-B PIC 9(02) VALUE 30.
```

Code:

```
IF WS-A = 10
IF WS-B = 20
        DISPLAY "WS-A : "  WS-A
ELSE
        NEXT SENTENCE
END-IF
        DISPLAY  "SAHASRA INFOTECH"
END-IF.          → Control Passed
        DISPLAY "SUCCESSFULL".
```

On Executing above code, The Control will come to Next Sentence because of Condition codes. The Next Sentence will try to pass control to Next Statement that is after period (i.e., after END-IF), so

Code Output: SUCCESSFULL

Code:

```
        IF WS-A = 10
        IF WS-B = 20
        DISPLAY "WS-A : "  WS-A
        ELSE
CONTINUE
        END-IF          -------  Control
        passed
                DISPLAY " SAHASRA INFOTECH "
        END-IF.
                DISPLAY " SUCCESSFUL ".
```

On Executing above code, The Control will pass to next statement that is after next explicit scope terminator i.e., first END-IF, because of Condition codes.

Code Output:

SAHASRA INFOTECH
SUCCESSFULL

SAHASRA INFOTECH
&
CONSULTING SERVICES

J C L
(Job Control Language)

www.sahasrainfotech.co.in

JOB CONTROL LANGUAGE

- JCL acts as an interface between Application Programming and MVS Operating System.
- JCL is used for Compilation and Execution of Batch Programs.
- Apart from the above functionalities JCL can also be used for,

1. Controlling the Jobs.
2. Create GDG's.
3. Allocate PDS, PS file with IBM Utilities.
4. Create Procs.
5. Sort the files.

JCL Coding Sheet:

1, 2, 3---------- Column Numbers----------72, 73----------80

```
//JOBNAME   JOB     PARAMETER 1, PARAMETER 2,.....          COMMENTS
//                EXEC
//                DD
//*      →Comment (* In 3rd Column Indicates Line is Comment)
//       → End of JCL
```

Where // → Identification Field
JOBNAME → Naming Field
JOB, EXEC, DD → Statement / Operation

E.g.:

```
1----------72----------80
//JOBNAME JOB 123,'SAHASRA', CLASS=A,                    Comments
        NOTIFY=&SYSUID (=> continuation parameters should
        Start in between 4-16 cols in next line)
```

Note:
If we want to continue parameters in the next line end the last parameter with "," and continue next parameter only in between 4 – 16 columns.
There are three statements in JCL.

1) JOB
2) EXEC
3) DD

JOB Statement:

Job statement is used to identify job name and Job related Parameters.
JOBCARD = Job Name + Job Statement + Job related parameters

JOBCARD:

Syntax:

```
//JOBNAME JOB ACCOUNT INFORMATION,'USERNAME',CLASS=A-Z/0-9,
//   NOTIFY=&SYSUID/RACF ID,MSGCLASS=(A-Z)/(0 – 9),MSGLEVEL=(X,Y),
//   PRTY= 0-15,TIME=(M,S),REGION=MB/KB,TYPRUN=SCAN/HOLD/COPY,
//   COND=(RC,OPERATOR,STEPNAME) OR COND=ONLY OR COND=EVEN,
//   RESTART=STEPNAME.
```

1) **ACCCONTING INFORMATION**:
   It is used for billing purpose, in real time when we submit any job it is going to take some CPU time. Based on the CPU time there will be some amount involved, where this amount has to go will be decided by A/C Information Parameter.

2) **USER NAME:**
   It is used to identify the user who has written the JCL.
   Note: Both A/C information and User name are Positional Parameters and the remaining Job card parameters are keyword parameters.

3) **NOTIFY:**
   To which user id the job has to be Notify after successful or Unsuccessful Completion. Successful Completion means MAXCC = 00 (or) 04, Unsuccessful Completion means MAXCC > 04.

4) **CLASS:**
   It is used to assign priorities to the job. Suppose if you submit 10 jobs at the same time which job has to run first time and which job has to run 2nd time and so on will be decided by Class Parameter.
   Note: This class parameter priority is set by MVS Admin Team.

5) **PRTY:**
   When we have multiple jobs with same class parameter then priority comes into consideration.

   PRTY = 15 → Highest Priority.
   PRTY = 0 → Lowest Priority.

   E.g.:

Job1	Job2	Job3
CLASS=X,PRTY=7	CLASS=X,PRTY=6	CLASS=X,PRTY=9

   Then the jobs will be executed in the following sequence Job3, Job1, and Job2.
   When we have CLASS, PRTY parameters same for all jobs then JOBID comes into consideration. JOBLIB is generated by Job Entry Subsystem (JES), it is the processor in MVS.

6) **MSGCLASS:**
It is used for routing (sending) the job messages and statements to output devices like SAR tool, SPOOL, SCHEDULAR, PRINTER etc.
E.g.: MSGCLASS = A → SPOOL B → PRINT C → SAR TOOL.

7) MSGLEVEL:

It is used for listing the messages and statements in the spool. SYNTAX: MSGLEVEL = (X,Y).
If I give MSGLEVEL = (1, 1)→ it is going to generate maximum listing of statements and messages.

- The default option for MSGLEVEL = (1,1)
- MSGLEVEL = (X, Y) → where X (Statements) = 0 / 1 / 2
  Y (Messages) = 0 / 1

8) TIME:
It is the CPU time given to the job. It tells us how much of the time the job has to run in the execution queue.

- Max time for time parameter is
  TIME = NOLIMIT / 1440(MINS) / MAXIMUM (8.25 MONTHS)
- When the given time limit exceeds we get
  S322 → CPU TIME OUT ERROR.
  Other Time Related Errors:
  S222 → USER CANCELS THE JOB.
  S122 → OPERATOR CANCELS THE JOB.

9) REGION:
It is used to allocate space for the entire job to run. Maximum value for region parameter is REGION = 0 M / 0 K.
Region Related Abends: S80A S804 S822

10) RESTART:
It is used to restart the job from a particular step by mentioning the step name.

- **Syntax**: Restart = stepname
- For Procedures: Restart = JOBSTEP NAME.PROCSTEP NAME

11) TYPRUN:
It is used for JCL syntax error checking.
Syntax:

- **TYPRUN=SCAN**, It checks for syntax errors without the job execution. If there are any syntax errors they are notified in spool.
- **TYPRUN=HOLD**, It checks for syntax errors if there are any syntax errors they are notified in spool, if there are no syntax errors the job will be placed in HOLD state, and the HOLD state should be released by the operator(MVS admin).
- **TYPRUN=COPY**, This is default option; the entire JCL will be copied to JESJCL because of this option.

12) COND:

It is used to execute or bypass the particular step in the job. It can be coded at both JOB level and STEP level. Coding the 'COND' parameter at step level is more popular.

If condition is true then the step will bypass and if the COND is false then step will be executed.

Syntax:

- **COND=(RC,OPERATOR,STEPNAME)**, If we don't mention any stepname, then the Return Code (RC) of all previous steps will be compared.
- **COND=ONLY**, The step will be executed if any of the previous steps are abended. We code it only once in a JOB.
- **COND=EVEN**, The step will be executed independent of other steps. We code this multiple times in our JOB.

Note: If we want to bypass a particular step then, Code Condition Code as COND=(00,LE)

EXECUTION STATEMENT (EXEC STMT):

- **EXEC** statement is used to identify program name or proc name.
- Maximum we can code 255 EXEC statements in a JOB.

Syntax:

```
//STEPNAME EXEC PGM= PROGRAM        (OR)
         PROC = PROCEDURE NAME
```

DATA DESCRIPTOR STAEMENT (DD STMT):

- It is used to identify files (input and output) used in JCL.

Syntax:

```
//DDNAME    DD     DSN=FILE NAME,
//        DISP= (CURRENT STATUS,NORMAL TERMINATION,ABNORMAL TERMINATION),
//        UNIT=SYSDA,
//        VOLUME=SER=VOLUME NAME,
//        SPACE=(UNIT,(PRIMARY QUANTITY,SECONDARY QUANTITY,DB),RLSE),
//        DCB=(DSORG=PO/PS,LRECL=N,BLKSIZE=N*10,RECFM=FB/VB)
```

DD name acts as a bridge b/w COBOL program and execution JCL.

DSN is a physical space or file where the records will be stored.

DISP= (CurrentStatus,NormalTermination,AbnormalTermination) can be NEW, OLD, SHR, MOD.

NEW	The file is not present in the system and we are creating the file in this step.
OLD	The file already present and we require exclusive access for accessing that file.
SHR	The file is already present and can be shared across different users.
MOD	The file is already present and we can add the records to the end of sequential file.

NORMAL TERMINATION (Successful Execution):

CATLG	To store the files in known location.
UNCATLG	To store the files in unknown location.
DELETE	Delete the data set or file.
KEEP	Keep the data set on a particular volume.
PASS	To pass the data set or file to the subsequent steps.

ABNORMAL TERMINATION:

- It indicates the unsuccessful execution and allowed parameters are similar to normal termination except pass.

Q) What is the max value we can give for BLKSIZE?
Ans) BLKSIZE= 0 / 32788

SYSPRINT	It is used to print the errors in the spool.
SYSOUT	Routing the output to the spool.

INSTREAM DATA it is identified starting with * and ending with /*.

Syntax:

```
//SYSIN DD *
  Values
/*
```

- In stream data is used to pass values from JCL to COBOL dynamically.
- To accept the values in COBOL program, we should have equivalent accept verbs.

PARM PARAMETER:

- It is used to pass values from JCL to COBOL program during execution time. It is coded on EXEC statement
- To receive the value coded in PARM parameter we should have Linkage section in COBOL program. Maximum we can pass 100 characters through PARM parameter.

Syntax:

```
//STEPNAME EXEC PGM=PGMNAME,PARM=(VALUE1, VALUE2, VALUE3,.....VALUEN)
```

LINKAGE SECTION.

```
01 LS-INPUT.
    05   PARM-LENGTH    PIC   S9(4) COMP.
    05   PARM-ID        PIC   9(5).
    05   PARM-NAME      PIC   X(10).
```

- If we have special char enclose them with in single quotes.

E.g.:

```
//JOBCARD
//STEP1 EXEC PGM=PGM1,PARM=(15000,'SAHASRA','500+800')
```

E.g.: COBOL PGM:

```
IDENTIFICATION DIVISION
PROGRAM-ID. PARMPGM.
ENVIRONMENT DIVISION.
DATA DIVISION.
LINKAGE SECTION.
    01    LS-DATA.
          05    LS-LENGTH        PIC    S9(4)COMP.
          05    LS-VAR1          PIC    9(3).
          05    LS-VAR2          PIC    X(5).
          05    LS-VAR3          PIC    X(5).
    PROCEDURE DIVISION USING LS-DATA.
          DISPLAY    LS-LENGTH.
          DISPLAY    LS-VAR1.
          DISPLAY    LS-VAR2.
          DISPLAY    LS-VAR3.
          STOP RUN.
```

EXECUTION JCL:

```
//JOBCARD
//STEP1      EXEC   PGM=PGM1,PARM=(123,'SAHAS','VENKY')
//STEPLIB    DD     DSN=FSS234.SAHASRA.LOADMODULE,DISP=SHR
//SYSPRINT   DD     SYSOUT=*
//SYSOUT     DD     SYSOUT=*
//
```

DIFFERENT LIBRARIES IN JCL: There are 3 different Libraries namely.

1) STEPLIB
2) JOBLIB
3) JCLLIB.

STEPLIB: It is used to specify the path of the load module. It is coded after EXEC statement.

JOBLIB: JOBLIB is also used to specify the path of the Load module. Using JOBLIB we can specify the path of all steps load modules.

- JOBLIB is coded after job statement.
- Generally it is better to write JOBLIB than STEPLIB.

Note: In real time projects, all the programs loads will be present in single Load PDS.

E.g.:

```
//JOBNAME JOB 213,'SAHASRA',NOTIFY=&SYSUID
//JOBLIB  DD DSN=FSS234.TEAM.LOADLIB,DISP=SHR
//          DD DSN=FSS234.XYZ.LOADLIB,DISP=SHR
//STEP1 EXEC PGM=PGM1
```

```
---------------
NO STEPLIB
---------------
//STEP2   EXEC PGM=PGM2
//STEPLIB DD DSN=FSS234.ABC.LOADLIB,DISP=SHR
-----------
-----------
//STEP3 EXEC PGM=PGM3
```

***Note*:**

- The EXEC statement will be checking for load module first in STEPLIB, if it is not present in STEPLIB it checks in JOBLIB if not found then it checks in SYSLIB.
- If the load is not present in any of the libraries then the job will be abended with S806.

Q) If I code both JOBLIB and STEPLIB in the job which one will be having higher priority?
Ans) STEPLIB.

JCLLIB: it is used to identify the path of the catalog procedures. It is coded after job statement.
Syntax:

```
//JOBNAME JOB 123,.....
//PROCLIB JCLLIB ORDER=FSS234.TEAM.PROCLIB
```

DIFFERENT UTILITIES BY IBM TO REDUCE THE MANUAL EFFORT:

1) IEBGENER
2) IEBCOPY
3) IEFBR14
4) IEBCOMPR
5) DFSORT / SORT UTILITY

1) IEBGENER:

- IEBGENER is used to copy one PS File to another PS File or group of PS Files (Concatenate) to another PS File.

Rules for Concatenation:

- All the Input Files should have same Logical Record Length and same Record Format.
- The block size can be different; the largest block size file should be placed first.
- Max we can concatenate up to 255 PS files.

JCL to Copy One PS to another PS File :

```
//JOBNAME JOB 123,'SAHASRA',NOTIFY=&SYSUID
//STEP1   EXEC PGM=IEBGENER
//SYSUT1 DD   DSN=FSS234.TEAM.CUSTFILE,DISP=SHR
//SYSUT2 DD   DSN=FSS234.TEAM.CUSTFILE.NEW,
//           DISP=(NEW,CATLG,DELETE),
```

```
//                SPACE=(TRK,(1,1),RLSE),
//                DCB=(DSORG=PS,LRECL=80,BLKSIZE=800,RECFM=FB)
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN    DD DUMMY
```

JCL to Concatenate PS Files :

```
//JOBNAME JOB 123,'SAHASRA',NOTIFY=&SYSUID
//STEP1   EXEC  PGM=IEBGENER
//SYSUT1  DD    DSN=FSS234.TEAM.CUSTFILE1,DISP=SHR
//          DD    DSN=FSS234.TEAM.CUSTFILE2,DISP=SHR
//          DD    DSN=FSS234.TEAM.CUSTFILE3,DISP=SHR
//          DD    DSN=FSS234.TEAM.CUSTFILE4,DISP=SHR
//SYSUT2  DD     DSN= FSS234.TEAM.CUSTFILE.CONCAT,
//                 DISP=(NEW,CATLG,DELETE),
//                 SPACE=(TRK,(1,1)RLSE),
//                 DCB=(DSORG=PS,LRECL=80,BLKSIZE=800,RECFM=FB)
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN    DD DUMMY
```

2) IEBCOPY:

- It is used to copy members from one PDS to another PDS.
- It is also used to copy selective members from one PDS to another PDS.
- It is also used to exclude selective members from one PDS to another PDS.

JCL to Copy All Members from PDS to another PDS:

```
//JOBNAME JOB 123,'SAHASRA',NOTIFY=&SYSUID
//STEP1   EXEC PGM=IEBCOPY
//SYSUT1  DD   DSN=FSS234.TEAM.COBOL,DISP=SHR
//SYSUT2  DD   DSN=FSS234.TEAM.COBOL.BKP,
//               DISP=(NEW,CATLG,DELETE),
//               SPACE=(TRK,(10,10,12),RLSE),
//               DCB=(DSORG=PO,LRECL=80,BLKSIZE=800,RECFM=FB)
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN    DD *
  COPY INDD=SYSUT1,
       OUTDD=SYSUT2
/*
//
```

FOR SELECTIVE COPY:

Syntax:

```
//SYSIN DD *
COPY INDD=SYSUT1,
        OUTDD=SYSUT2
SELECT MEM/MEMBER=(MEMBER1,MEMBER2,MEMBER3)
/*
//
```

FOR EXCLUDE MEMBERS:

Syntax:

```
//SYSIN DD *
COPY INDD=SYSUT1,
        OUTDD=SYSUT2
EXCLUDE MEM/MEMBER=(MEMBER1,MEMBER2,MEMBER3)
/*
//
```

3) IEBCOMPR:

- It is used to compare 2 PS Files and it is also used to compare members of PDS.

PS Compare:

```
//JOBNAME JOB 123,'SAHASRA',NOTIFY=&SYSUID
//STEP1     EXEC PGM=IEBCOMPR
//SYSUT1    DD   DSN=FSS234.TEAM.CUSTFILE1,DISP=SHR
//SYSUT2    DD   DSN=FSS234.TEAM.CUSTFILE2,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//SYSOUT    DD   SYSOUT=*
//SYSIN     DD *
  COMPARE TYPORG=PS
/*
//
```

PDS Compare:

```
//JOBNAME JOB 123,'SAHASRA',NOTIFY=&SYSUID
//STEP1     EXEC PGM=IEBCOMPR
//SYSUT1    DD DSN=FSS234.TEAM.COBOL(MEM),DISP=SHR
//SYSUT2    DD DSN= FSS234.TEAM.COBOL(MEM2),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSIN     DD *
  COMPARE TYPORG=PO
/*
//
```

4) IEFBR14:

- It is used to allocate or delete a PS File or PDS File.

Allocating PS File:

```
//JOBNAME JOB 123,'SAHASRA',NOTIFY=&SYSUID
//STEP1   EXEC PGM=IEFBR14
//SYSUT1 DD    DSN=FSS234.TEAM.CUSTFILE,
//                DISP=(NEW,CATLG,DELETE),
//                SPACE=(TRK,(1,1),RLSE),
//                DCB=(DSORG=PS,LRECL=80,BLKSIZE=800,RECFM=FB)
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN      DD DUMMY
```

Allocating PDS File:

```
//JOBNAME JOB 123,'SAHASRA',NOTIFY=&SYSUID
//STEP1   EXEC PGM=IEFBR14
//SYSUT1 DD    DSN=FSS234.TEAM.COBOL,
//                DISP=(NEW,CATLG,DELETE),
//                SPACE=(TRK,(10,10,12),RLSE),
//                DCB=(DSORG=PO,LRECL=80,BLKSIZE=800,RECFM=FB)
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN      DD DUMMY
```

Deleting PS / PDS File:

```
//JOBNAME JOB 123,'SAHASRA',NOTIFY=&SYSUID
//STEP1   EXEC PGM=IEFBR14
//SYSUT1 DD    DSN=PS/PDS FILE,
//                DISP=(MOD,DELETE,DELETE)
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN      DD DUMMY
```

Note:
Generally in most of the JCLs we code deletion of PS files as the first step in the job. We delete all the new files created in the job by using IEFBR14 delete. It avoids duplicate data set error.

5) SORT UTILITY:

- SORT Utility is used to sort the records in the input file based on the conditions specified and then write into output file.
- The input DD name should be given as SORTIN and the output DD name should be given as SORTOUT.

There are two types of sorts:

1) Internal Sort (Sorting the files using Cobol program)
2) External Sort (Sorting the files using JCL), this is also called DFSORT.

***Note*:** We don't use internal sort in any of the project.

Syntax:

```
SORT FIELDS=(STARTING POSITION,LENGTH,TYPE,ASC/DSC),
TYPE=CH(ALPHANUMERIC/NUMERIC),
PD(COMP-3),
BI(COMP)
```

Retrieving the Values in Ascending Order Using Sort Utility:

```
//JOBNAME JOB 123,'SAHASRA',NOTIFY=&SYSUID
//STEP1     EXEC PGM=SORT
//SORTIN    DD   DSN=FSS234.TEAM.CUSTFILE,DISP=SHR
//SORTOUT DD   DSN=FSS234.TEAM.CUSTFILE1,
//                DISP=(NEW,CATLG,DELETE),
//                SPACE=(TRK,(1,1)RLSE),
//                DCB=(DSORG=PS,LRECL=80,BLKSIZE=800,RECFM=FB)
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN     DD *
  SORT FIELDS=COPY, -------→ Default
  SORT FIELDS=(23,8,CH,ASC)
 /*
 //
```

Sort on 2 Fields:

```
//SYSIN DD *
  SORT FIELDS=(26,8,CH, ASC,  1,6,CH,ASC)
               (Major Key)    (Minor Key)
```

JCL to copy selected range of records from input file to the output file:

```
//SYSIN DD *
  SORT FIELDS=COPY,
  SKIPREC=50          →How many records we need to skip from input file.
  STOPAFT=10          →How many records we need to copy to output file.
/*
//
```

INCLUDE:

- Include is used to copy the records to output files based on some condition checking.

INCLUDE Syntax:

INCLUDE COND =(STARTING POS, LENGTH,TYPE,OPERATOR,C'VALUE')

Operator = EQ/GT/LT/GE/LE

E.g.:

```
//SYSIN   DD *
  SORT   FIELDS=COPY,
  INCLUDE COND=(41,2,CH,EQ,C'TX')
/*
//
```

OMIT Syntax:

OMIT COND=(STARTING POS, LENGTH,TYPE,OPERATOR,C'VALUE')

```
//SYSIN   DD *
  SORT   FIELDS=Copy
  OMIT  COND=(41,2,CH,EQ,C'TX')
/*
//
```

JCL to Eliminate Duplicates Records in a File:

```
//SYSIN DD *
  SORT  FILEDS=(26,8,CH,ASC),EQUALS,
  SUM   FILEDS=NONE          → which eliminates duplicates from File.
/*
//
```

***Note*:**

If we code EQUALS, Then only first record will be copied and other records will be eliminated.

If we don't code equals any of the records will be eliminated and any of the record will be copied to output file. (Not sure whether it is 1st One /2nd One or.../)

JCL to Copy the Duplicate Records form Input File:

```
//JOBNAME JOB 123,'SAHASRA',NOTIFY=&SYSUID
//STEP1    EXEC PGM=SORT
//SORTIN    DD   DSN=FSS234.TEAM.CUSTFILE,DISP=SHR
//SORTOUT DD   DSN=FSS234.TEAM.CUSTFILE.UNIQUE,
//                DISP=(NEW,CATLG,DELETE),
//                SPACE=(TRK,(1,1),RLSE),
//                DCB=(DSORG=PS,LRECL=80,BLKSIZE=800,RECFM=FB)
//SORTXSUM DD  DSN=FSS234.TEAM.CUSTFILE.DUP,
//                DISP=(NEW,CATLG,DELETE),
//                SPACE=(TRK,(1,1),RLSE),
//                DCB=(DSORG=PS,LRECL=80,BLKSIZE=800,RECFM=FB)
//SYSPRINT DD    SYSOUT=*
//SYSOUT   DD    SYSOUT=*
```

```
//SYSIN      DD *
  SORT FIELDS=COPY
  SUM FIELDS=(NONE,XSUM)
/*
//
```

Sort JCL to Split Single File into Multiple Files:

```
//JOBNAME JOB 123,'SAHASRA',NOTIFY=&SYSUID
//STEP1     EXEC PGM=SORT
//SORTOF1 DD    DSN=I/P FILE,DISP=SHR
//SORTOF2 DD    DSN=O/P FILE,
//                DISP=(NEW,CATLG,DELETE),
//                SPACE=(TRK,(1,1),RLSE),
//                DCB=(DSORG=PS,LRECL=40,BLKSIZE=400,RECFM=FB)
//SORTOF3 DD    DSN=O/P FILE2,
//                DISP=(NEW,CATLG,DELETE),
//                SPACE=(TRK,(1,1),RLSE),
//                DCB=(DSORG=PS,LRECL=40,BLKSIZE=400,RECFM=FB)
//SORTOF4 DD    DSN=O/P FILE3,
//                DISP=(NEW,CATLG,DELETE),
//                SPACE=(TRK,(1,1),RLSE),
//                DCB=(DSORG=PS,LRECL=40,BLKSIZE=400,RECFM=FB)
//SYSPRINT DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
//SYSIN      DD *
  OUTFIL FILES=2 INCLUDE=(41,2,CH,EQ,C'TX')
  OUTFIL FILES=3 INCLUDE=(41,2,CH,EQ,C'NY')
  OUTFIL FILES=4 INCLUDE=(41,2,CH,EQ,C'AL')
/*
//
```

GDG'S (Generation Data Groups)

- When we want to store important data over a period of time we go for GDG's.
- GDG's are collection of data sets which are functionally related to each other.

Advantages of GDG's:

- Maintains historical information by creating generations
- It overcomes duplicate dataset error and helps in automating or scheduling the jobs.

❖ Assume we have created a GDG base with name USERID.NAME.GDG, and then its generations will be created as follows.

Syntax:

USERID.NAME.GDG → BASE GDG

USERID.NAME.GDG.GNNNNVMM → GENERATIONS OF GDG

E.g.: 1st Run ← FSS234.XYZ.GDG.G0001V00 (PS FILES)
2nd Run ← FSS234.XYZ.GDG.G0002V00

FSS234.XYZ.GDG.GNNNNVMM
GNNNN →GENERATIONS MAX OF 9999
VMM →VERSIONS MAX OF 99

- Version numbers need to be changed manually. System won't increment version numbers generally we increment version number when we need to update the file.
- Don't change in the existing file.
- Copy V00 generation to V01 generation and then modify the V01 generation file.

Steps Required to Work on GDG's:

1. Create a write file program.
2. Compile the Program.
3. Define the GDG base with IDCAMS Utility
4. Create a model dataset with IEFBR14 Utility
5. Write execution JCL (in execution JCL we have to make use of GDG base and Model dataset).

GDG Base Creation JCL:

```
//JOB CARD
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN DD *
  DEFINE GDG(NAME(USERID.NAME.GDG)-
  LIMIT(N)-      → WHERE  0 < N <=255
  EMPTY/NO EMPTY-
  SCRATCH/NO SCRATCH-
  EXPIRY DATE(SOME DATE)-
  OWNER(M/F USERID)
/*
//
```

Different Parameters used for Defining GDG Base:

1. Name
2. Limit
3. Empty / No Empty
4. Scratch / No Scratch
5. Expiry Date
6. Owner

1) **NAME:**
   - Name is used to name the GDG base.

2) **LIMIT:**
   - It is used to specify the no of generations the GDG base can create at a time. Maximum we can create 255 generation at a time.

3) **EMPTY:**
   - When the limit is reached and when we specify option as empty then all the previous generations will be un-cataloged.

   E.g.: If Limit=5 and empty Assume we have submitted job for 6 times.

   USERID.NAME.GDG.G0001V00
   USERID.NAME.GDG.G0002V00
   USERID.NAME.GDG.G0003V00
   USERID.NAME.GDG.G0004V00
   USERID.NAME.GDG.G0005V00 → Up to here these are created Generations.
   USERID.NAME.GDG.G0006V00 → Only this generation will be Present.

4) **NO EMPTY:**
   - When the limit is reached and we have specified the option as no empty then the oldest Generations can un-cataloged.

   E.g.: If Limit=5 and no empty and assume we have submitted job for 6 times.

   USERID.NAME.GDG.G0001V00 → It will be Un-Cataloged.
   USERID.NAME.GDG.G0002V00
   USERID.NAME.GDG.G0003V00
   USERID.NAME.GDG.G0004V00
   USERID.NAME.GDG.G0005V00
   USERID.NAME.GDG.G0006V00 → Up to here these are created Generations.

   - The number of generations at any time will be equal to the limit specified.

5) **SCRATCH:**
   - When we specify scratch then all the un-cataloged Generations will be Permanently or Physically Deleted.

6) **NO SCRATCH:**
   - When we specify no scratch then the un-cataloged Generations will be Temporarily / Not Physically Deleted.

7) **EXPIRY DATE:**
   - It specifies the date when the GDG base should be Expired / Deleted.

Note: If GDG base is deleted then all its associated Generations will be Deleted.

8) **OWNER:**
   - In owner we specify the user id of person who is creating the GDG base.

Create Model Data Set:

```
//JOB CARD
//STEP1   EXEC PGM=IFBR14
//SYSUT1 DD    DSN=USERID.NAME.ModelDataSet,
//                DISP=(NEW,CATLG,DELETE),
//                SPACE=(TRK,(1,1),RLSE),
//                DCB=(DSORG=PS,LRECL=40,BLKSIZE=400,RECFM=FB)
//SYSOUT DD   SYSOUT=*
```

EXECUTION JCL:

```
//JOB CARD
//STEP1    EXEC PGM=WRIFIL
//STEPLIB  DD    DSN=LOAD PDS,DISP=SHR
//CUSTDD DD    DSN=USERID.NAME.GDG(+1),
//                DISP=(NEW,CATLG,DELETE),
//                SPACE=(TRK,(1,1),RLSE),
//                DCB=(USERID.NAME.MODELPS)
//SYSPRINT DD  SYSOUT=*
//SYSOUT    DD  SYSOUT=*
//SYSIN      DD *
  VALUES
/*
//
```

Note:

The GDG Generations will be created after the job compilation (not after step compilation) where as PS files will be created after step compilation.

How to Identify the Generations :

GDG Base(0) Current Generation.
GDG Base(+n) Future Generation.
GDG Base(-n) Previous Generation.

Q) Assume my job is having 3 steps and in step1 I am creating a new Generation (i.e. userid.name.GDG.G0008V00) then how should i code GDG( ) and what should be my DISP parameter in step1 and in step3 and I want to use the Generation created in step1 as input then what should I give GDG base( ) and DISP parameter in step3.

Ans) In step1 we have to give
GDG BASE(+1), DISP=(NEW,CATALG,DELETE) IN STEP3
USERID.NAME.GDG(+1),DISP=SHR

E.g.:

```
//JOB CARD
//STEP1    EXEC PGM=WRIFIL
//STEPLIB  DD    DSN=LOAD PDS,DISP=SHR
//CUSTDD  DD    DSN=USERID.NAME.GDG(+1),
```

```
//                DISP=(NEW,CATLG,DELETE),
//                SPACE=(TRK,(1,1),RLSE),
//                DCB=(USERID.NAME.MODELPS)
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN    DD *
  VALUES
  --------
  ---------
/*
//STEP2 EXEC PGM=PGM2
   --------
   ---------
// STEP2  EXEC PGM=IEBGENER
//SYSUT1 DD   DSN=USERID.NAME.GDG(+1),DISP=SHR
//SYSUT2 DD   DSN=FSS234.TEAM.CUSTFIL
       ---------
       ---------
//
```

Q) Assume your job got abended in step2 what are necessary modifications we need to do for the JCL?

Ans) 1) Restart=step2 in job card.
2) Change GDG(+1) to GDG(+0) in step3. (Because step1 is submitted and generation is created in step1.

Q) Write a job where I need to create G0009 and G0010, G0012 in a single job.

Ans) GDG Base(+1) in step1 → 8=1=9
GDG Base(+2) in step2 → 8+2=10
GDG Base(+3) in step3 → 8+3=11
GDG Base(+4) in step4 → 8+4=12

Q) Write a step to delete all Generations at a time?

Ans)

```
//JOBCARD
//STPE1 EXEC PGM=IEFBR14
//SYSUT1 DD DSN= GDG BASE NAME,
//                DISP=(MOD,DELETE,DELETE)
//
```

Note: GDG base has pointers to all its generation's. Assume if we want to read the data present in all generation's we can do that by specifying the GDG base name.

PROCEDURES (PROCS):

- When we want to repeat a group of JCL statements multiple times we go for procedures.
- Procs is a collection of pretested JCL statements.

Syntax:
```
PROCNAME  PROC
          STATEMENT—1
          STATEMENT—2
                :
                :
          STATEMENT—n
          PEND.
```

There are 2 types of PROCS namely,

1. Instream Procs.
2. Catalog Procs.

Advances of Procs:

- Reduces Memory Space.
- Reusability.
- Reduces Manual Effort and Manual Errors.

INSTREAM PROC:

- Instream proc starting is identified by proc statement and ended with PEND statement.
- PEND is mandatory statement in instream proc.
- Instream procs are limited for only single job.

Note:
When we define a proc, the proc will not be executed we need to call or invoke the proc using EXEC statement. We can create maximum 15 instream procs in a job.

Syntax:
```
//STEPNAME EXEC PROC=PROCNAME
                Or
//STEPNAME EXEC PROCNAME
```

Date Set Overwriting:
```
//STEPNAME IN PROC.DDNAME IN PROC DD DSN=NEW FILE NAME
```

E.g.:
```
//JOBCARD
//INPROC PROC
//STEP1     EXEC   PGM=IEFBR14
//SYSUT1    DD     DSN=FF244.TEAM.CUSTFILE,
//                 DISP=(NEW,CATLG,DELETE),
```

```
//                    UNIT=SYSDA,
//                    SPACE=(TRK,(1,1),RLSE),
//                    DCB=(DSORG=PS,LRECL=40,BLSIZE=400,RECFM=FB)
//SYSPRINT    DD      SYSOUT = *
//SYSOUT      DD      SYSOUT = *
//           PEND     (Ending of Proc)
//STEP2       EXEC    PROC = INPROC
//STEP3       EXEC    INPROC
//STEP1.SYSUT1  DD    DSN=FSS244.TEAM.CUSTFIL1
//STEP4       EXEC    INPROC
//STEP1.SYSUT1  DD    DSN=FSS244.TEAM.CUSTFIL2
```

CATALOG PROCEDURE:

- In catalog PROCS we define the group of statements in a member of PDS, we call the member in JCL using EXEC statement and we specify the path of catalog proc using JCLLIB.

Syntax: //PROCLIB JCLLIB ORDER=USERID.NAME.PROCLIB

- PEND statement is optional in catalog PROCS. Once we create catalog proc, the proc can be used in any number of jobs.

Steps Required to Work on Catalog Procs:

- Create a new PDS with name USERID.NAME.PROCLIB
- Create a member with name (CATPROC)
- Write all the JCL statements to be repeated in member
- Call the proc in JCL using EXEC Statement
- Specify the path of PDS using JCLLIB

Example for Catalog Proc:

```
//JOBCARD
//PROCLIB JCLLIB ORDER = USERID.NAME.PROCLIB
//STEP2     EXEC    PROC = CATPROC
//STEP3     EXEC    CATPROC
//STEP1.SYSUT1  DD  DSN=FSS244.TEAM.FILE1
//STEP4     EXEC    CATPROC
//STEP1.SYSUT1  DD  DSN=FSS244.TEAM.TFILE2
//
```

Note:

- INSTREAM data is not allowed in procedures.
- In PROCS we pass the data using control cards.

CONTROL CARDS:

- Control cards are used to pass data from JCL to COBOL programs.
- Control card is a member of PDS (Control card PDS)
- As in procs we can't pass data through instream we use control cards to pass the data.
- Control cards provide security to the input data.

Syntax:

```
//JOBCARD
//INPROC PROC
//STEP1       EXEC   PGM=WRIFILE
//STEPLIB     DD     DSN=LOADPDS,DISP = SHR
//CUSTDD      DD     DSN=FF244.TEAM.CUSTFILE,
//                   DISP=(NEW,CATLG,DELETE),
//                   UNIT=SYSDA,
//                   SPACE=(TRK,(1,1),RLSE),
//                   DCB=(DSORG=PS,LRECL=40,BLKSIZE=400,RECFM=FB)
//SYSOUT      DD     SYSOUT=*
//SYSPRINT    DD     SYSOUT=*
//SYSIN       DD     DSN=USERID.NAME.CONTROL(WRIINPUT),
//                   DISP=SHR
//                   PEND
//STEP2       EXEC   PROC=INPROC
//STEP3       EXEC   INPROC
//STEP1.CUSTDD  DD   DSN=FSS244.TEAM.CUSTFIL1
//STEP1.SYSIN   DD   DSN=USERID.NAME.CONTROL(WRIINP1),DISP=SHR
//
```

Q). Different Parameters That Can Be Written Both At Job Level And Exec Level:

1. Condition Parameter 2. Time 3. Region

Note:

Condition and region parameter when coded in both job and exec level, Job will have higher priority. Time when coded in both job and exec level the exec level will be having more priority.

PARAMETER OVERWRITING:

- When we call the proc we can overwrite the parameter value with the help of parameter overwriting.

Syntax: PARAMETERNAME.STEPNAME IN PROC=NEW VALUE.

E.g.:

```
//JOBCARD
//INPROC PROC
//STEP1       EXEC PGM=PGM1,TIME=2M,REGION=6K
```

```
                .....
                .....GROUP OF JCL STMT'S.....
                .....
//              PEND
//STEP2         EXEC INPROC,TIME.STEP1=4M,REGION.STEP1=10K
```

Syntax for Nullifying the Parameter Value:

PARAMETER.STEPNAME=

SYMBOLIC PARAMETER:

- Symbolic parameter is identified by '&' Symbol. Instead of hard coding the values we use variables with prefix '&' (Symbolic Variable) then we pass the values for variable when we invoke or call the proc.

E.g.:

```
//JOBCARD
//INPROC PROC
//STEP1         EXEC    PGM=WRIFILE
//STEPLIB       DD      DSN=LOADPDS,DISP = SHR
//CUSTDD        DD      DSN=FF244.&SLD.CUSTFILE,
//                      DISP=(&X,CATLG,DELETE),
//                      UNIT=SYSDA,
//                      SPACE=(TRK,(&Y,&Z),RLSE),
//                      DCB=(DSORG=PS,LRECL=40,BLKSIZE=400,RECFM = FB)
//SYSOUT        DD      SYSOUT=*
//SYSIN         DD      DSN=USERID.NAME.CONTROL(&MEM),
//                      DISP=SHR
//                      PEND
//STEP2         EXEC    INPROC,SLD=SAHASRA,X=NEW,Y=10,Z=5,MEM=WRIIP1
//STEP3         EXEC    INPROC,SLD=INFOTECH,X=NEW,Y=12,Z=6,MEM=WRIIP2
//
```

JCL ABEND CODES

S No	JCL Error Code	Error Description
1	**SB37**	End of volume. Increase the size of Primary & Secondary memory reasonably. If we increase size blindly then also we will get this abend.
2	**SD37**	Secondary space not given, we have coded only Primary and it is already filled.
3	**SE37**	End of volume. This is same as SB37 usually we get this abend for PDS.
4	**S80A** or **S804**	When the requested region is not enough for the program to run.
5	**S822**	When the required region is not available.
6	**S122**	Operator cancelled your job as it requests some unavailable resource.
7	**S222**	User cancelled job.
8	**S322**	Time out. Refer time parameter for solution.
9	**S522**	Job exceeded maximum wait time.
10	**S722**	Spool limit exceeded, Output lines exceeded the limit set by lines parameter.
11	**S706**	Load module found but it is not executable.
12	**S806**	Load module not found.
13	**SOC1**	Operation exception. Misspelled DD names.
14	**SOC4**	Protection exception. Trying to access a memory location for which you do not have access.
15	**SOC5**	Addressing exception. Trying to access a memory location that is not available in memory.
16	**SOC7**	Data exception. Non-numeric operation on numeric field. It is usually due to un initialized numeric item.
17	**SOCA** or **SOCB**	Decimal point. Overflow and divide exception respectively.
18	**SOCC** or **SOCD**	Floating point. Exception under flow and over flow exception respectively.
19	**S013**	Open problem usually this abend occurs when the program tries to read a member of PDS and the member is not found.

SAHASRA INFOTECH
&
CONSULTING SERVICES

V S A M
(Virtual Storage Access Method)

www.sahasrainfotech.co.in

VIRTUAL STORAGE ACCESS METHOD

- VSAM is IBM's latest and most advanced access method. It makes efficient use of virtual storage of operating system and hence named as virtual storage access method. VSAM is a high performance access method used in MVS operating system.
- Access method services (AMS) is a service program that helps to allocate maintain and delete Datasets. It currently consists of one utility program called IDCAMS.

❖ IDCAMS is a multipurpose utility that can be used for performing following functions.
- Define VSAM Datasets.
- Define GDG Base.
- Copy any file to any file and merge the files.
- Delete VSAM, NON-VSAM and GDG Base Datasets.
- Print the records in the file in different formats.
- Alter the attributes of VSAM Datasets.

Advantages:
- The access of records is faster because of efficiently organized index.
- We can add the records in middle because of imbedded free space in control intervals and control areas.
- The records in VSAM Datasets are physically deleted.
- VSAM provides data security through password protection of a dataset at different levels such as read and update.
- VSAM Datasets can be shared across different regions.
- VSAM Datasets are device independent.

Disadvantage:
- Free space must be left in control interval for inserting the records in future, this leads to memory wastage.

There are four types of Datasets.
1. **ESDS** → Entry Sequenced Data Set.
2. **KSDS** → Key Sequenced Data Set.
3. **RRDS** → Relative Record Data Set.
4. **LDS** → Linear Data Set.

- In VSAM the files are known as Clusters.
- Cluster contain two components
  1. Data Component. 2. Index Component.
- The records are present in data component and the indexed component contains a pointer to the records present in data component.
- In PS files the records are stored in Blocks (Block can contain one or more records).
- In VSAM the records are stored in unit of record storage called a CONTROL INTERVAL (CI).

CONTROL INTERVAL (CI):

Control Interval is a collection of records. There are four components in a CI.

1. Data.
2. Freespace.
3. Record Description Field (RDF).
4. Control Information Description Field (CIDF).

<----------Data---------->

Record1	Record2	Record3	Freespace	RDF	RDF	CIDF

DATA The data component consists of records.

Freespace Freespace helps in inserting the records in middle.

RDF (Record Description Field) It is of three byte long. It is used to specify whether the record is fixed length or variable length or spanned record. A CI may contain one or more RDF's.

CIDF (Control Information Description Field): It is four bytes long and occupies the last four bytes of control interval. It contains information about the freespace available within the CI.

Note: Both RDF and CIDF components are called as Control Fields. CIDF, RDF and Freespace are transparent to application program.

The records are classified based on record size parameter (We will learn this parameter during Cluster creation)

Syntax: RECORDSIZE (AVERAGE LENGTH MAX LENGTH)

Fixed Length Records:

If both Average length and max length are same we call them as FIXED LENGTH RECORDS.

E.g.: RECORDSIZE(80 80)

Variable length records:

If both Average length and max length are different we call them as VARAIABLE LENGTH RECORDS.

E.g.: RECORDSIZE(80 100)

Spanned Records:

When a record size is larger than the control interval size, the record must be contained a more than one control interval. Such a record is called as SPANNED RECORD.

Note: Freespace left by spanned record in a C.I cannot be used by another record.

CONTROL AERA:

Control Area is a collection of control intervals. A Control Area should contain minimum of two Control Intervals, the minimum size of control area is **1** Track and the maximum size is **1** cylinder.

CONTROL INTERVAL							CONTROL FIELDS
Record 1	Record 2	Record 3	Record 4	Record 5	Record 6	Record 7	
Control Interval							
Control Interval							
Control Interval							
Control Interval							
Control Interval							
Control Interval							
Control Interval							
Control Interval							
Control Interval							
Control Interval							
Control Interval							

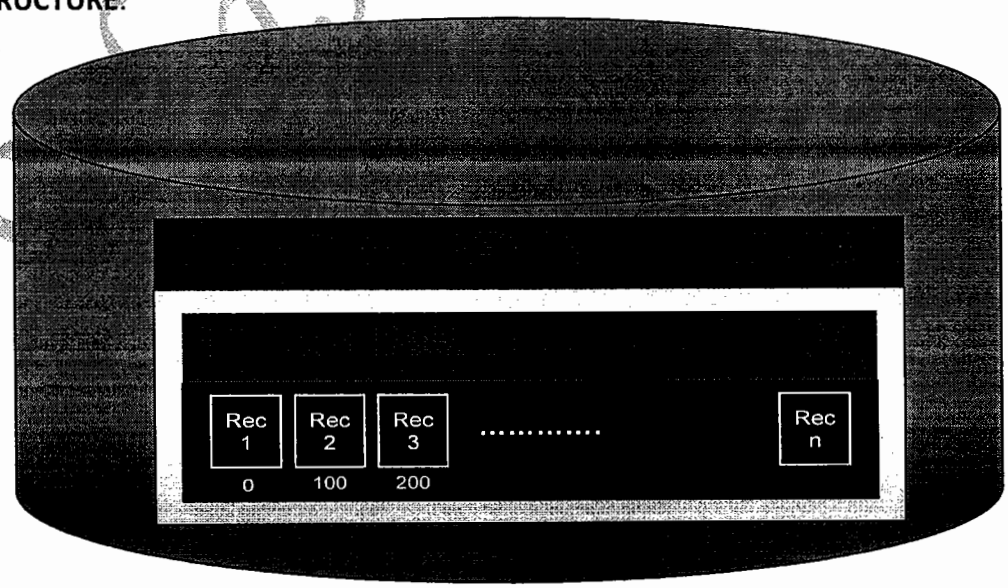
CI Split and CA Split:

- When we want to insert a record in control interval and if the free space provided is not sufficient then the record will be moved from one control interval to another control interval this is called CONTROL INTERVAL SPLIT.
- When the entire control area free space is completed the record will be moved from one control area to another control area this is called CONTROL AREA SPLIT.

ESDS (ENTRY SEQUENCED DATA SET):

- Records can be accessed sequentially on RBA value or directly by supplying the RBA of desired record.
- It contains only Data component.

ESDS STRUCTURE:



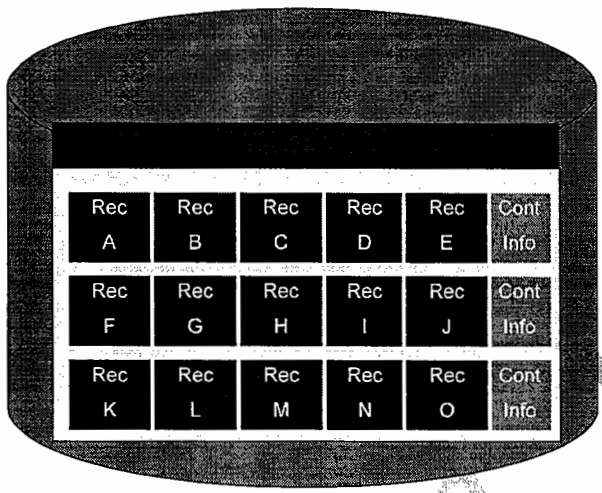
100 Byte Records:

Record 1 RBA = 0
Record 2 RBA = 100
Record 3 RBA = 200, Etc.,

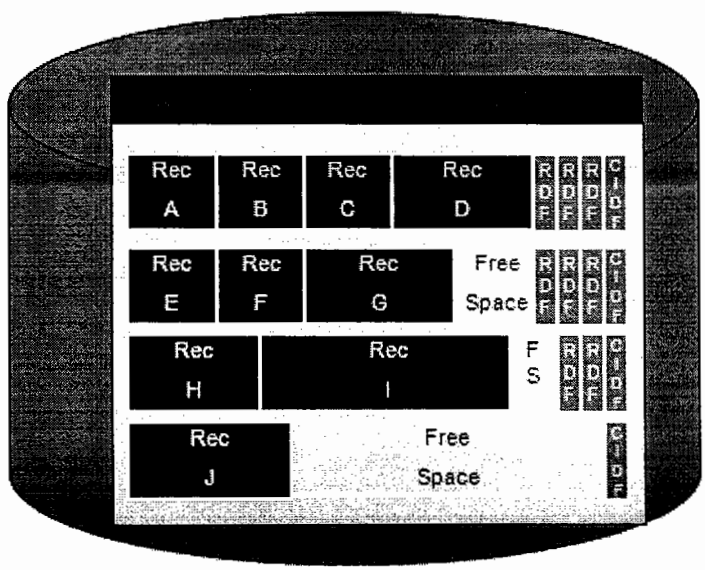
ESDS CONTROL INTERVAL:

- Each CI is filled before records are written into the next control interval in sequence
- Control Areas are filled with C.I's that contain records

ESDS Control Interval



ESDS Control Area



ESDS PROCESSING:

- ESDS can be updated using REWRITE either
  - Sequentially on RBA **or** Directly with given RBA.
- ESDS deletion is only logical, not physical space is not reclaimed.
  - VSAM just sets a code denoting logical deletion.
- Records are loaded in physical sequence.
- VSAM fills each control interval with as many records as will fit.
- No index.
- Any leftover space at end of each CI becomes free space.
- This free space is NOT usable in subsequent insertions because all records are added at the end of the data set.

ESDS RETRIEVAL:

- For sequential processing VSAM.
  - Retrieves the records in the sequence in which they are stored in the data set.
  - Can also be retrieved in descending order starting with any record whose RBA is known.
- For direct processing VSAM.
  - Must be given the RBA of the record to be read.

Note: We can create alternate index for ESDS Datasets.

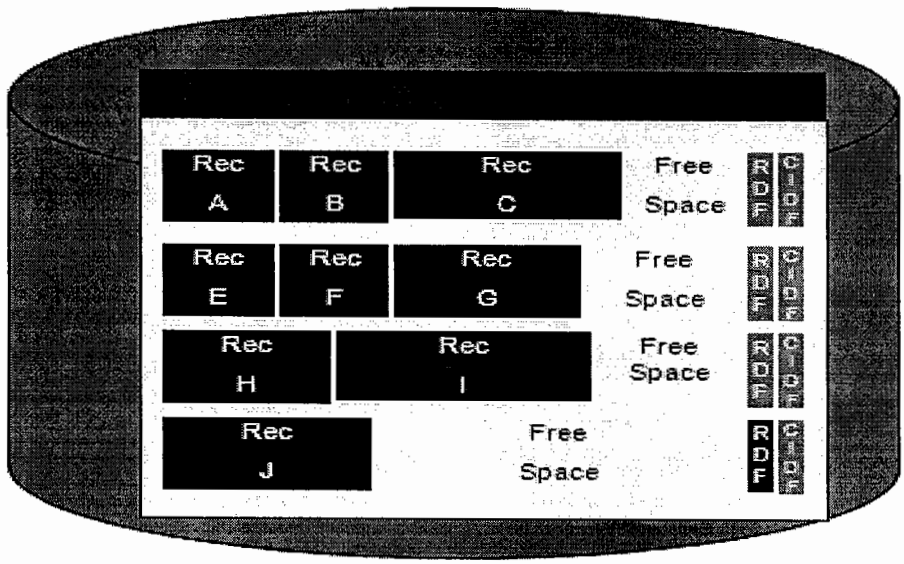
KSDS (KEY SEQUENCED DATASET):

- Records can be accessed sequentially or randomly or dynamically. Records can be accessed directly by supplying the KEY field.
- Has all the access methods Sequential, Random and Dynamic (SKIP Sequential).

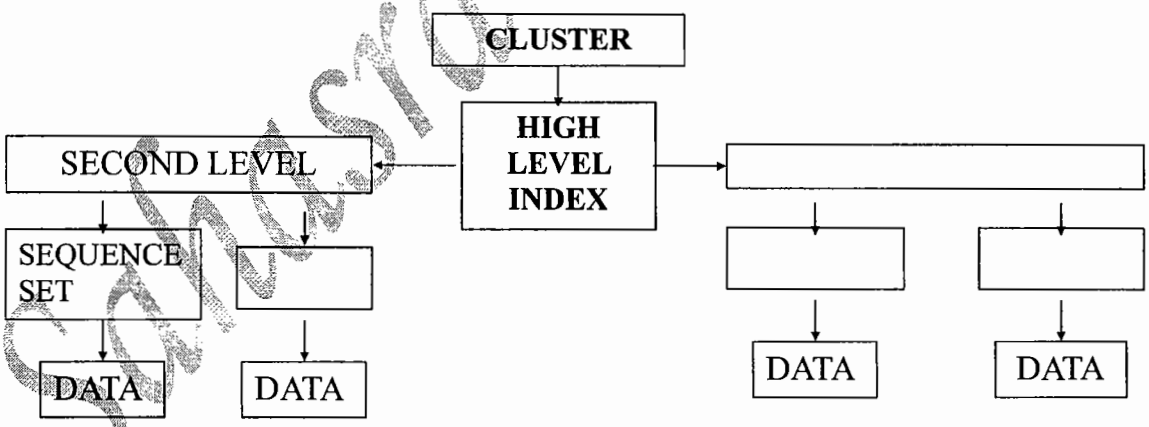
FEATURES OF KSDS CLUSTER:

1. It has all three components of VSAM (CLUSTER, INDEX and DATA).
   a. Index Component: It contains the key values which internally point to actual record present in data component.
   b. Data Component: It contains the actual data including KEY values.
2. Key sequenced Dataset.
3. Primary key should be
   a. Unique.
   b. Same position in every record.
   c. Is not split (has to be contiguous).
4. Records can be inserted in middle because of imbedded free space in control Interval.
5. Records can be deleted physically.
6. Primary key cannot be changed.
7. Allows Alternate Index.

KSDS Control Area



KSDS Structure



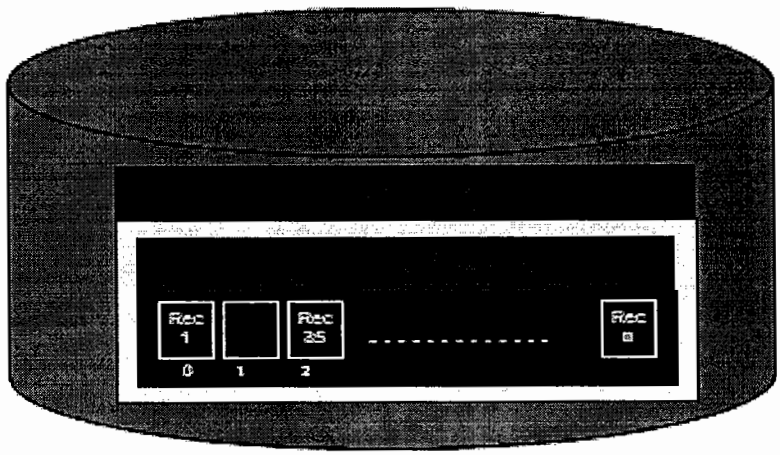
- Can have several levels of Indexes
- Lowest level of Index is called `Sequence Set'
- Index is organised as Inverted Binary Tree

RRDS (Relative Record Data Set):

STRUCTURE:

- It is used in conjunction with randomizing or hashing functions.
- A record can be accessed either sequentially in physical order OR directly by supplying the relative record number of the slot that contains the desired record.
- Processing can be combined in a program between physical sequential and direct.
- Relative Record Data Set,
  - Consists of a data component only.
  - Contains fixed-length records only.
  - Each record contained in a fixed-length slot Addressed by the relative record number of the slot in which it is stored starting with number 0.

RRDS Structure



Fixed-Length Records:

Record 1 in Slot 0.

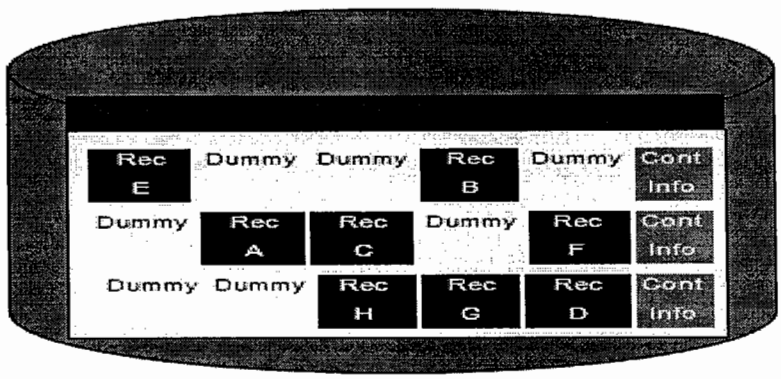
Dummy Record in Slot 1.

Record 35 in Slot 2. etc..

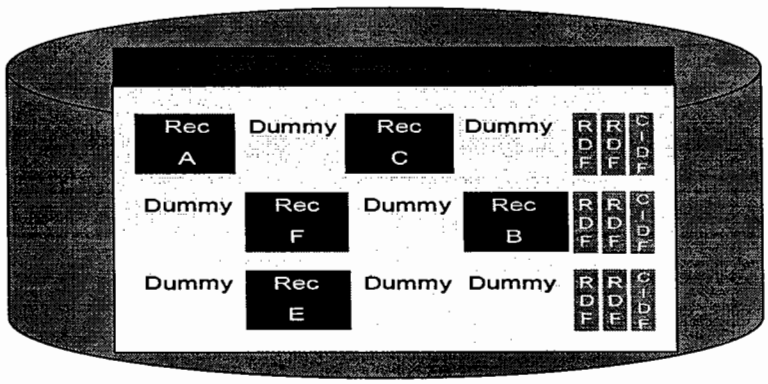
RRDS CONTROL INTERVAL:

- Each CI is filled with fixed-length slots which contain either
  - An active record OR
  - A dummy record
- Slots containing dummy records are available for use in the addition of new records to the file.

RRDS Control Interval



RRDS Control Area



RRDS LOADING:

- Records can be loaded either sequentially or directly.
- It is more common to load directly.
- Records are loaded using WRITE statement that specifies the relative record number of the slot into which the record is to be placed.
- VSAM
  - Locates the desired slot.
  - Verifies that the slot does not contain an active record.
  - Write the record into that slot.
- Free space that exists at the end of each control interval cannot be reused.
- There may be available space after the load has been completed.
  - There are dummy records into which no active record has been loaded.
- More space is allocated in an RRDS than other types of VSAM data sets.

RRDS RETRIEVAL:

- For direct processing VSAM
  - The same hashing routine is used to determine the slot number of the record needed.
- The program should then check to make sure that the record key is the correct one.

RRDS UPDATING:

- RRDS records can be updated while processing the file either sequentially or directly.
- VSAM performs a deletion by converting the slot into a dummy record.
- VSAM performs an update by finding a dummy record an replacing it with a new active record.

LDS (Linear Data Set**):**

- When we perform any SQL operations they are internally stored in LDS files.

VSAM dataset choice

	KSDS	ESDS	RRDS
Less DASD		✓	
On-line Direct Access	✓		✓
Batch Sequential		✓	
Alternate Index	✓		
Static data			✓
Ease of programming and maintenance	✓		

COMMON PARAMETERS USED FOR DEFINING A CLUSTER:

1. **Job Card:**
   This parameter is used to identify the job name and job related parameters.
2. **Exec:**
   This statement is used to identify the step name and program name. And here this Exec card invokes Access Method Services from IDCAMS utility.
3. **Sysprint:**
   This is used to print the error messages in the spool.
4. **Sysin:**
   This parameter is used to indicate the in-stream data and for defining a cluster every functional command and its relevant parameters should define in sysin only and those are as follows.

- **DEFINE CLUSTER:**
  This command is used to create and name a VSAM Cluster.

- **NAME:**
  It is used to specify name the VSAM cluster.
  Syntax: DEFINE CLUSTER(NAME(USERID.SAHASRA.KSDS.CLUST))

- **CYLINDERS or TRACKS:**
  This is used to provide space for the VSAM Dataset created.
  It is advisable to use Cylinders because it ensures a CA size of one cylinder.
  Syntax: CYLINDERS(07,03) (or) TRACKS(07,03)
  Where 07 → primary quantity, 03 →secondary quantity.

- **VOLUMES:**
  It gives the volume serial number where the memory allocated for defined cluster.
  Syntax: VOLUMES(S7SYS1)

- **CONTROL INTERVAL SIZE:**
  It specifies the size of the control interval and it should be the multiples of 512.
  Syntax: CISZ(N*512) Where N can be a value from 1 to 16.

- **RECORDSIZE:**
  This provides AMS with Average and MAX Record lengths for a Dataset.
  Syntax: RECORDSIZE(AVERAGE LENGTH,MAX LENGTH)

- **Fixed Block:**
  If both Average Length and Max Length are same we call them as FIXED BLOCK.
  E.g.: RECORDSIZE(80,80)

- **Variable Block:**
  If both Average Length and Max Length are different we call them as Varaiable Block.
  E.g.: RECORDSIZE(80,100)

- **Spanned Records & Nonspanned:**
  When a Recordsize is larger than the control interval size, the record must be contained in more than one control interval. Such a record is called as SPANNED RECORD.
  Note:
  - ✓ Freespace left by spanned record in a C.I cannot be used by another record.
  - ✓ If the max record length is more than CI size specified, allocation will fail if we don't specify spanned parameter during cluster creation. By default it will be Nonspanned.

- **FREESPACE:**
  This parameter is coded only for KSDS Cluster and this is used to provide freespace in CI and CA. Freespace will be used to add the records to CA / CI in future.
  Syntax: FREESPACE(CI%,CA%), **E.g.**: FREESPACE(10,10)

Values of freespace are coded as percentage and by default the values of freespace are (0,0).

- **KEYS:**
  This parameter gives the length of the KSDS key and its offset (starting position 1) from the beginning of record.
  Syntax: KEYS(KEY LEGNTH , OFFSET)
- **DATASET TYPE PARAMETER:**
  This parameter specifies whether the dataset is INDEXED(KSDS), NONINDEXED(ESDS), NEMBERED(RRDS) OR LINEAR(LDS).
  Note: INDEXED is the default.

- **REUSE PARAMETER:**
  This specifies the cluster can be opened again and again as reusable cluster, i.e., when we open the dataset in output mode in COBOL program, all the records in the dataset are logically deleted. NO REUSE is the Default option.
  Note: A Cluster cannot be REUSED if an alternate index is built on it.

- **SHAREOPTIONS:**
  This Parameter specifies how a VSAM dataset can be shared across the regions and across the system.
  Syntax: SHAREOPTIONS(cross-region,cross-system)
  Default Option: (1,3)

Possible values for cross-region & cross-system diagram:

TYPE	VALUE	MEANING
CR	1	READ and WRITE Integrity. Any number of jobs can read the dataset OR only one job can write to the dataset.
CR	2	ONLY WRITE Integrity. Any number of jobs can read the dataset AND one job can write to the dataset.
CR/CS	3	NO Integrity. File is fully shared. It is programmer responsibility to take proper lock over the file before use. Default value for CS.
CS	4	Same as 3 but additionally forces a buffer refresh for each random access.

- **PASSWORD PROTECTION:**
  VSAM datasets can be password protected at four different levels.
  1. READPW Provides read only.
  2. UPDATEPW Records can be read, updated, added or deleted.
  3. CONTROLPW Provides the access capabilities of READPW and UPDATEPW.
  4. MASTERPW All the above operations and the authority to delete the dataset.

  Note: RACF ignores VSAM passwords and imposes its own security and for all VSAM datasets RACF security is sufficient.
- **INDEX COMPONENT:**
  It contains the key values which internally point to actual record present in data component.
- **DATA COMPONENT:** It contains the actual data including KEY values.

CLUSTER CREATION FOR ALL VSAM DATASETS

ESDS Cluster Creation JCL:

```
//JOBCARD
//STEP1     EXEC   PGM=IDCAMS
//SYSPRINT  DD     SYSOUT=*
//SYSOUT    DD     SYSOUT=*
//SYSIN     DD     *
        DEFINE CLUSTER(NAME(USERID.SAHASRA.ESDS.CLUSTER) -
               VOLUME(VOLUME NAME)                       -
               CYLINDER(5,2)                             -
               CISZ(4096)                                -
               RECORDSIZE(40,40)                         -
               NONINDEXED)                               -
               DATA(NAME(USERID.SAHASRA.ESDS.DATA))
/*
//
```

KSDS Cluster Creation JCL:

```
//JOBCARD
//STEP1     EXEC   PGM=IDCAMS
//SYSPRINT  DD     SYSOUT=*
//SYSOUT    DD     SYSOUT=*
//SYSIN     DD     *
        DEFINE CLUSTER(NAME(USERID.SAHASRA.KSDS.CLUSTER) -
               VOLUME(VOLUME NAME)                       -
               CYLINDER(3,2)                             -
               RECORDSIZE(40,40)                         -
               CISZ(4096)                                -
               FREESPACE(10,20)                          -
               KEYS(3,0)                                 -
               INDEXED)                                  -
               DATA(NAME(USERID.SAHASRA.KSDS.DATA))      -
               INDEX(NAME(USERID.SAHASRA.KSDS.INDEX))
/*
//
```

RRDS Cluster Creation JCL:

```
//JOBCARD
//STEP1     EXEC   PGM=IDCAMS
//SYSPRINT  DD     SYSOUT=*
//SYSOUT    DD     SYSOUT=*
//SYSIN     DD     *
        DEFINE CLUSTER(NAME(USERID.SAHASRA.RRDS.CLUSTER) -
               VOLUME(VOLUME NAME)                       -
               CYLINDER(5,2)                             -
```

```
                CISZ(4096)                                      -
                RECORDSIZE(40,40)                               -
                NUMBERED)                                       -
                DATA(NAME(USERID.SAHASRA.RRDS.DATA))
    /*
    //
```

LDS Cluster Creation JCL:

```
    //JOBCARD
    //STEP1     EXEC  PGM=IDCAMS
    //SYSPRINT  DD    SYSOUT=*
    //SYSOUT    DD    SYSOUT=*
    //SYSIN     DD    *
        DEFINE CLUSTER(NAME(USERID.SAHASRA.LDS.CLUSTER)  -
                VOLUME(VOLUME NAME)                      -
                CYLINDER(5,2)                            -
                CISZ(4096)                               -
                RECORDSIZE(40,40)                        -
                LINEAR)                                  -
                DATA(NAME(USERID.SAHASRA.LDS.DATA))
    /*
    //
```

PROGRAMS ON VSAM:

P1. Write a Program to Insert Records in ESDS Cluster.

```
    IDENTIFICATION DIVISION.
    PROGRAM-ID. ESDSWRI.
    ENVIRONMENT DIVISION.
    INPUT-OUTPUT SECTION.
    FILE-CONTROL.
        SELECT CUSTFILE ASSIGN TO AS-CUSTDD
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL
        FILE   STATUS  IS   WS-STAT.
    DATA DIVISION.
    FILE SECTION.
    FD CUSTFILE.
    01    CUST-REC.
        05    CUST-ID      PIC   9(5).
        05    CUST-NAME    PIC   X(15).
        05    CUST-ADDR    PIC   X(20).
    WORKING-STORAGE SECTION.
    01    WS-EOF       PIC   X      VALUE 'Y'.
        88    NOT-WS-EOF  VALUE 'N'.
    01    WS-STAT      PIC   X(2)   VALUE SPACES.
    PROCEDURE DIVISION.
```

```
A100-MAIN-PARA.
        PERFORM      B100-OPEN-PARA      THRU   B100-OPEN-EXIT.
        PERFORM B200-PROCESS-PARA THRU B200-PROCESS-EXIT UNTIL NOT-WS-EOF.
        PERFORM      C100-CLOSE-PARA     THRU   C100-CLOSE-EXIT.
        STOP RUN.
B100-OPEN-PARA.
        OPEN OUTPUT CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE OPENED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN B100-OPEN-PARA'
        END-IF.
B100-OPEN-EXIT.
        EXIT.
B200-PROCESS-PARA.
        ACCEPT          CUST-ID.
        ACCEPT          CUST-NAME.
        ACCEPT          CUST-ADDR.
        WRITE CUST-REC.
        IF      WS-STAT = 00
                DISPLAY 'RECORD INSERTED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN B200-PROCESS-PARA'
        END-IF.
        ACCEPT WS-EOF.
B200-PROCESS-EXIT.
        EXIT.
C100-CLOSE-PARA.
        CLOSE CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE CLOSED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN C100-CLOSE-PARA'
        END-IF.
C100-CLOSE-EXIT.
        EXIT.
```

Q) By seeing the COBOL program itself how can I identify whether it is PS file or ESDS file?

Ans) By prefixing AS-DDNAME. In execution JCL, we only mention DD name, not AS-DDNAME.

➔ **EXECUTION JCL for the VSAM ESDS Write file program.**

```
//JOBCARD
//STEP1         EXEC    PGM=ESDSWRI
//STEPLIB       DD      DSN=PDS-LOAD-MODULE,DISP=SHR
//CUSTDD        DD      DSN=USERID.SAHASRA.ESDS.CLUSTER,DISP=SHR
```

```
//SYSPRINT    DD      SYSOUT=*
//SYSOUT      DD      SYSOUT=*
//SYSIN       DD      *
            ..............................
            ...........VALUES.............
            ..............................
/*
//
```

P2. Write a Program to Read the Particular Record in ESDS Cluster.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ESDSREAD.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT CUSTFILE ASSIGN TO AS-CUSTDD
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL
        FILE    STATUS  IS      WS-STAT.
DATA DIVISION.
FILE SECTION.
FD CUSTFILE.
01      CUST-REC.
        05      CUST-ID         PIC     9(5).
        05      CUST-NAME       PIC     X(15).
        05      CUST-ADDR       PIC     X(20).
WORKING-STORAGE SECTION.
01      WS-EOF          PIC     X       VALUE 'Y'.
        88      NOT-WS-EOF  VALUE 'N'.
77      WS-STAT         PIC     X(2)    VALUE SPACES.
77      WS-CUST-ID      PIC     9(5)    VALUE ZEROS.
PROCEDURE DIVISION.
A100-MAIN-PARA.
        PERFORM     B100-OPEN-PARA      THRU    B100-OPEN-EXIT.
        PERFORM B200-PROCESS-PARA THRU B200-PROCESS-EXIT UNTIL NOT-WS-EOF.
        PERFORM     C100-CLOSE-PARA     THRU    C100-CLOSE-EXIT.
        STOP RUN.
B100-OPEN-PARA.
        OPEN INPUT CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE OPENED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN B100-OPEN-PARA'
        END-IF.
B100-OPEN-EXIT.
```

```
        EXIT.
B200-PROCESS-PARA.
        ACCEPT WS-CUST-ID.
        READ   CUSTFILE
        AT END
                MOVE 'N' TO WS-EOF
                DISPLAY 'RECORD NOT FOUND'.
        NOT AT END
                IF      WS-CUST-ID = CUST-ID
                PERFORM D100-DISPLAY-PARA   THRU   D100-DISPLAY-EXIT
                END-IF
        END-READ.
B200-PROCESS-EXIT.
        EXIT.
D100-DISPLAY-PARA.
        DISPLAY   CUST-REC.
D100-DISPLAY-EXIT.
        EXIT.
C100-CLOSE-PARA.
        CLOSE CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE CLOSED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN C100-CLOSE-PARA'
        END-IF.
C100-CLOSE-EXIT.
        EXIT.
```

→ **EXECUTION JCL for the VSAM ESDS Read file program.**

```
//JOBCARD
//STEP1     EXEC   PGM=ESDSREAD
//STEPLIB   DD     DSN=PDS-LOAD MODULE,DISP=SHR
//CUSTDD    DD     DSN=USERID.SAHASRA.ESDS.CLUSTER,DISP=SHR
//SYSPRINT  DD     SYSOUT=*
//SYSOUT    DD     SYSOUT=*
//SYSIN     DD     *
/*
//
```

→ To browse the VSAM files we should go to 'DITTO' Tool.

P3. WRITE FILE PROGRAM using KSDS RANDOM Method.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. KSDSWRI.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
```

```
FILE-CONTROL.
        SELECT CUSTFILE ASSIGN TO CUSTDD
        ORGANIZATION IS INDEXED
        ACCESS MODE IS RANDOM
        RECORD KEY IS CUST-ID
        FILE    STATUS  IS    WS-STAT.
DATA DIVISION.
FILE SECTION.
FD CUSTFILE.
01      CUST-REC.
        05      CUST-ID         PIC     9(5).
        05      CUST-NAME       PIC     X(15).
        05      CUST-ADDR       PIC     X(20).
WORKING-STORAGE SECTION.
01      WS-EOF          PIC     X       VALUE 'Y'.
        88      NOT-WS-EOF      VALUE 'N'.
77      WS-STAT         PIC     X(2)    VALUE ZEROS.
PROCEDURE DIVISION.
A100-MAIN-PARA.
        PERFORM       B100-OPEN-PARA     THRU   B100-OPEN-EXIT.
        PERFORM B200-PROCESS-PARA THRU B200-PROCESS-EXIT UNTIL NOT-WS-EOF.
        PERFORM       C100-CLOSE-PARA    THRU   C100-CLOSE-EXIT.
        STOP RUN.
B100-OPEN-PARA.
        OPEN OUTPUT CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE OPENED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN B100-OPEN-PARA'
        END-IF.
B100-OPEN-EXIT.
        EXIT.
B200-PROCESS-PARA.
        ACCEPT          CUST-ID.
        ACCEPT          CUST-NAME.
        ACCEPT          CUST-ADDR.
        WRITE CUST-REC      INVALID KEY   DISPLAY 'ERROR IN WRITING'.
        ACCEPT WS-EOF.
B200-PROCESS-EXIT.
        EXIT.
C100-CLOSE-PARA.
        CLOSE CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE CLOSED SUCCESSFULLY'
```

```
        ELSE
            DISPLAY 'ERROR OCCURRED IN C100-CLOSE-PARA'.
        END-IF.
C100-CLOSE-EXIT.
        EXIT.
```

➔ **EXECUTION JCL for the VSAM KSDS Write file program.**

```
//JOBCARD
//STEP1     EXEC  PGM=KSDSWRI
//STEPLIB   DD    DSN=PDS-LOAD-MODULE,DISP=SHR
//CUSTDD    DD    DSN= USERID.SAHASRA.KSDS.CLUSTER,DISP=SHR
//SYSPRINT  DD    SYSOUT=*
//SYSOUT    DD    SYSOUT=*
//SYSIN     DD    *
        ..............................
        ..........VALUES.........
        ..............................
/*
//
```

P4. READ FILE PROGRAM using KSDS RANDOM Method.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. KSDSREAD.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT CUSTFILE ASSIGN TO CUSTDD
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS RANDOM
        FILE    STATUS  IS    WS-STAT.
DATA DIVISION.
FILE SECTION.
FD CUSTFILE.
01      CUST-REC.
        05      CUST-ID         PIC     9(5).
        05      CUST-NAME       PIC     X(15).
        05      CUST-ADDR       PIC     X(20).
WORKING-STORAGE SECTION.
01      WS-CUST-ID    PIC     9(5).
01      WS-EOF        PIC     X       VALUE 'Y'.
        88      NOT-WS-EOF    VALUE 'N'.
77      WS-STAT       PIC     X(2)    VALUE ZEROS.
PROCEDURE DIVISION.
A100-MAIN-PARA.
        PERFORM     B100-OPEN-PARA     THRU   B100-OPEN-EXIT.
        PERFORM B200-PROCESS-PARA THRU B200-PROCESS-EXIT UNTIL NOT-WS-EOF.
```

```
        PERFORM     C100-CLOSE-PARA     THRU   C100-CLOSE-EXIT.
        STOP RUN.
B100-OPEN-PARA.
        OPEN INPUT CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE OPENED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN B100-OPEN-PARA'
        END-IF.
B100-OPEN-EXIT.
        EXIT.
B200-PROCESS-PARA.
        ACCEPT          WS-CUST-ID.
        MOVE            WS-CUST-ID TO CUST-ID.
        READ   CUST-REC
        INVALID KEY   DISPLAY         'RECORD NOT FOUND'
        NOT INVALID KEY
        PERFORM     D100-DISPLAY-PARA   THRU   D100-DISPLAY-EXIT
        ACCEPT          OPTION
        END-READ.
B200-PROCESS-EXIT.
        EXIT.
D100-DISPLAY-PARA.
        DISPLAY CUST-REC.
D100-DISPLAY-EXIT.
        EXIT.
C100-CLOSE-PARA.
        CLOSE CUSTFILE
        IF      WS-STAT = 00
                DISPLAY 'FILE CLOSED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN C100-CLOSE-PARA'
        END-IF.
C100-CLOSE-EXIT.
        EXIT.
```

➔ **EXECUTION JCL for the VSAM KSDS Random Read File Program.**

```
//JOBCARD
//STEP1       EXEC   PGM=KSDSREAD
//STEPLIB     DD     DSN=PDS-LOAD-MODULE,DISP=SHR
//CUSTDD      DD     DSN= USERID.SAHASRA.KSDS.CLUSTER,DISP=SHR
//SYSPRINT    DD     SYSOUT=*
//SYSOUT      DD     SYSOUT=*
//SYSIN       DD     *
Value 1
```

```
Option 1
Value 2
Option 2
 /*
 //
```

P5. READ FILE PROGRAM using KSDS DYNAMIC Method.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. KSDSDYM.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT CUSTFILE ASSIGN TO CUSTDD
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS DYNAMIC
        RECORD KEY IS CUST-ID
        FILE    STATUS  IS    WS-STAT.
DATA DIVISION.
FILE SECTION.
FD CUSTFILE.
01      CUST-REC.
        05      CUST-ID         PIC     9(5).
        05      CUST-NAME       PIC     X(15).
        05      CUST-ADDR       PIC     X(20).
WORKING-STORAGE SECTION.
01      WS-CUST-ID      PIC     9(5).
01      WS-EOF          PIC     X       VALUE 'Y'.
        88      NOT-WS-EOF      VALUE 'N'.
77      WS-STAT         PIC     X(2)    VALUE ZEROS.
PROCEDURE DIVISION.
A100-MAIN-PARA.
        PERFORM     B100-OPEN-PARA THRU B100-OPEN-EXIT.
        ACCEPT      WS-CUST-ID.
        MOVE WS-CUST-ID TO CUST-ID.
        START CUSTFILE
                INVALID KEY DISPLAY 'ERROR'.
        PERFORM B200-PROCESS-PARA THRU B200-PROCESS-EXIT UNTIL NOT-WS-EOF.
        PERFORM     C100-CLOSE-PARA     THRU   C100-CLOSE-EXIT.
        STOP RUN.
B100-OPEN-PARA.
        OPEN INPUT CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE OPENED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN B100-OPEN-PARA'
```

```
        END-IF.
B100-OPEN-EXIT.
        EXIT.
B200-PROCESS-PARA.
        READ CUST-REC NEXT RECORD
        AT END
                MOVE 'N' TO WE-EOF
        NOT AT END
                PERFORM     D100-DISPLAY-PARA  THRU  D100-DISPLAY-EXIT
        END-READ.
B200-PROCESS-EXIT.
        EXIT.
D100-DISPLAY-PARA.
        DISPLAY CUST-REC.
D100-DISPLAY-EXIT.
        EXIT.
C100-CLOSE-PARA.
        CLOSE CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE CLOSED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN C100-CLOSE-PARA'
        END-IF.
C100-CLOSE-EXIT.
        EXIT.
```

➔ **EXECUTION JCL for the VSAM KSDS Dynamic Read file program.**

```
//JOBCARD
//STEP1       EXEC   PGM=KSDSDYM
//STEPLIB     DD     DSN=PDS-LOAD-MODULE,DISP=SHR
//CUSTDD      DD     DSN=USERID.SAHASRA.KSDS.CLUSTER,DISP=SHR
//SYSPRINT    DD     SYSOUT=*
//SYSOUT      DD     SYSOUT=*
//SYSIN       DD     *
Value1
OPTION
/*
//
```

P6. Write a Program to Load Values in RRDS Cluster.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. RRDSWRI.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT CUSTFILE ASSIGN TO CUSTDD
```

```
        ORGANIZATION IS RELATIVE
        ACCESS MODE IS RANDOM
        RELATIVE KEY IS WS-CUST-ID
        FILE    STATUS  IS    WS-STAT.
DATA DIVISION.
FILE SECTION.
FD CUSTFILE.
01      CUST-REC.
        05      CUST-ID         PIC     9(5).
        05      CUST-NAME       PIC     X(15).
        05      CUST-ADDR       PIC     X(20).
WORKING-STORAGE SECTION.
01      WS-CUST-ID  PIC     9(5).
01      WS-EOF      PIC     X       VALUE 'Y'.
        88      NOT-WS-EOF  VALUE 'N'.
77      WS-STAT     PIC     X(2)    VALUE ZEROS.
PROCEDURE DIVISION.
A100-MAIN-PARA.
        PERFORM     B100-OPEN-PARA     THRU   B100-OPEN-EXIT.
        PERFORM B200-PROCESS-PARA THRU B200-PROCESS-EXIT UNTIL NOT-WS-EOF.
        PERFORM     C100-CLOSE-PARA    THRU   C100-CLOSE-EXIT.
        STOP RUN.
B100-OPEN-PARA.
        OPEN   OUTPUT   CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE OPENED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN B100-OPEN-PARA'
        END-IF.
B200-PROCESS-PARA.
        ACCEPT          WS-CUST-ID.
        ACCEPT          CUST-ID.
        ACCEPT          CUST-NAME.
        ACCEPT          CUST-ADDR.
        WRITE           CUST-REC   INVALID KEY   DISPLAY 'ERROR'.
        ACCEPT          WS-EOF.
B200-PROCESS-EXIT.
        EXIT.
C100-CLOSE-PARA.
        CLOSE CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE CLOSED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN C100-CLOSE-PARA'
```

```
            END-IF.
        C100-CLOSE-EXIT.
            EXIT.
```

➔ **EXECUTION JCL for the VSAM RRDS Write File Program.**

```
//JOBCARD
//STEP1     EXEC   PGM=RRDSWRI
//STEPLIB   DD     DSN=PDS-LOAD- MODULE,DISP=SHR
//CUSTDD    DD     DSN=USER.SAHASRA.RRDS.CLUSTER,DISP=SHR
//SYSPRINT  DD     SYSOUT=*
//SYSOUT    DD     SYSOUT=*
//SYSIN     DD     *
            ..............................
            .....VALUES.....
            ..............................
/*
//
```

Major Functionalities that can be performed using IDCAMS:

1. DEFINE
   a. GDG
   b. CLUSTER.
   c. AIX, BLDINDEX, PATH
2. DELETE
3. REPRO
4. LISTCAT
5. PRINT
6. ALTER
7. VERIFY

DELETE:

► DELETE is used to delete PS, VSAM datasets and GDG base.

```
//JOBCARD
//STEP1     EXEC   PGM=IDCAMS
//SYSPRINT  DD     SYSOUT=*
//SYSOUT    DD     SYSOUT=*
//SYSIN     DD     *
            DELETE(PS FILE NAME / VSAM File Name / GDG Base Name)
/*
//
```

E.g.:

```
//SYSIN       DD     *
            DELETE(FSS244.SAHASRA.CUSTFILE)
                        OR
            DELETE(FSS244.SAHASRA.KSDS.CLUSTER)
/*
//
```

Note: When we delete GDG Base all its associated generations will also be deleted. Similarly, when we delete base cluster then its associated Index and Data Component also gets deleted.

REPRO:

- REPRO is used to read an input dataset and load records into an output dataset. The possibilities to load records can be
  - PS-File to PS-File.
  - VSAM to VSAM.
  - VSAM to PS-File.
  - PS-File to VSAM.

Syntax 1: When the output dataset is already present we use following syntax.

```
//JOBCARD
//STEP1     EXEC   PGM=IDCAMS
//SYSPRINT  DD     SYSOUT=*
//SYSOUT    DD     SYSOUT=*
//SYSIN     DD     *
        REPRO                                    -
        INDATASET(INPUT DATASET NAME)            -
        OUTDATASET(OUTPUTDATASET NAME)
/*
//
```

E.g.: **VSAM to VSAM**

```
//SYSIN     DD     *
        REPRO                                    -
        INDATASET(FSS244.SAHASRA.KSDS)           -
        OUTDATASET(FSS244.SAHASRA.KSDS.BKP)
/*
//
```

Syntax 2: We use following syntax when the output data is not cataloged or present in system:

```
//JOBCARD
//STEP1     EXEC PGM=IDCAMS
//DD1       DD     DSN=INPUT FILE NAME,DISP=SHR
//DD2       DD     DSN=OUTPUT PS FILE NAME,
//                 DISP=(NEW,CATLG,DELETE),
//                 SPACE=(TRK,(1,1),RLSE),
//                 DCB=(DSORG=PS,LRECL=40,BLKSIZE=800,RECFM=FB)
//SYSPRINT  DD     SYSOUT=*
//SYSOUT    DD     SYSOUT=*
//SYSIN     DD     *
        REPRO                                    -
        INFILE(Input file DDNAME)                -
        OUTFILE(Output File DDNAME)
/*
//
```

E.g.: VSAM TO PS-FILE

```
//JOBCARD
//STEP1     EXEC PGM=IDCAMS
//DD1       DD   DSN=FSS244.SAHASRA.KSDS,DISP=SHR
//DD2       DD   DSN=FSS244.SAHASRA.PSFILE,
//               DISP=(NEW,CATLG,DELETE),
//               SPACE=(TRK,(1,1),RLSE),
//               DCB=(DSORG=PS,LRECL=40,BLKSIZE=800,RECFM=FB)
//SYSPRINT  DD   SYSOUT=*
//SYSOUT    DD   SYSOUT=*
//SYSIN     DD   *
        REPRO       -
        INFILE(DD1) -
        OUTFILE(DD2)
/*
//
```

PS-FILE TO VSAM:

Note: When we copy PS file to KSDS file, care should be taken such that the records in PS files should be mandatory in sorted order on the key field and should not contain any duplicate values. **E.g.**: CUST-ID

- If this weren't so, REPRO will throw sequence error or duplicate index error. So, it is better to Load a KSDS with two step process.

1. In the First step sort the input file on key field and eliminate if there are any duplicate records using External SORT.
2. In the second step load the sorted file to the KSDS Cluster.

JCL containing two step processes:

```
//JOBCARD
//STEP1     EXEC  PGM=SORT
//SORTIN    DD    DSN=FSS244.SAHASRA.CUSTFILE,DISP=SHR
//SORTOUT   DD    DSN=FSS244.SAHASRA.CUSTFILE.SORT,
//               DISP=(NEW,CATLG,DELETE),
//               SPACE=(TRKS,(1,1),RLSE),
//               DCB=(DSORG=PS,LRECL=40,BLKSIZE=400,RECFM=FB)
//SYSPRINT  DD    SYSOUT=*
//SYSOUT    DD    SYSOUT=*
//SYSIN     DD    *
  SORT FIELDS =(1,3,CH,ASC),EQUALS,
  SUM FIELDS=NONE
/*
//STEP2     EXEC  PGM=IDCAMS
//DD1       DD    DSN= FSS244.SAHASRA.CUSTFILE.SORT,DISP=SHR
//DD2       DD    DSN=FSS244.SAHASRA.KSDS.CLUST1,DISP=OLD
//SYSPRINT  DD    SYSOUT=*
//SYSOUT    DD    SYSOUT=*
```

```
//SYSIN       DD    *
        REPRO       -
        INFILE(DD1) -
        OUTFILE(DD2)
/*
//
```

We can selectively copy records from input file by specifying following options during loading:

```
//SYSIN    DD *
        REPRO       -
        INFILE(DD1) -
        OUTFILE(DD2)
        SKIP(X)     → No of records needs to skip from input file
        COUNT(Y)    → No of records needs to be copied to output file.
/*
```

- If Skip is not coded then the default is SKIP(0).

E.g.: JCL to copy three records(29,30,31) to output file:

```
//SYSIN       DD *
        REPRO       -
        INFILE(DD1) -
        OUTFILE(DD2)
        SKIP(28)    → SKIPS initial 28 records from input file.
        COUNT(3)    → Copies 3 records from 29 to output file.
/*
```

- REPRO is also used for merging or concatenating the datasets when we merge the dataset the output dataset should be non-empty dataset.

REPLACE:

- This optional Parameter is used to merge records into a VSAM File. This has meaning only when the target data set already has records. It eliminates the records with duplicate primary keys from the input file.

PRINT:

- PRINT is used to print the records in a file. We can print the records in different formats like,
  1. Character
  2. Hex/Packed Decimal
  3. Dump (Combination of CHAR and Hexa Decimal)

Syntax:

```
//JOBCARD
//STEP1       EXEC  PGM=IDCAMS
//SYSPRINT    DD    SYSOUT=*
//SYSOUT      DD    SYSOUT=*
//SYSIN       DD    *
        PRINT                       -
        INDATASET(DATASET NAME)     -
        CHAR/HEX/DUMP               -
```

```
        SKIP(X)                          -
        COUNT(Y)
/*
//
```

JCL to print the record 29, 30, 31 in character format.

```
//JOBCARD
//STEP1      EXEC   PGM=IDCAMS
//SYSPRINT   DD     SYSOUT=*
//SYSOUT     DD     SYSOUT=*
//SYSIN      DD     *
        PRINT                            -
        INDATASET(DATASET NAME)          -
        CHARACTER                        -
        SKIP(28)                         -
        COUNT(3)
/*
//
```

JCL to check whether the file is empty or not.

```
//JOBCARD
//STEP1      EXEC   PGM=IDCAMS
//SYSPRINT   DD     SYSOUT=*
//SYSOUT     DD     SYSOUT=*
//SYSIN      DD     *
        PRINT                            -
        INDATASET(DATASET NAME)          -
        CHARACTER                        -
        COUNT(1)
/*
//
```

Note: If return code is 04 then that file is empty else it contains records.

LISTCAT:

- LISTCAT is used to list the catalog information of VSAM files and GDG base.

Syntax: //JOBCARD

```
//STEP1      EXEC   PGM=IDCAMS
//SYSPRINT   DD     SYSOUT=*
//SYSOUT     DD     SYSOUT=*
//SYSIN      DD     *
        LISTCAT                          -
        ENTRIES(VSAM FILENAME)           -
        ALL / Volume
                (Or)
        LISTCAT                          -
        ENTRIES(GDG BASE NAME)           -
```

```
        GDG                     –
        ALL / Limit
    /*
    //
```

ALTER: ALTER is used to → Rename VSAM files.
→ To increase the GDG limit.
→ To add volumes/remove volumes/ increasing free space.

Syntax:

```
    //JOBCARD
    //STEP1     EXEC   PGM=IDCAMS
    //SYSPRINT  DD     SYSOUT=*
    //SYSOUT    DD     SYSOUT=*
    //SYSIN     DD     *
        ALTER                   –
        OLD FILE NAME           –
        NEWNAME(NEW FILE NAME)
    /*
    //
```

E.g. 1:

```
    //JOBCARD
    //STEP1     EXEC   PGM=IDCAMS
    //SYSPRINT  DD     SYSOUT=*
    //SYSOUT    DD     SYSOUT=*
    //SYSIN     DD     *
        ALTER                   –
        FSS244.SAHASRA.KSDS1    –
        NEWNAME(FSS244.SAHASRA.KSDS2)
    /*
    //
```

E.g. 2:

```
    //JOBCARD
    //STEP1     EXEC   PGM=IDCAMS
    //SYSPRINT  DD     SYSOUT=*
    //SYSOUT    DD     SYSOUT=*
    //SYSIN     DD     *
        ALTER                   –
        GDG BASE NAME           –
        LIMIT(10)
    /*
```

E.g. 3:

```
    //JOBCARD
    //STEP1     EXEC   PGM=IDCAMS
    //SYSPRINT  DD     SYSOUT=*
    //SYSOUT    DD     SYSOUT=*
    //SYSIN     DD     *
        ALTER                                   –
        VSAM KSDS DATA COMPONENT                –
```

```
        ADD VOLUMES( VOLUME NAME)          –
        REMOVE VOLUMES(VOLUME NAME)        –
        FREESPACE(20 20)
    /*
```

Limitations of Alter: Using ALTER we cannot change Record Size and Record Key Parameter.

VERIFY:

- VERIFY is used to reset the index of KSDS file to starting position.

Syntax:

```
    //JOBCARD
    //STEP1      EXEC   PGM=IDCAMS
    //SYSPRINT   DD     SYSOUT=*
    //SYSOUT     DD     SYSOUT=*
    //SYSIN      DD     *
          VERIFY       –
          (KSDS CLUSTER NAME)
    /*
    //
```

- We get file-status 97 when we are trying to open a VSAM KSDS file which is already in opened state. To overcome from this, we go for verify option in IDCAMS.

AIX (ALTERNATE INDEX)

1. Define Cluster (KSDS Cluster).
2. Define AIX.
3. Define PATH.
4. BLDINDEX.

Define AIX Cluster:

```
    //JOBCARD
    //STEP1      EXEC   PGM=IDCAMS
    //SYSPRINT   DD     SYSOUT=*
    //SYSIN      DD     *
          DEFINE AIX(NAME(USERID.NAME.AIX)          –
                RELATE(KSDS FILENAME)               –
                VOLUMES(VOLUME NAME)                –
                CYLINDERS(3,2)                      –
                RECORDSIZE(40,40)                   –
                CISZ(4096)                          –
                FREESPACE(10,20)                    –
                KEYS(LENGTH,OFFSET)                 –
                NONUNIQUE KEY/ UNIQUE KEY           –
                UPGRADE                             –
                INDEXED)                            –
                INDEX(NAME(USERID.NAME.AIX.INDEX))  –
                DATA(NAME(USERID.NAME.AIX.DATA))
    /*
    //
```

- RECORDSIZE(AVERAGE,MAX)
- Alternate Key Size + Key Field Size + 5.
- Non Unique key means duplicates like CustName's.
- Unique Key no duplication like SS NO.

Define PATH Program:

```
//JOBCARD
//STEP1      EXEC   PGM=IDCAMS
//SYSPRINT   DD     SYSOUT=*
//SYSOUT     DD     SYSOUT=*
//SYSIN      DD     *
       DEFINE PATH(NAME(FSS244.SAHASRA.KSDS.AIX.PATH)      –
              PATHENTRY(FSS244.SAHASRA.KSDS.AIX)           –
              UPDATE)
/*
//
```

BLDINDEX PROGRAM:

```
//JOBCARD
//STEP1      EXEC   PGM=IDCAMS
//SYSPRINT   DD     SYSOUT=*
//SYSIN      DD     *
       BLDINDEX INDATASET(FSS244.SAHASRA.KSDS) –
       OUTDATASET(FSS244.SAHASRA.KSDS.AIX)
/*
//
```

Write a Read file program using AIX Read.

```
IDENTIFICATION  DIVISION.
PROGRAM-ID. AIXREAD.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.                          {NOTE: DDNAME Must Have 8 Charters}
       SELECT CUSTFILE ASSIGN TO CUSTREDD
       ORGANIZATION IS INDEXED
       ACCESS MODE IS RANDOM
       RECORD KEY IS CUST-ID
       ALTERNATE KEY IS CUST-NAME WITH/WITHOUT DUPLICATES
       FILE STATUS IS WS-STAT.
DATA DIVISION.
FILE SECTION.
FD CUSTFILE.
01     CUST-REC.
       05     CUST-ID      PIC    9(5).
       05     CUST-NAME    PIC    X(15).
       05     CUST-ADDR    PIC    X(20).
```

```
WORKING-STORAGE SECTION.
01      WS-CUST-NAME        PIC     X(15).
01      WS-EOF          PIC     X       VALUE 'Y'.
        88      NOT-WS-EOF  VALUE 'N'.
77      WS-STAT         PIC     9(2)    VALUE ZEROS.
PROCEDURE DIVISION.
A100-MAIN-PARA.
        PERFORM   B100-OPEN-PARA   THRU    B100-OPEN-EXIT.
        PERFORM  B200-PROCESS-PARA  THRU B200-PROCESS-EXIT  UNTIL NOT-WS-EOF.
        PERFORM  C100-CLOSE-PARA   THRU    C100-CLOSE-EXIT .
        STOP RUN.
B100-OPEN-PARA.
        OPEN INPUT CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE OPENED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED IN B100-OPEN-PARA'.
B100-OPEN-EXIT.
        EXIT.
B200-PROCESS-PARA.
        ACCEPT    WS-CUST-NAME.
        MOVE    WS-CUST-NAME      TO      CUST-NAME.
        READ    CUSTFILE      KEY     IS      CUST-NAME
        INVALID KEY
                MOVE 100 TO RETURN-CODE
                DISPLAY 'STATUS CODE:' WS-STAT
        NOT INVALID KEY
                PERFROM   D100-DISPLAY-PARA   THRU   D100-DISPLAY-EXIT
        END-READ.
        ACCEPT    WS-EOF.
B200-PROCESS-EXIT.
        EXIT.
D100-DISPLAY-PARA.
        DISPLAY CUST-REC.
D100-DISPLAY-EXIT.
        EXIT.
C100-CLOSE-PARA.
        CLOSE CUSTFILE.
        IF      WS-STAT = 00
                DISPLAY 'FILE CLOSED SUCCESSFULLY'
        ELSE
                DISPLAY 'ERROR OCCURRED  IN  C100-CLOSE-PARA'
C100-CLOSE-EXIT.
        EXIT.
```

➔ **EXECUTION JCL:**

```
//JOBCARD
//STEP1       EXEC   PGM=AIXPGM
//STEPLIB     DD     DSN=FSS244.SAHASRA.LOADLIB,DISP=SHR
/SYSPRINT     DD     SYSOUT=*
//SYSOUT      DD     SYSOUT=*
//CUSTREDD    DD     DSN=FSS244.SAHASRA.KSDS,DISP=SHR
//CUSTRED1    DD     DSN=FSS244.SAHASRA.KSDS.AIX.PATH,DISP=SHR
//SYSIN       DD     *
SAHASRA
Y
/*
//
```

SAHASRA INFOTECH
&
CONSULTING SERVICES

D B 2
(Data Base 2)

www.sahasrainfotech.co.in

DATABASE - 2

- The collection of inter related data is called as Database.
- Database is a central pool of data that can be accessed by many users.

ADVANTAGES OF DATABASE:

- We can store large volumes of data in database.
- Accessing of data present in database is easy when compared with accessing of data stored in files.
- Database provides the data security, integrity and concurrency.
- We can decrease the data redundancy (duplication of data).
- Advanced Recovery and Restart in DBMS system, the Recovery and Restart can be handled automatically.

E.g.: If a transaction terminated abnormally while updating a database, the DBMS will return back to initial state i.e. DBMS will make sure that updating of database only when the entire transaction successful.

DATABASE SYSTEM COMPONENTS: There are four system components

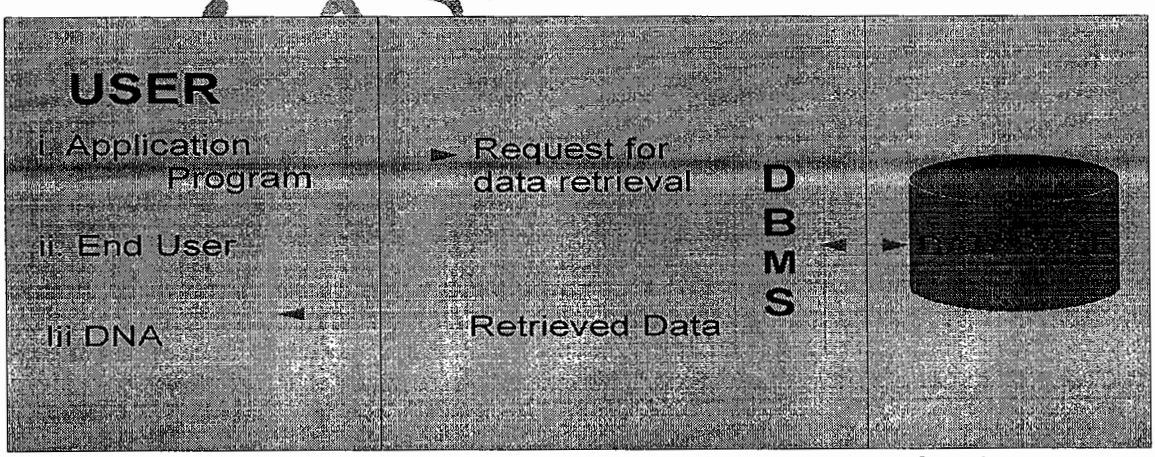
1. Hardware.
2. Data.
3. User.
4. Database (Managed Software).

Hardware: The hardware is the configuration of CPU, DASD and I/O devices etc. These all are required to install database management software and Run.

Data: The information stored in database is called Data.

User: Users can be Application programmers, DB2-DBA's and end users.

DATABASE MANAGEMENT SYSTEM or DBMS:



- The database management system (DBMS) will act as an interface between user and data.
- DBMS is responsible for storing the data and then making it available to the user for retrieval and update upon request.

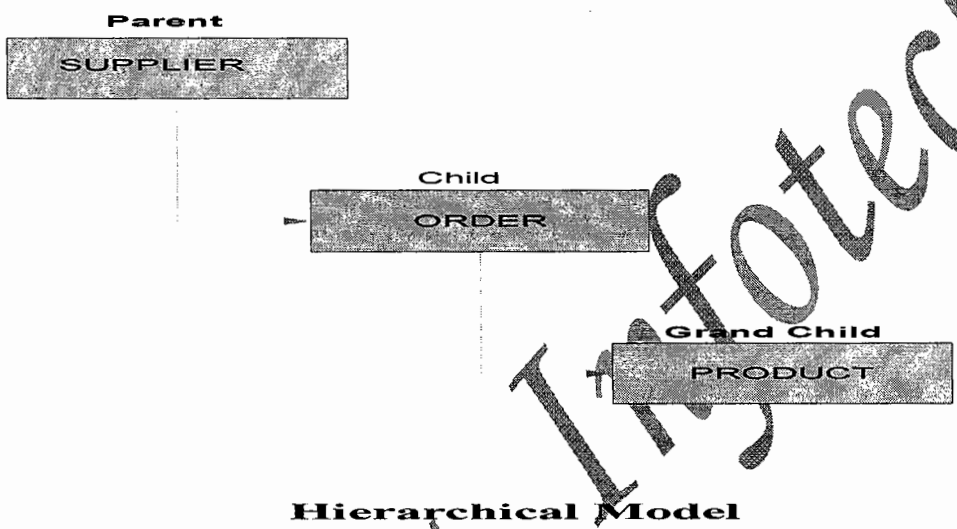
DBMS DATA MODELS:

DBMS is categorized based on in which manner they present the data to the user. There are four types of DBMS data models.

1. Hierarchical.
2. Network.
3. Relational.

Hierarchical Model:

- The structure of the hierarchical model is a tree structure.
- The data records are typically connected with embedded pointer.

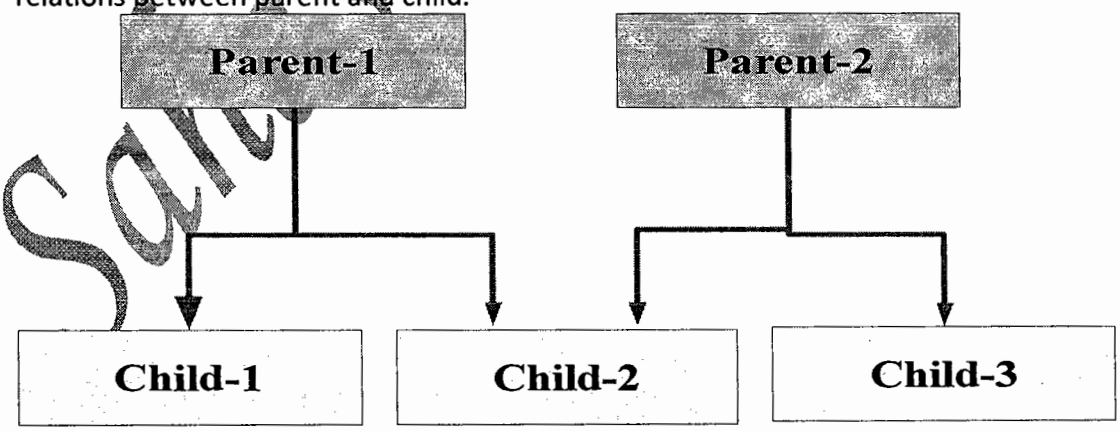


Hierarchical Model

Eg: IMS DB

Network Model:

- The structure of the Network model is same like hierarchical but having many to many relations between parent and child.



Network Model

E.g.: IDMS

Relational Model:

- In Relational Model Database management system, the data can be represented in the form of tabels.
- In Relational Model DBMS provides the content of database to the user as collection of tables.
- User no needs to specify the navigation to get the data, instead only user can request for data by naming the table.
  E.g.: DB2, ORACLE, SYBASE etc....

DB2:
DB2 is the Relational Database Management System for the MVS Operating System.
It was released by IBM in 1983.

DB2 DIRECTORY:
It contains the information about the Database objects for internal use of DB2.

DB2 CATALOG:
DB2 Catalog contains the information about the user defined database and its objects. It can be used by DB2 users and DB2.

- Nearly it contains 54 system tables.

E.g.:

SYSIBM.SYSTABLE;
It contains all the information of tables created in the DB2 database.

SELECT * FROM SYSIBM.SYSCOLUMNS;
It contains all the information of the columns created in DB2 Database.

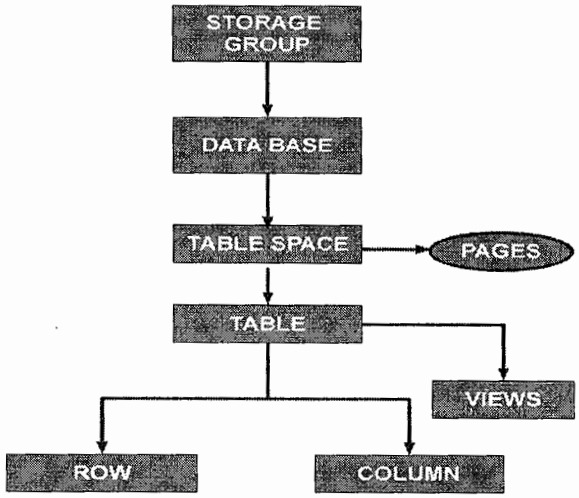
DB2 OBJECTS:
An object is anything you can create and manipulate with SQL. There are two types of objects.

1) System Objects.
2) Data Objects.

System Objects: Objects that are controlled and used by DB2.
E.g.: DB2 Directory, DB2 Catalog, Buffer pools, Locks etc.

Data objects: Objects that are created and used by users.
E.g.: Tables, Indexes, views, Table spaces, Databases, Storage groups.

STRUCTURE OF DATABASE



HIERARCHY OF DB2 OBJECTS

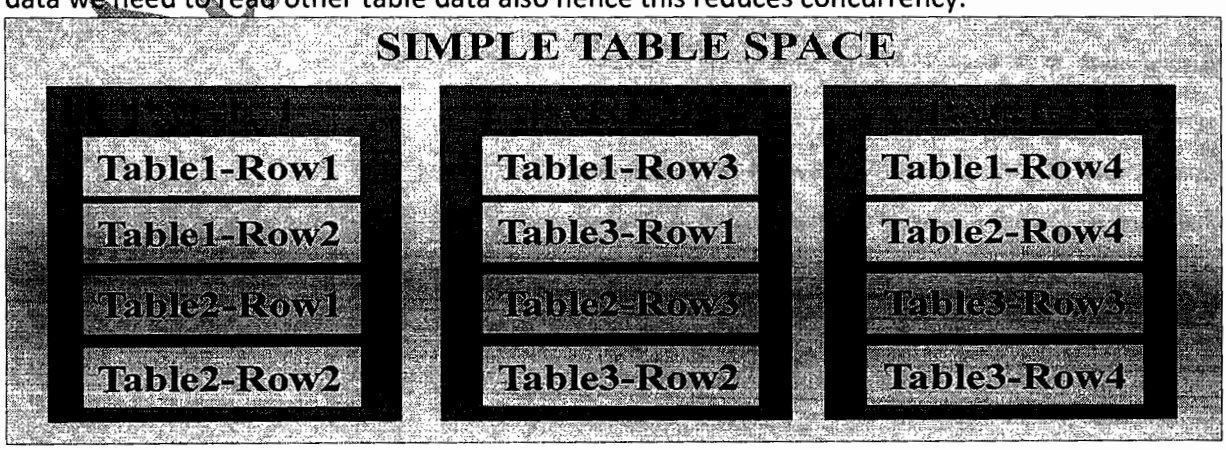
STORAGE GROUP:
It is a group of similar Volumes. Storage Group can have maximum 133 volumes.

DATABASE:
A Database is set of tables and table spaces. Objects in Database are logically related to each other.

TABLE SPACE:
The tables are physically stored in Table Space. The data in Table Space will be stored in the form of pages. There are three different types of Table Spaces:

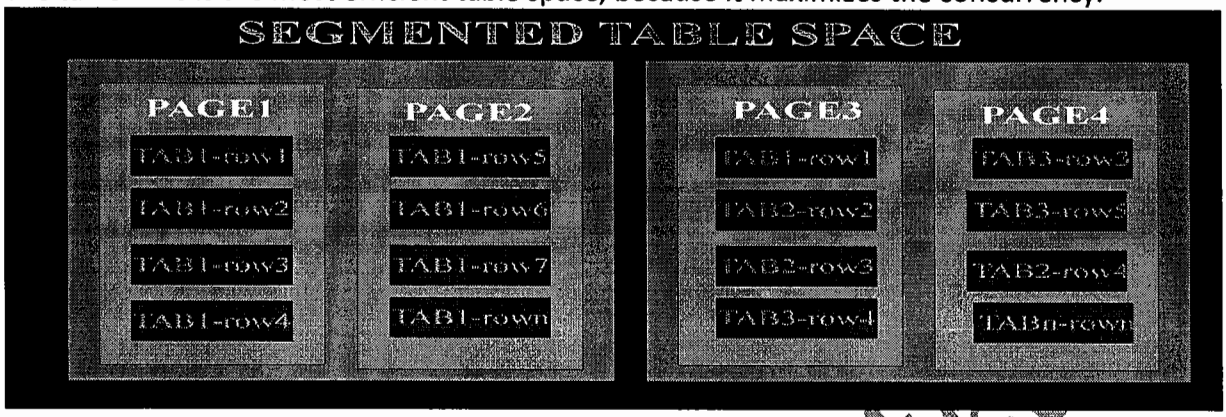
1. Simple Table Space.
2. Segmented Table Space.
3. Partitioned Table Space.

Simple Table Space:
The space in simple table space is divided into pages without any higher level structure. A simple table space can contain data from more than one table. So if we want access one table data we need to read other table data also hence this reduces concurrency.



Segmented Table Space:

The space is divided into units called segments and each segment contains the data from only one table. This is the most efficient table space, because it maximizes the concurrency.



Partitioned Table Space:

In Partitioned Table Space the space is divided into units called partitions.

Each Partition contains rows from single table. It is suitable for large tables that contain millions of pages.

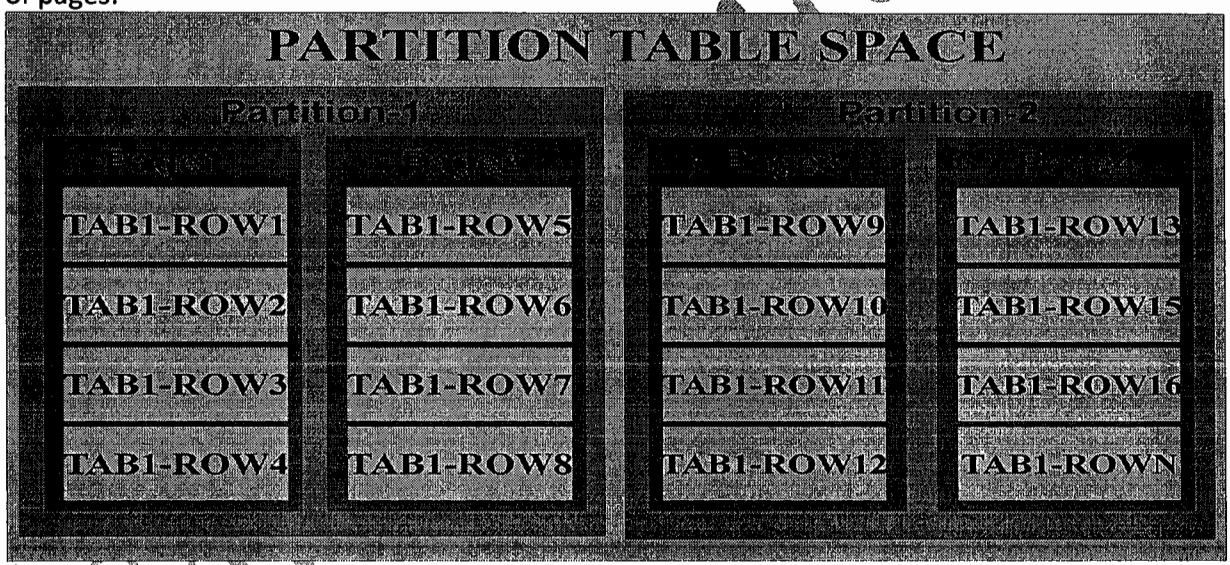


TABLE:

Table is collection of rows and columns.

VIEWS:

Views are Virtual Tables which are derived from more than one Table.

There are three types of statements in SQL:

1. DDL Data Definition Language.
2. DML Data Manipulation Language.
3. DCL Data Control Language.

DDL: Statements in DDL.

- ✓ CREATE (we can create Table Space, DB, Table etc.)
- ✓ ALTER
- ✓ DROP

DML: Statements in DML.

- ✓ SELECT
- ✓ INSERT
- ✓ UPDATE
- ✓ DELETE

DCL: Statements in DCL.

- ✓ GRANT
- ✓ REVOKE

CREATE:

It is used to create Database object.

Syntax:

```
CREATE TABLE TABLE-NAME
        (COL1 DATATYPE CONSTRAINT,
         COL2 DATATYPE CONSTRAINT,
         COL3 DATATYPE CONSTRAINT,
         PRIMARY KEY (COL1, COL2),
         FOREIGN KEY (COL1, COL2))
         IN DATABASE.TABLESPACE;
```

Note: To create table we should know the data types and constraints.

COLUMN NAMING RULES:

- ▶ Column name can be maximum up to 30 characters.
- ▶ We can use alphabets and numbers.

CONSTRAINT:

Constraint is a mechanism to control data. We have three constraints.

1. Null.
2. Not Null.
3. Not Null with default.

Null:

Null means unknown, If the value is not supplied during an insertion of row then null will be inserted into this column (Null is identified as .....).

Not Null:

We need to mandatorily pass values for the columns defined with not null constraint. We specify all the primary keys with Not Null.

Not Null With Default:

If the value is not supplied during an insertion of row, then based on the column, default values will be moved into the table.

DEFAULT VALUES:

Data Type	Default Value
Char	Spaces
Varchar	Empty String
Date, Time & Timestamp	Current Date, Time & Timestamp
Int, Smallint & Decimal	Zeros

PRIMARY KEY:

Primary key is used to identify the rows uniquely in the table. It cannot contain null values. Programs normally provide primary keys to get the row values from the table. Primary key can have one column or combination of columns. Primary key columns should have unique index. Unique index does not allow duplicate values.

Primary Key Syntax:

PRIMARY KEY (COL1, COL2);

Unique Index Syntax:

CREATE UNIQUE INDEX INDEX-NAME ON TABLENAME (COLUMN-NAME ASC/DESC);

FOREIGN KEY:

If we define any column as primary key in one table we can define that column only as only foreign key in any other table. It can contain duplicate and null values.

Defining a foreign key establishes a referential constraint between the foreign key (Child Table) and primary key (Parent Table)

Foreign Key Syntax:

FOREIGN KEY(COLUMN1, COLUMN2) references PARENT TABLE(Column Name) on delete CASCADE/RESTRICT/SET NULL.

REFERENTIAL INTEGRITY:

The foreign key values of dependent table should have a matching primary key values in a parent table is called "referential integrity".

DELETE RULES:

On Delete Cascade:

- If we delete the rows in a parent table then the related rows of dependent table will be deleted automatically.

On Delete Set Null:

- If we delete the rows of parent table then the foreign key values of the related rows in dependent table will set to nulls.

On Delete Restrict:

- If we delete rows of parent table which are having matching foreign key values in dependent table then we will get SQL error -532.
- In order to overcome this error we can delete the rows from dependent table first and then delete the rows from parent table.
- If we won't mention any of these by default ON DELETE RESTRICT will be substituted.

ALTER Syntax:

ALTER TABLE TABLENAME
ADD COLUMN NAME DATA TYPE CONSTRAINT;

- ALTER is used to modify the Table.

DROP Syntax:

DROP TABLE TABLENAME;

- DROP is used to drop entire Table.

SELECT Syntax:

SELECT * FROM TABLE NAME
WHERE CONDITION
ORDER BY COLUMN NAME ASC/DESC
GROUP BY COLUMN NAME
HAVING CONDITION;

- It is used to retrieve rows from the table

INSERT Syntax:

INSERT INTO TABLE NAME
(COLUMN1, COLUMN2.....COLUMNn)
VALUES
(COLUMN1 VALUE, COLUMN2 VALUE....COLUMNn VALUE);

- It is used to insert rows in the table.

UPDATE Syntax:

UPDATE TABLE NAME
SET COLUMN NAME = NEW VALUE
[WHERE CONDITION];

- It is used to update all rows in the table or selective rows.

DELETE Syntax:

DELETE FROM TABLE NAME
WHERE CONDITION;

- It is used to delete selective rows from table.

GRANT Syntax:

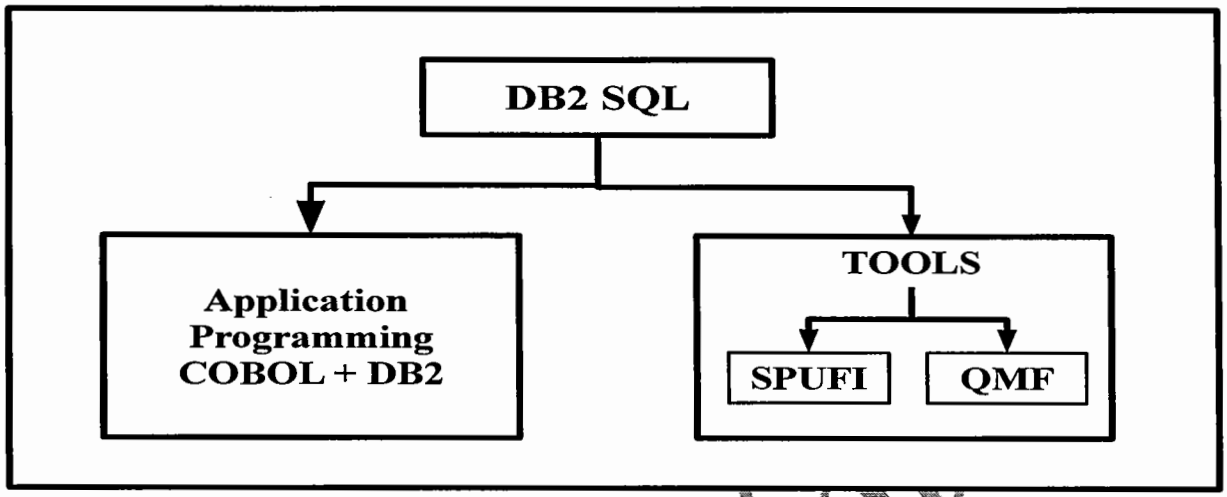
GRANT SELECT, INSERT, DELETE ON TABLENAME FORM RACFID;

- It is used to give privileges on table.

REVOKE Syntax:

REVOKE INSERT, DELETE ON TABLENAME FROM RACFID;

WE CAN EXECUTE ALL THE DB2 STATEMENTS USING DB2 SQL BY USING TOOLS SPUFI/QMF OR USING APPLICATION PROGRAMMING:



PART 1: USING SQL

PROCEDURE TO ENTER INTO SPUFI:

1. CREATE ONE PDS AS INPUT FILE.
   - USERID.NAME.DB2.INPUT
2. CREATE ONE SEQUENTIAL DATA SET AS OUTPUT FILE.
   - USERID.NAME.DB2.OUTPUT
3. FROM THE ISPF PRIMARY OPTION MENU SELECT D (DB2I).
4. SELECT 1 FOR SPUFI.
5. IN SPUFI SCREEN ENTER INPUT DS NAME ALONG WITH MEMBER NAME
   - USERID.NAME.DB2.INPUT (EMPMAST)
6. IN SPUFI SCREEN ENTER OUTPUT DS NAME
   - USERID.NAME.DB2.OUTPUT
7. WRITE SQL QUERY (DATABASE & TABLE SPACE IN FSS197DB.FSS197TS)
8. RUN THE QUERY BY GIVING ' ;;; '
   [EDIT;EXECUTE;AUTOCOMMIT;] IN COMMAND LINE & PRESS F3.
9. SQLCODE 00/100 MEANS QUERY EXECUTED SUCCESSFULLY

1) CREATE:

CREATE is used to create an object in the Database.

SQL 1: CREATE TABLE

```
CREATE TABLE EMPMAST
        (EMPID      CHAR(5)      NOTNULL,
         EFNAME     CHAR(15)     NOTNULL,
         ELNAME     CHAR(15)     NOTNULL,
         EMPDOB     DATE         NOTNULL,
         EMPSAL     DECIMAL(7,2) NOTNULL,
         EMPDEPT    CHAR(5)      NOT NULL,
         PRIMARY KEY(EMPID)) IN FSSADMDB.FSSADMTS;
```

2) INDEX:

- Index is a set of pointers to the rows of a table. It is used to access the records quickly.
- DB2 use index to ensure uniqueness and to improve performance by providing efficient access path to data for queries.

SQL 2: CREATE INDEX

- CREATE UNIQUE INDEX IDX2 ON EMPMAST(EMPID ASC/DESC);

Note: When the table is created that contains the Primary key, UNIQUE INDEX must be created on primary key. DB2 marks the table definition is incomplete until index is created on primary key.

3) INSERT:

- INSERT is used to insert the values into table.

SQL 3: INSERTING VALUES

```
INSERT INTO EMPMAST(EMPID,EFNAME,ELNAME,EMPDOB,EMPSAL,EMPDEPT)
        VALUES('A0001','JAIKUM','PRADEEP','1977-07-25',7000,'D0001');
INSERT INTO EMPMAST(EMPID,EFNAME,ELNAME,EMPDOB,EMPSAL,EMPDEPT)
                VALUES('A0003',' PRIYA','PV','1897-04-05',6000,'D0002');
INSERT INTO EMPMAST(EMPID,EFNAME,ELNAME,EMPDOB,EMPSAL,EMPDEPT)
        VALUES('B0001','ANURAG','GN','1997-07-05',12000,'D0003');
INSERT INTO EMPMAST(EMPID,EFNAME,ELNAME,EMPDOB,EMPSAL,EMPDEPT)
        VALUES('B0004','ABHILASH','GN','1900-12-05',10000,'D0001');
INSERT INTO EMPMAST(EMPID,EFNAME,ELNAME,EMPDOB,EMPSAL,EMPDEPT)
        VALUES('E0001','PRADEEP','PV','1999-11-05',15000,'D0001');
INSERT INTO EMPMAST(EMPID,EFNAME,ELNAME,EMPDOB,EMPSAL,EMPDEPT)
        VALUES('E0003','SEEMA','ROY','1977-08-30',8000,'D0005');
INSERT INTO EMPMAST(EMPID,EFNAME,ELNAME,EMPDOB,EMPSAL,EMPDEPT)
        VALUES('E0005','SINI','VINOD','1956-05-23',20000,'M0001');
INSERT INTO EMPMAST(EMPID,EFNAME,ELNAME,EMPDOB,EMPSAL,EMPDEPT)
        VALUES('E0007','DILEEP','KUMAR','1988-07-12',8000,'M0001');
INSERT INTO EMPMAST(EMPID,EFNAME,ELNAME,EMPDOB,EMPSAL,EMPDEPT)
        VALUES('E0008','DEEP','KU','1981-04-12',5000,'E0002');
INSERT INTO EMPMAST(EMPID,EFNAME,ELNAME,EMPDOB,EMPSAL,EMPDEPT)
        VALUES('E0009','MINI','MA','1981-08-07',9000,'E0002');
```

Note: Give Date In The Format YYYY-MM-DD.

4) SELECT:

- SELECT is used to retrieve the values from the table based on given query.

SQL 4: SELECT ALL RECORDS IN A TABLE

- To retrieve all the rows from the table.

```
SELECT * FROM EMPMAST;
```

SQL 5: DISPLAY 2 COLOUMNS – SELECT

- To retrieve selective Columns from the table.

```
SELECT EMPID,EFNAME FROM EMPMAST;
```

SQL 6: RETRIEVAL WITH WHERE – SELECT

- WHERE is used to check the condition.
- To select particular row based on the condition.

```
SELECT * FROM EMPMAST WHERE EMPID='E0001';
SELECT * FROM EMPMAST WHERE EMPDEPT='D0001';
```

ORDER BY:

- OREDR BY is used to retrieve the values of selective column in Sorted order and it may be either Ascending or Descending order. By default it takes Ascending order.

SQL 7: RETRIEVAL WITH ORDER BY.

```
SELECT * FROM EMPMAST ORDER BY EMPSAL;
```

SQL 8: RETRIEVAL WITH ORDER BY ASC – SELECT.

- To arrange the records in either ascending or descending order based on the selective column. By default it takes as ascending order.

```
SELECT * FROM EMPMAST ORDER BY EMPSAL ASC;
```

SQL 9: RETRIEVAL WITH ORDER BY - DESC – SELECT

```
SELECT EMPID,EFNAME FROM EMPMAST ORDER BY EMPID DESC;
```

SQL 10: RETRIEVAL WITH ' > ' SELECT

- To retrieve the values for selective column using arithmetic operator.

```
SELECT EMPID,EFNAME,EMPSAL FROM EMPMAST WHERE EMPSAL>10000;
```

SQL 11: RETRIEVAL WITH '> AND <' SELECT

- To retrieve the range of values for selective column using arithmetic operators.

```
SELECT EMPID,EFNAME,EMPSAL FROM EMPMAST WHERE EMPSAL>5000 AND EMPSAL<15000;
```

SQL 12: SELECT - DISTINCT

- Distinct is used to eliminate the values without duplicates.

```
SELECT DISTINCT (EMPDEPT) FROM EMPMAST;
```

BETWEEN:

To specify a range of values in the expression 'BETWEEN' is the operator to be implemented.

SQL 13: SELECT - BETWEEN || >= AND <=

- To retrieve the range of values for selective column using logical operators.

```
SELECT EMPID,EFNAME,EMPSAL FROM EMPMAST          WHERE
                                EMPSAL BETWEEN 7000 AND 15000;
                        OR
SELECT EMPID, EFNAME,EMPSAL FROM EMPMAST         WHERE
                                EMPSAL >=7000 AND EMPSAL<=15000;
```

NOT BETWEEN:

- NOT BETWEEN is used to retrieve the values which is not in the given range of values.

SQL 14: SELECT – NOT BETWEEN

```
SELECT EMPID,EFNAME,EMPSAL FROM EMPMAST          WHERE
                        EMPSAL NOT BETWEEN 7000 AND 15000;
```

IN:

- To extract data by comparing more than one value with the selected column of a table is possible with 'IN' operator. This operator provides facility of giving multiple possibilities for the comparison of one result value with the column.

SQL 15: SELECT - IN

```
SELECT EMPID,EFNAME,EMPDEPT FROM EMPMAST    WHERE
                EMPDEPT IN('D0001','D0002','D0003');
        OR
SELECT EMPID,EFNAME,EMPDEPT FROM EMPMAST         WHERE
                EMPDEPT = 'D0001' OR EMPDEPT = 'D0002' OR EMPDEPT = 'D0003';
```

NOT IN:

- This operator reverses the result of the expression from True of false vice versa.

SQL 16: SELECT – NOT IN

```
SELECT EMPID,EFNAME,EMPDEPT FROM EMPMAST         WHERE
                EMPDEPT NOT IN('D0001','D0002','D0003');
        OR
SELECT EMPID,EFNAME FROM EMPMAST                 WHERE
EMPDEPT NOT= 'D0001' OR EMPDEPT NOT= 'D0002' OR EMPDEPT NOT= 'D0003';
```

LIKE:

- To extract data from the table, user can compare characters of the column as single one or multiple ones. Single character comparison with '_(under Score)' and multiple character comparison with '%'(percentage).

SQL 17: RETRIEVAL WITH LIKE – SELECT

- To retrieve the values for EFNAME which starts with A using LIKE operator.

```
SELECT * FROM EMPMAST WHERE EFNAME LIKE 'A%';
```

SQL 18: RETRIEVAL WITH LIKE – SELECT

```
SELECT * FROM EMPMAST WHERE ELNAME LIKE 'R%';
```

AGGREGATE FUNCTION

SUM:

- It gives the sum of all the column values.

SQL 19: AGGREGATE FUNCTION – SUM

```
SELECT SUM(EMPSAL) FROM EMPMAST;
```

AVG:

- It gives the average of all the column values.

Note: AVG function will give an incorrect average in case there are null values in the columns.

SQL 20: AGGREGATE FUNCTION – AVG

```
SELECT AVG(EMPSAL) FROM EMPMAST;
```

SQL 21: AGGREGATE FUNCTION – MIN

- It gives the minimum value from a set of columns values.

```
SELECT MIN(EMPSAL) FROM EMPMAST;
```

SQL 22: AGGREGATE FUNCTION – MAX

- It gives the maximum value from a set of columns values.

```
SELECT MAX(EMPSAL) FROM EMPMAST;
```

SQL 23: AGGREGATE FUNCTION – COUNT

- It gives the count of columns values, exclude null values.

```
SELECT COUNT(*) FROM EMPMAST WHERE EMPDEPT='D0001';
```

SQL 24: RETRIEVAL WITH LIKE (STARTS WITH 'P' OR ENDS WITH 'R')

```
SELECT ELNAME,EMPSAL FROM EMPMAST WHERE ELNAME LIKE 'K%'
                    OR                  ELNAME LIKE '%R';
```

If We Give →

```
SELECT EMPSAL FROM EMPMAST WHERE EMPID LIKE 'E%'
                    OR                  ELNAME LIKE '%3';
```

- It will display corresponding field starting with e and ending with 3 if the field contains maximum size given at the time of creating.
- In order to get result for '%R' declare ELNAME with VARCHAR instead of CHAR.

SQL 25: RETRIEVAL WITH COMPUTATION

```
SELECT EFNAME,EMPSAL+500 FROM EMPMAST WHERE EMPDEPT='E0002';
```

SQL 26: RETRIEVAL WITH COMPUTTATION

```
SELECT EFNAME,EMPSAL-5000 FROM EMPMAST WHERE EMPDEPT LIKE 'M%';
```

GROUP BY CLAUSE:

- It is used to group the related values based on condition. Whenever we use GROUP BY clause we must use one aggregate function.

SQL 27: GROUP BY & SUM

```
SELECT EMPDEPT,SUM(EMPSAL) FROM EMPMAST GROUP BY EMPDEPT;
```

HAVING CLAUSE:

- It should contain one aggregate function. It must be used with GROUP BY clause.
- HAVING clause operates on aggregated groups that are already selected from GROUP BY clause.
- HAVING clause should contain one condition which includes aggregate function.
- GROUP BY clause should not contain where clause it may have only having clause.

SQL 28: GROUP BY WITH HAVING & SUM

```
SELECT EMPDEPT,SUM(EMPSAL) AS EMPSALS FROM EMPMAST    GROUP BY
        EMPDEPT  HAVING SUM(EMPSAL) > 10000;
```

SQL 29: DELETE - ON DELETE CASCADE

STEP - 1

```
CREATE TABLE S
        (S#             CHAR(2)         NOT NULL,
        SNAME           CHAR(10)        NOT NULL,
        STATUS          SMALLINT        NOT NULL,
        CITY            VARCHAR(10)  NOT NULL,
        PRIMARY KEY(S#)) IN FSS197DB.FSS197TS;
```

STEP - 2

```
CREATE UNIQUE INDEX IDX3 ON S(S#);
```

STEP - 3

```
CREATE TABLE P
        (P#             CHAR(2)         NOT NULL,
        PNAME           CHAR(10)        NOT NULL,
        COLORS          CHAR(10)        NOT NULL,
        WEIGHT          SMALLINT        NOT NULL,
        CITY            CHAR(10)        NOT NULL,
        PRIMARY KEY(P#)) IN FSS197DB.FSS197TS;
```

STEP - 4

```
CREATE UNIQUE INDEX IDX4 ON P(P#);
```

STEP - 5

```
CREATE TABLE SP
        (SP#            CHAR(2)         NOT NULL,
        PS#             CHAR(2)         NOT NULL,
        QTY             SMALLINT        NOT NULL,
        FOREIGN KEY(SP#) REFERENCES S ON DELETE CASCADE,
        FOREIGN KEY(PS#) REFERENCES P ON DELETE SET NULL)
        IN FSS197DB.FSS197TS;
```

STEP - 6 "INSERT 5 ROWS IN TABLE S"

```
INSERT INTO S(S#,SNAME,STATUS,CITY) VALUES('S1','SMITH',20,'LONDON');
INSERT INTO S(S#,SNAME,STATUS,CITY) VALUES('S2','JONES',10,'PARIS');
INSERT INTO S(S#,SNAME,STATUS,CITY) VALUES('S3','BLAKE',30,'PARIS');
INSERT INTO S(S#,SNAME,STATUS,CITY) VALUES('S4','CLARK',20,'LONDON');
INSERT INTO S(S#,SNAME,STATUS,CITY) VALUES('S5','ADAMS',30','ATHENS');
```

STEP – 7 "INSERT 6 ROWS IN TABLE P"

```
INSERT INTO P(P#,PNAME,COLORS,WEIGHT,CITY) VALUES('P1','NUT','RED',12,'LONDON');
INSERT INTO P(P#,PNAME,COLORS,WEIGHT,CITY) VALUES('P2','BOLT','GREEN',17,'PARIS');
INSERT INTO P(P#,PNAME,COLORS,WEIGHT,CITY) VALUES('P6','COG','RED',19,'LONDON');
INSERT INTO P(P#,PNAME,COLORS,WEIGHT,CITY) VALUES('P4','SCREW','RED',14,'LONDON');
INSERT INTO P(P#,PNAME,COLORS,WEIGHT,CITY) VALUES('P5','CAM','BLUE',12,'PARIS');
INSERT INTO P(P#,PNAME,COLORS,WEIGHT,CITY) VALUES('P3','SCREW','BLUE',17,'ROME');
```

STEP – 8 "INSERT 12 ROWS IN TABLE SP"

```
INSERT INTO SP(SP#,PS#,QTY) VALUES('S1','P1',300);
INSERT INTO SP(SP#,PS#,QTY) VALUES('S1','P2',200);
INSERT INTO SP(SP#,PS#,QTY) VALUES('S1','P3',400);
INSERT INTO SP(SP#,PS#,QTY) VALUES('S1','P4',200);
INSERT INTO SP(SP#,PS#,QTY) VALUES('S1','P5',100);
INSERT INTO SP(SP#,PS#,QTY) VALUES('S1','P6',10);
INSERT INTO SP(SP#,PS#,QTY) VALUES('S4','P5',400);
INSERT INTO SP(SP#,PS#,QTY) VALUES('S2','P2',400);
INSERT INTO SP(SP#,PS#,QTY) VALUES('S3','P2',200);
INSERT INTO SP(SP#,PS#,QTY) VALUES('S4','P2',200);
INSERT INTO SP(SP#,PS#,QTY) VALUES('S4','P4',300);
INSERT INTO SP(SP#,PS#,QTY) VALUES('S4','P4','300');
```

STEP – 9 DISPLAY ALL TABLE VALUES.

TABLE S SELECT * FROM S;

S#	SNAME	STATUS	CITY
S1	SMITH	20	LONDON
S2	JONES	10	PARIS
S3	BLAKE	30	PARIS
S4	CLARK	20	LONDON
S5	ADAMS	30	ATHENS

TABLE P SELECT * FROM P;

P#	PNAME	COLORS	WEIGHT	CITY
P1	NUT	RED	12	LONDON
P2	BOLT	GREEN	17	PARIS
P6	COG	RED	19	LONDON
P4	SCREW	RED	14	LONDON
P5	CAM	BLUE	12	PARIS
P3	SCREW	BLUE	17	ROME

TABLE SP SELECT * FROM SP;

SP#	PS#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	10
S4	P5	400
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S2	P1	300

STEP - 10 (DELETE A ROW)

DELETE FROM S WHERE S#='S2';

STEP - 11

AFTER DELETING THE VALUES IN TABLE ARE

TABLE S SELECT * FROM S;

S#	SNAME	STATUS	CITY
S1	SMITH	20	LONDON
S3	BLAKE	30	PARIS
S4	CLARK	20	LONDON
S5	ADAMS	30	ATHENS

TABLE SP SELECT * FROM SP;

SP#	PS#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	10
S3	P2	200
S4	P2	200
S4	P4	300

Note: If we give **on delete restrict** at the time of definition, then table 'S' first checks to ensure that there are no rows in table 'SP' with S2 values. If db2 finds any row in SP with values S2 in S# column then the delete fails.

If we give **on delete set null** then S2 row is deleted from table s and corresponding S# value in table 'SP' is set to null.

SQL 30: UNION:

- Union is used to retrieve the rows from two different tables.
- The no of columns that are being selected from both the tables should be same and must contain same length and data type.

UNION: (REDUNDANT DUPLICATES ARE ELIMINATED)

QUERY: GET PART NUMBERS FOR PARTS THAT EITHER WEIGHT MORE THAN 16 POUNDS OR ARE SUPPLIER BY S2 (UNION)

```
SELECT P# FROM P WHERE WEIGHT>16      UNION    SELECT PS# FROM SP WHERE S#='S2';
   +---------+
       P#
   +---------+
       P1
       P2
       P3
       P6
   +---------+
```

UNION ALL: (REDUNDANT DUPLICATES ARE NOT ELIMINATED)

- It retrieves all the rows from both the table along with duplicates.

```
SELECT P# FROM P WHERE WEIGHT>16  UNION ALL  SELECT PS# FROM SP WHERE S#='S2';
   +---------+
       P#
   +---------+
       P2
       P6
       P3
       P2
       P1
   +---------+
```

SQL 31: JOIN:

- It is used to combine columns from more than one table. It is a query used to retrieve the data from more than one table. This join was classified in to three types.
  1. **Simple Join.**
  2. **Inner Join.**
  3. **Outer Join.**

STEP - 1

```
CREATE TABLE EMP
            (EMPNO      INT          NOT NULL,
             LASTNAME   CHAR(10)     NOT NULL)
             IN FSS197DB.FSS197TS;
```

STEP – 2 "INSERT 7 ROWS"

```
INSERT INTO EMP(EMPNO,LASTNAME) VALUES(000010,'JOHN');
INSERT INTO EMP(EMPNO,LASTNAME) VALUES(000150,'ROBERT');
INSERT INTO EMP(EMPNO,LASTNAME) VALUES(000020,'TOM');
INSERT INTO EMP(EMPNO,LASTNAME) VALUES(000250,'SMITH');
INSERT INTO EMP(EMPNO,LASTNAME) VALUES(000100,'BETTY');
INSERT INTO EMP(EMPNO,LASTNAME) VALUES(000070,'JIL');
INSERT INTO EMP(EMPNO,LASTNAME) VALUES(000200,'JACK');
```

STEP - 3

```
CREATE TABLE DEPT
            (DEPTNAME   CHAR(10)      NOT NULL,
             MGRNO      INT           NOT NULL)
             IN FSS197DB.FSS197TS;
```

STEP – 4 INSERT 5 ROWS

```
INSERT INTO DEPT(DEPTNAME,MGRNO) VALUES('SPCOMP',000010);
INSERT INTO DEPT(DEPTNAME,MGRNO) VALUES('DEVP',-------);
INSERT INTO DEPT(DEPTNAME,MGRNO) VALUES('SW',000100);
INSERT INTO DEPT(DEPTNAME,MGRNO) VALUES('ADMIN',000070);
INSERT INTO DEPT(DEPTNAME,MGRNO) VALUES('BRANOFF',-------);
```

STEP – 5 DISPLAY TABLES

TABLE EMP SELECT * FROM EMP;

EMPNO	LASTNAME
10	JOHN
150	ROBERT
20	TOM
250	SMITH
100	BETTY
70	JIL
200	JACK

TABLE DEPT SELECT * FROM DEPT;

DEPTNAME	MGRNO
SPCOMP	10
DEVP	---------
SW	100
ADMIN	70
BRANOFF	---------

STEP - 6

SIMPLE JOIN:

SELECT LASTNAME,DEPTNAME FROM EMP,DEPT WHERE EMPNO=MGRNO;

LASTNAME	DEPTNAME
JOHN	SPCOMP
JIL	ADMIN
BETTY	SW

INNER JOIN:

- It retrieves only the matched records from the join tables.

SELECT LASTNAME,DEPTNAME FROM EMP INNER JOIN DEPT ON EMPNO=MGRNO;

LASTNAME	DEPTNAME
JOHN	SPCOMP
JIL	ADMIN
BETTY	SW

OUTER JOIN:

- There are three types of OUTER JOIN.
  1. RIGHT OUTER JOIN.
  2. LEFT OUTER JOIN.
  3. FULL JOIN.

RIGHT OUTER JOIN:

- It retrieves the matched and unmatched rows from right table with the left table columns set to null.

SELECT LASTNAME,DEPTNAME FROM EMP RIGHT OUTER JOIN DEPT ON EMPNO=MGRNO;

LASTNAME	DEPTNAME
JOHN	SPCOMP
JIL	ADMIN
BETTY	SW
----------	DEVP
----------	BRANOFF

LEFT OUTER JOIN:

- It retrieves the matched records and unmatched records from left table with the right table columns set to null.

SELECT LASTNAME,DEPTNAME FROM EMP LEFT OUTER JOIN DEPT ON EMPNO=MGRNO;

LASTNAME	DEPTNAME
JOHN	SPCOMP
TOM	----------
JIL	ADMIN
BETTY	SW
ROBERT	----------
SMITH	----------
JACK	----------

FULL JOIN:

- It retrieve the all matched and unmatched rows from left and right tables with the corresponding left and right table columns values set to null.

SELECT LASTNAME,DEPTNAME FROM EMP FULL JOIN DEPT ON EMPNO=MGRNO;

LASTNAME	DEPTNAME
JOHN	SPCOMP
TOM	----------
JIL	ADMIN
BETTY	SW
ROBERT	----------
JACK	----------
SMITH	----------
----------	DEVP
----------	BRANOFF

VIEWS:

- It is an alternate way of looking data in one or more tables. And these are two types.
  1. Updatable Views.
  2. Read Only Views.

UPDATABLE VIEW:

- A VIEW set to be updatable when it satisfies all below condition.
  1. View is created on a single table.
  2. No aggregate function or arithmetic function use.
  3. No HAVING, GROUP BY, DISTINCT clauses are used.
  4. No Sub Query is used.
  5. Base table column other than view columns are nullable.

Syntax:

```
CREATE VIEW VIEWNAME AS
SELECT QUERY
```

NON-UPDATABLE VIEW:

- A view is set to be non updatable when does not satisfy any one of all the updatable view conditions.

Syntax:

CREATE VIEW VIEWNAME AS
SELECT QUERY

ALIAS & SYNONYMS:

- Both Alias & Synonym are alternate way of looking data in the tables.

SYNONYMS:

- Only the user who has created the synonym can access it.
- When the base table is dropped, associated synonym will be dropped.
- We need not require system admin authorization to create Synonym.

Syntax:

CREATE SYNONYM SYNONYMNAME AS SELECT * FROM TABLENAME.

ALIAS:

- Accessible by the entire users.
- When the base table is dropped, associated alias will not be dropped.
- We need system admin authorization to create alias.

Syntax:

CREATE ALIAS ALIASNAME AS SELECT * FROM TABLENAME.

SUBQUERY:

- A query with in a query is called sub query. We can code a maximum of 15 sub queries.

Syntax:

SELECT * FROM TABLE WHERE = (SELECT * FROM TABLE NAME)

- There are two types of sub queries.
  1. CORRELATED SUB QUERY.
  2. NON-CORRELATED SUB QUERY.

CORRELATED SUB QUERY:

- First the outer query is executed and for each row of outer query for the inner query is executed.

NON CORRELATED SUB QUERY:

- First the inner query is executed then the outer query

E.g.: To find the 2nd maximum salary from a table.

```
SELECT MAX(EMPSAL) FROM EMPMAST      WHERE
        SAL < ( SELECT MAX(EMPSAL) FROM EMPMAST)
```

PART 2: USING APPLICATION PROGRAMMING

DATA TYPES IN DB2 AND EQUIVALENT COBOL FIELDS:

DB2	COBOL
CHAR(n)	10 FIELD-A PIC X(n)
VARCHAR(n)	10 FIELD-A 49 FIELD-A-LENGTH PIC S9(4) COMP 49 FIELD-A-TEXT PIC X(n)
SMALLINT	10 FIELD-A PIC S9(4) COMP
INTEGER	10 FIELD-A PIC S9(9) COMP
DECIMAL(p,q)	10 FIELD-A PIC S9(p-q) V 9(q)COMP3
DATE	10 FIELD-A PIC X(10)
TIME	10 FIELD-A PIC X(8)
TIME STAMP	10 FIELD-A PIC X(26)

DATE FORMAT: 10 Bytes → (YYYY-MM-DD)
TIME FORMAT: 8 Bytes→ (HH:MM:SS)
TIME STAMP: 26 Bytes → (YYYY-MM-DD: HH:MM:SS: NNNNNN)

We can access the DB2 database through application program by coding the SQL statements in the application program.
We have to code the SQL statements and any DB2 related DCLGEN and SQLCA should be coded between the delimiters.

```
EXEC SQL
---------
---------
END-EXEC.
```

E.g.:
```
EXEC SQL
     INCLUDE SQLCA
END-EXEC.
```

DCLGEN:

- DCLGEN is a DB2 provided function to generate the DCLGEN.
- It contains the declaration of the table with all columns and the structure of equivalent COBOL host variables.
- We have to include the DCLGEN in working-storage section of the program in Area-B.

E.g.: EXEC SQL
INCLUDE EMPDCL
END-EXEC.

- To create DCLGEN we have to go to option DB2

INCLUDE:

- Include is same as copy statement in COBOL, include will expand the DCLGEN at the time of pre-compilation process.

SQLCA:

- SQL Communication Area.
- It contains the set of the fields which can be updated by DB2 after completion of each SQL statements.
- There are two important fields in SQLCA
  1. SQLCODE.
  2. SQLERRD(3).

SQLCODE:

- It is updated by DB2 after execution of each SQL statement with appropriate statement. This SQL code is used to handle the DB2 errors in the application program.

SQLERRD(3):

- This contains the number of rows affected by the statements insert, update, delete.
- We have to include the SQLCA in working storage section of the application program.

```
EXEC SQL
        INCLUDE SQLCA
END-EXEC.
```

HOST VARIABLE:

- Using embedded SQL statement in the application program we have to specify the storage area into which the DB2 can place the data.

Syntax:

Select into.....

- The INTO clause names the coded fields that will be used to store the data which is retrieved from DB2. These fields are called HOST VARIABLES.
- Host variables must be preceded with column**(:)** in SQL statement which are used in the application program.

E.g.:

```
EXEC SQL
    INSERT INTO EMPTAB
            (EMP_NO,
            EMP_NAME,
            EMP_CITY)
        VALUES(:HS-EMP-NO,
            :HS-EMP-NAME,
            :HS-EMP-CITY)
END-EXEC.
```

NULL INDICATOR:

- To handle the null values in an application program we have to use null indicator.
- While we are retrieving the data from the DB2, if any column have null we have to handle them in application program by using null indicator otherwise the program will abend with SQL code -305.
- We have to declare null indicator in working-storage section with picture clause S9(4) Comp and then use in SQL statements of application program.

```
WORKING-STORAGE SECTION.
01      NAME-ID      PIC     S9(4) COMP.
PROCEDURE DIVISION.
        EXEC SQL
                SELECT INTO
                        (:HS-EMP-NO
                         :HS-EMP-NAME       NAME-ID)
                FROM  EMPTAB
        END-EXEC.
```

1. If null indicator name-id is zero then there is a data existing for EMP-NAME column, it retrieved successfully into the application program.
2. If null indicator is -1 the value in the EMP-NAME is null and no data is retrieved into the application program.
3. If null indicator is positive value then the EMP-COLUMN is having the value but it is truncating while retrieving the application program.

COBOL DB2 PROGRAM PREPARATION:

APPLICATION PROGRAMMING

We can execute 'SQL' Queries in application program by embedding SQL statements between EXEC SQL an END-EXEC.

Syntax:

```
EXEC  SQL
      DB2 ST1
      DB2 ST2
END-EXEC.
```

Note: We need to code in area-B / margin-B

- To write any COBOL+DB2 program, we need to make use of following two copy books.
  1. SQLCA (SQL Communication Area)
  2. DCLGEN (Declaration Generator)

1. SQLCA:

- It is DB2 Pre-Defined Copy book
- The total size of SQLCA copy book is 136 bytes.
- Of all the fields in SQLCA, the important fields are SQL CODE and SQL EERD(3)

A) SQL CODE:

- It is used for error handling for DB2 in application programming.

SQL CODE = 0 → Successful Operation
SQL CODE = +ve → Warnings
SQL CODE = - ve → Errors

B) SQL ERRD(3):

- It holds the count of rows affected by the query.
- We include SQLCA copy book by following syntax.

```
EXEC SQL
        INCLUDE SQLCA
END-EXEC.
```

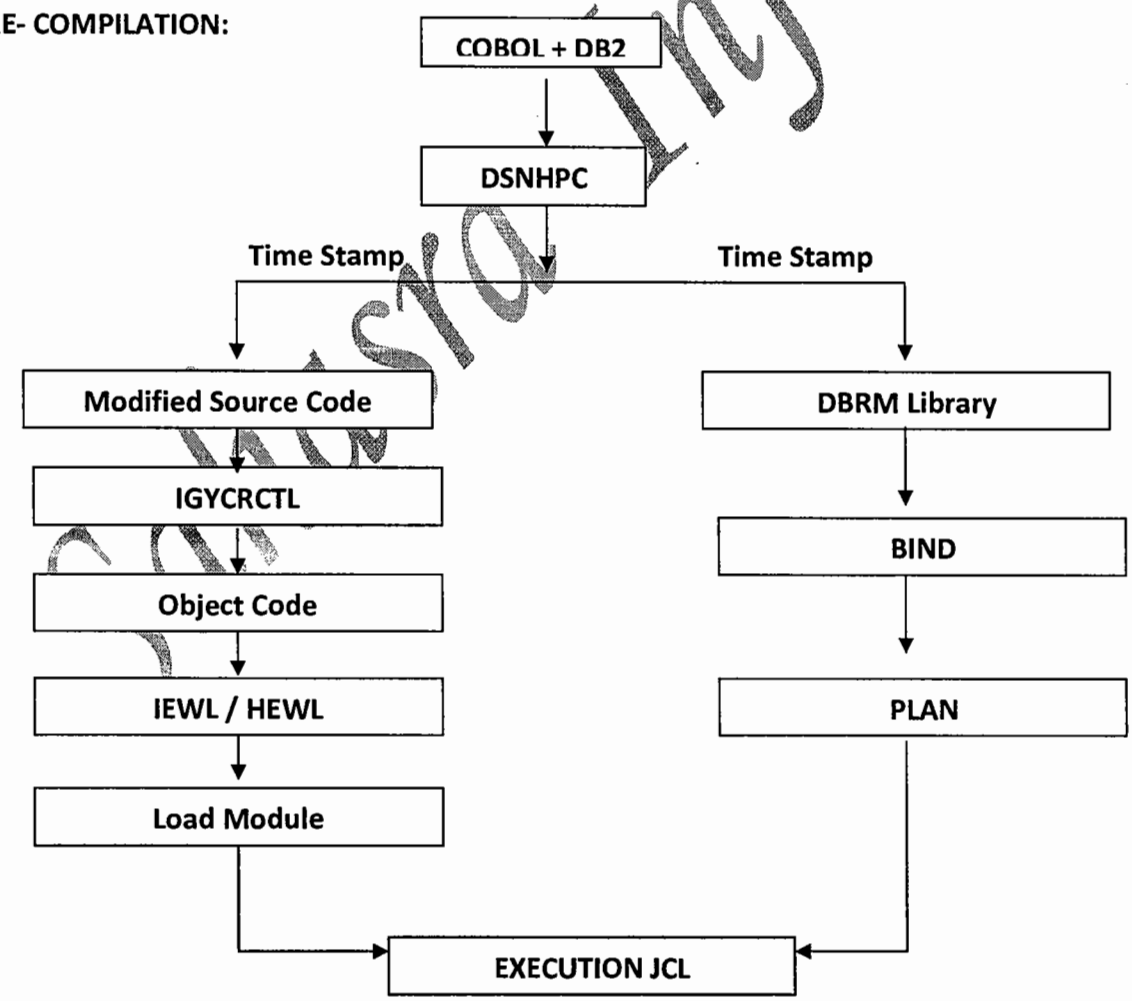
2) DCLGEN:

- DCLGEN is a tool, designed to convert DB2 equivalent variables into COBOL equivalent variables.
- The converted COBOL equivalent variables are referred as 'Host Variables'.
- Host variables acts as an Interface between application programming and DB2 subsystems.

Program Preparation:

1. Write COBOL + DB2 program.
2. Pre-compilation.
3. BIND JCL.
4. Execution JCL.

PRE- COMPILATION:



Pre-Compiler:

- As the normal COBOL compiler can't understand the statements embedded between EXEC and END-EXEC we go for process called "Pre-Compilation".
- Pre-compilation divides the COBOL + DB2 program into half.
- One half we get all the COBOL statements and where ever it finds DB2 statements it will replace with CALL statements. This is called 'MODIFY SOURCE CODE'.
- As modify source code contains only COBOL statements will be stored in 'DBRMLIB'.

Note:

- During the separation pre-compiler pass a time-stamp to load module and DBRMLIB to maintain 'Integrity' (Correctness of data)
- Later the time-stamp in 'DBRMLIB' will be stored in 'Plan'.

BIND JCL:

- In BINDJCL we make use of utility program 'IKJEFT01/ (1A/1B).
- The input for BINDJCL will be 'DBRMLIB'.
- During the Bind process it checks the authorization of users to execute the queries, if the user is authorized to execute the quires will be executed and the result will be stored in package and packages are not executable.
- So we convert package into plan, which is executable.

EXECUTION JCL:

- In execution JCL, we combine both load module and plan.
- During the execution, time stamp runtime supervisor utility will check the time stamp between load module and plan.
- If time stamp is same in load module and plan, our execution JCL will be find, if there is mismatch then it will abended with SQL error code '- 818'.

DB2 EXECUTION JCL:

```
//JOBNAME    JOB      123, 'XYZ', CLASS=A, NOTIFY=&SYSUID
//STEP1               EXEC   PGM=IKJEFT01
//SYSPRINT  DD  *
          DSN (SUB SYSTEM NAME) –
          MEMBER (PROGRAM NAME) / RUN PROGRAM (PROGRAM NAME) –
          PLAN (PLAN NAME) –
          LOADLIB (LOAD MODULE P.D.S)
          END
/*
//SYSIN DD *
 VALUE 1
 VALUE 2
 ------
 ------
 VALUE N
/*
//
```

BIND JCL

```
//JOBBIND    JOB   123,'SAHASRA',CLASS=A,NOTIFY=&SYSUID
//STEP1      EXEC  PGM=IKJEFT01
//SYSPRINT   DD    SYSOUT=*
//SYSOUT     DD    SYSOUT=*
//SYSTSIN    DD    *
       DSN(SUB SYSTEM NAME)              –
       PLAN(PLAN NAME)                   –
       MEMBER(MEMBER NAME)               –
       ISOLATION LEVEL(CS/RS/RR/UR)      –
       ACTION LEVEL(ADD/REPLACE)         –
       EXPLAIN(YES/NO)                   –     → (For Shortest Path)
       AQUIRE(USE/ALLOCATE)              –
       RELEASE(COMMIT/ROLLBACK)          –
       VALIDATE(BIND/RUN)                –
       SQL ERRORS(NO PACKAGE / CONTINUE)–
       DBRMLIB(DBRMLIB P.D.S)
/*
//
```

DSN:

- It is going to identify the sub-system name, in order to execute BIND JCL, the DB2 Sub-System should be up.
- Generally the subsystem name will be 4 characters.

PLAN:

- The output of BIND JCL will be plan and it is executable.

MEMBER:

- It is a DBRMLIB member name.

ISOLATION LEVEL:

- It contains 3 sub parameters.
  1. Cursor Stability (CS)
  2. Repeatable Read (RR)
  3. Uncommitted Read (UR)

1. Cursor Stability (CS):

- It is low level locking, when the cursor works on particular row then the lock will be acquire on the row, when the user moves to next row then the lock over previous row will be released.
- This is the best option for Isolation Level Parameter.

2. Repeatable Read (RR):

- It is a default option and it acquires the lock until it commit across commit point and release the lock when it moves to next row after commit point.

3. Uncommitted Read (UR):

- It doesn't acquire any lock.
- This is faster of all isolation level options.
- We use this option only for 'Select'.

ACTION:

- ADD: if the PLAN is not present in SYSIBM.SYSPLAN we specify option as ADD.
- REPLACE: If the PLAN is already present, it is going to replace in the old one with new plan.

EXPLAIN:

- YES: if we specify 'YES' it is going t call predefined utility 'OPTEMIZER', it calls another utility 'RUN STATS'.
- Based on the historical information. It is going to choose the shortest access path and fetch the result and it stores in Package.
- NO: if we can choose any of the access paths, this option increases CPU time.

AQUIRE:

- USE: it acquires the lock over indexes, tables when we are using it.
- ALLOCATE: it acquires the lock over indexes and tables when it is allocating.

RELEASE:

- COMMIT: it release the lock over indexes and tables when it comes across COMMIT.
- ROLLBACK: it releases the lock over indexed and tables when it comes across rollback statement.
- The best option for acquires and release are USE & COMMIT.

VALIDATE:

- BIND: it is going to check the user authorization during BIND time, this option is more efficient.
- RUN: it is going to check the user authorization during run time.

SQL ERROR:

- NO PACKAGE: if there are SQL errors it will not create any package
  - This is default option.
- CONTINUE: it create a package, even when there are SQL errors.

DBRMLIB:

It will be the input to the bind JCL and it contains all the SQL errors.

LOCKS:

- To protect the data in DB2 database. DB2 uses the locking process. By using locks to maximize the concurrency so the user can quickly access the data.
- There are three types of locks.

1. **SHARE LOCK**: More than one user can read the data but the data cannot modify in this lock.
2. **UPDATE LOCK**: Only one user can update the data by using update lock, but the other user can only read the data.
3. **EXCLUSIVE LOCK**: Only one user can issue the exclusive lock, but no other user can read the data.

PROGRAM TO INSERT ROWS INTO THE TABLE... EMPMAST

```
IDENTIFICATION DIVISION.
PROGRAM-ID. INSPGM.
AUTHOR. SAHASRA.
ENVIRONMENT DIVISION.
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
     EXEC SQL
       INCLUDE SQLCA
     END-EXEC.
          EXEC SQL
       INCLUDE EMPDCL
     END-EXEC.
01 WS-EMP-REC.
        05      WS-EMPID      PIC     X(5)          VALUE SPACES.
        05      WS-EFNAME     PIC     X(15)         VALUE SPACES.
        05      WS-ELNAME     PIC     X(15)         VALUE SPACES.
        05      WS-EMPDOB     PIC     X(10)         VALUE SPACES.
        05      WS-EMPSAL     PIC     S9(5)V99      VALUE ZEROS .
        05      WS-EMPDEPT    PIC     X(5)          VALUE SPACES.
01      WS-EOT          PIC     X(3)    VALUE         'NO'.
PROCEDURE DIVISION.
000-MAIN-RTN.
        PERFORM 200-PROCESS-RTN THRU 200-PROCESS-EXIT UNTIL OPTION = 'YES'.
        STOP RUN.
200-PROCESS-RTN.
        ACCEPT WS-EMPID.
        ACCEPT WS-EFNAME.
        ACCEPT WS-ELNAME.
        ACCEPT WS-EMPDOB.
        ACCEPT WS-EMPSAL.
        ACCEPT WS-EMPDEPT.
        MOVE  WS-EMPID      TO      HS-EMPID.
        MOVE  WS-EFNAME     TO      HS-EFNAME.
        MOVE  WS-ELNAME     TO      HS-ELNAME.
        MOVE  WS-EMPDOB     TO      HS-EMPDOB.
        MOVE  WS-EMPID      TO      HS-EMPID.
        MOVE  WS-EMPSAL     TO      HS-EMPSAL.
        MOVE  WS-EMPDEPT    TO      HS-EMPDEPT.
        EXEC SQL
             INSERT INTO EMPMAST(EMPID,
                                 EFNAME,
                                 ELNAME,
                                 EMPDOB,
                                 EMPSAL,
                                 EMPDEPT)
                VALUES(:HS-EMPID,
                       :HS-EFNAME,
                       :HS-ELNAME,
                       :HS-EMPDOB,
                       :HS-EMPSAL,
                       :HS- EMPDEPT)
```

```
        END-EXEC.
        IF SQLCODE NOT EQUAL 0
                DISPLAY 'ERROR DURING INSERTION:' SQLCODE
        END-IF.
                ACCEPT WS-EOT.
200-PROCESS-EXIT
        EXIT.
```

SELCTING THE ROWS FROM THE TABLE WITHOUT USING CURSOR...

Note: Select query when used in application program fetches only one row from the table. If we try to fetch more than one row from the table we get -811 SQL Error Code.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SELPGM.
AUTHOR. SAHASRA.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
     EXEC SQL
       INCLUDE SQLCA
     END-EXEC.
     EXEC SQL
       INCLUDE EMPDCL
     END-EXEC.
01 WS-EMP-REC.
        05      WS-EMPID     PIC    X(5)     VALUE SPACES.
        05      WS-EFNAME    PIC    X(15)    VALUE SPACES.
        05      WS-ELNAME    PIC    X(15)    VALUE SPACES.
        05      WS-EMPDOB    PIC    X(10)    VALUE SPACES.
        05      WS-EMPSAL    PIC    S9(5)V99 VALUE ZEROS.
        05      WS-EMPDEPT   PIC    X(5)     VALUE SPACES.
PROCEDURE DIVISION.
000-MAIN-RTN.
        PERFORM 200-PROCESS-RTN THRU 200-PROCESS-EXIT UNTIL OPTION = 'YES'.
        STOP RUN.
200-PROCESS-RTN.
        ACCEPT WS-EMPID.
        MOVE WS-EMPID TO HS-EMPID.
        EXEC SQL
                SELECT EMPID,
                       EFNAME,
                       ELNAME,
                       EMPDOB,
                       EMPSAL,
                       EMPDEPT
```

```
            INTO   :HS-EMPID,
                   :HS-EFNAME,
                   :HS-ELNAME,
                   :HS-EMPDOB,
                   :HS-EMPSAL,
                   :HS-EMPDEPT
            FROM  EMPMAST
            WHERE EMPID = :HS-EMPID
      END-EXEC.
      MOVE  HS-EMPID      TO      WS-EMPID.
      MOVE  HS-EFNAME     TO      WS-EFNAME.
      MOVE  HS-ELNAME     TO      WS-ELNAME.
      MOVE  HS-EMPDOB     TO      WS- EMPDOB.
      MOVE  HS-EMPSAL     TO      WS-EMPSAL.
      MOVE  HS-EMPDEPT    TO      WS-EMPDEPT.
      IF SQLCODE = 0
            DISPLAY       'EMPLOYEE NUMBER:'          WS-EMPID
            DISPLAY       'EMPLOYEE FNAME:'           WS-EFNAME
            DISPLAY       'EMPLOYEE LNAME:'           WS-ELNAME
            DISPLAY       'EMPLOYEE DATE OF BIRTH:'   WS- EMPDOB
            DISPLAY       'EMPLOYEE SALARY:'          WS-EMPSAL
            DISPLAY       'EMPLOYEE DEPT NUM:'        WS-EMPDEPT
      ELSE IF SQLCODE = 100
            DISPLAY       'ROW NOT FOUND:'
      ELSE
            DISPLAY       'SQLERROR, SQLCODE:'        SQLCODE
      END-IF
      END-IF.
      ACCEPT WS-EOT.
200-PROCESS-EXIT.
      EXIT.
```

PROGRAM TO UPDATE THE ROW IN EMPMAST TABLE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UPDPGM.
AUTHOR. SAHASRA.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
      EXEC SQL
            INCLUDE SQLCA
      END-EXEC.
      EXEC SQL
            INCLUDE EMPDCL
      END-EXEC.
01 WS-EMP-REC.
```

```
        05 WS-EMPID                 PIC X(5) VALUE SPACES.
        05 WS-NEW-EMPDEPT           PIC X(5) VALUE SPACES.
01 WS-EOT                           PIC X(3) VALUE 'NO'.
PROCEDURE DIVISION.
000-MAIN-RTN.
        PERFORM 200-PROCESS-RTN THRU 200-PROCESS-EXIT UNTIL WS-EOT = 'YES'.
        STOP RUN.
200-PROCESS-RTN.
        ACCEPT      WS-EMPID.
        ACCEPT      WS-NEW-EMPDEPT.
        MOVE  WS-EMPID          TO    HS-EMPID.
        MOVE  WS-NEW-EMPDEPT    TO    HS-EMPDEPT.
        EXEC SQL
                UPDATE EMPMAST
                        SET EMPDEPT = :HS-EMPDEPT
                        WHERE EMPID= :HS-EMPID
        END-EXEC.
        IF SQLCODE = 0
                DISPLAY 'UPDATE SUCCESSFUL'
        ELSE
                DISPLAY 'UPDATE ERROR, SQLCODE:' SQLCODE
        END-IF.
        ACCEPT WS-EOT.
200-PROCESS-EXIT.
        EXIT.
```

DELETING THE ROW FROM EMPMAST TABLE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DELPGM.
AUTHOR. SAHASRA.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
        EXEC SQL
                INCLUDE SQLCA
        END-EXEC.
        EXEC SQL
                INCLUDE EMPDCL
        END-EXEC.
01      WS-EMP-REC.
        05      WS-EMPID    PIC    X(5)    VALUE SPACES.
01      WS-EOT              PIC    X(3)    VALUE 'NO'.
PROCEDURE DIVISION.
000-MAIN-RTN.
        PERFORM 200-PROCESS-RTN THRU 200-PROCESS-EXIT UNTIL WS-EOT = 'YES'.
        STOP RUN.
```

```
200-PROCESS-RTN.
        ACCEPT WS-EMPID.
        MOVE  WS-EMPID TO HS-EMPID.
        EXEC SQL
                DELETE FROM EMPMAST
                WHERE EMPID = :HS-EMPID
        END-EXEC.
        IF SQLCODE = 0
                DISPLAY 'DELETE SUCCESSFUL'
        ELSE
                DISPLAY 'DELETE ERROR, SQLCODE:' SQLCODE
        END-IF.
        ACCEPT WS-EOT.
 200-PROCESS-EXIT
        EXIT.
```

PROGRAM TO COPY THE RECORDS FROM EMPFILE TO EMPMAST TABLE

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FILTAB.
AUTHOR. SAHASRA.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT EMPFILE ASSIGN TO EMPDD
     ORGANIZATION IS SEQUENTIAL
     ACCESS MODE IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD EMPFILE.
01 EMP-REC.
        05      FL-EMPID        PIC     X(5).
        05      FL-EFNAME       PIC     X(15).
        05      FL-ELNAME       PIC     X(15).
        05      FL-EMPDOB       PIC     X(10).
        05      FL-EMPSAL       PIC     S9(5)V99.
        05      FL-EMPDEPT      PIC     X(5).
 WORKING-STORAGE SECTION.
        EXEC SQL
                INCLUDE EMPDCL
        END-EXEC.
        EXEC SQL
                 INCLUDE SQLCA
        END-EXEC.
 01     WS-EOF          PIC     X(3)    VALUE 'NO'.
 PROCEDURE DIVISION.
 000-MAIN-RTN.
```

```
        PERFORM 100-OPEN-RTN      THRU  100-OPEN-EXIT.
        PERFORM 200-PROCESS-RTN THRU  200-PROCESS-EXIT UNTIL WS-EOF = 'YES'.
        PERFORM 300-CLOSE-RTN     THRU  300-CLOSE-EXIT.
        STOP RUN.
100-OPEN-RTN.
        OPEN   INPUT  EMPFILE.
100-OPEN-EXIT.
        EXIT.
200-PROCESS-RTN.
        READ   EMPFILE
        AT END   MOVE 'YES' TO WS-EOF
        NOT AT END
        PERFORM 250-MOVE-AND-INSERT-PARA THRU 250-MOVE-AND-INSERT-EXIT.
        END-READ.
200-PROCESS-EXIT.
        EXIT.
250-MOVE-AND-INSERT-PARA.
        MOVE FL-EMPID       TO HS-EMPID.
        MOVE FL-EFNAME      TO HS-EFNAME.
        MOVE FL-ELNAME      TO HS-ELNAME.
        MOVE FL-EMPDOB      TO HS-EMPDOB.
        MOVE FL-EMPSAL      TO HS-EMPSAL.
        MOVE FL-EMPDEPT     TO HS-EMPDEPT.
     EXEC SQL
       INSERT INTO EMPMAST(EMPID,
                           EFNAME,
                           ELNAME,
                           EMPDOB,
                           EMPSAL,
                           EMPDEPT)
                    VALUES(:HS-EMPID,
                           :HS-EFNAME,
                           :HS-ELNAME,
                           :HS-EMPDOB,
                           :HS-EMPSAL,
                           :HS-EMPDEPT);
     END-EXEC.
     IF SQLCODE NOT EQUAL TO ZERO
       DISPLAY 'ERROR IN INSERT: SQLCODE:' SQLCODE
     END-IF.
250-MOVE-AND-INSERT-EXIT.
        EXIT.
300-CLOSE-RTN.
     CLOSE EMPFILE.
300-CLOSE—EXIT.
        EXIT.
```

CURSORS:

If a Select Statement in an application program retrieving more than one row, then the program will get abend with the **SQL CODE -811**

In order to overcome this problem we have a concept of cursors in DB2

CURSOR:

- It is a pointer to the resultant table of a select statement.
- Cursor will fetch the data one row at a time into the application program.

There are FOUR Steps involved in Use of Cursors,

SQL STATEMENT	DESCRIPTION	COBOL EQUIVALENT
DELCARE Cursor	Defines a result table and names a cursor for it	None
OPEN Cursor Name	Creates the result table and positions the cursor before first row in the table	OPEN File Name
FETCH Cursor Name	Fetches the next row from the table	READ File Name
CLOSE Cursor Name	Closes the result table	CLOSE File Name

DECLARE CURSOR:

Syntax:

```
EXEC SQL
        DECLARE EMPCUR CURSOR FOR
        SELECT EMPID,EFNAME,EMPSAL,EMPDEPT FROM EMPMAST
END-EXEC.
```

- For the select statement which is using the application program and which is retrieving more than one row.
- Cursors generally will declare in working storage section, we can declare in procedure division also.

OPEN CURSOR:

Syntax:

```
EXEC SQL
        OPEN EMPCUR--→ CURSOR NAME
END-EXEC.
```

- Open cursor operation has to do in procedure division.
- Once the open cursor operation has been executed the select queries in the cursor has been executed and create the resultant table then set the point to first row of the resultant table.
- Open cursor is the executable operation.

FETCH CURSOR:

Fetch operation will fetch the first row of the resultant table into the application program.

Then we need to process the data by keeping the fetch operation in a loop.

Syntax:

```
EXEC SQL
        FETCH EMPCUR
        INTO(:HS-EMPID,
        :HS-EFNAME,
        :HS-EMPSAL,
        :HS-EMPDEPT)
END-EXEC.
```

CLOSE CURSOR:

- Once the processing of data is completed then we need to close the cursor.

Syntax:

```
EXEC SQL
        CLOSE EMPCRS
END-EXEC.
```

- If we are issuing any COMMIT in the application program the cursor will be closed automatically.

PROGRAM TO COPY RECORDS FROM TABLE TO FILE USING CURSORS:

```
IDENTIFICATION DIVISION.
PRORAM-ID. CURTABFL.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT EMPFILE ASSIGN TO EMPDD.
DATA DIVISION.
FILE SECTION.
FD EMPFILE
        01      FL-EMP-REC.
                05      FL-EMPID        PIC     X(5).
                05      FL- EFNAME      PIC     X(15).
                05      FL- EMPSAL      PIC     9(5)V99.
                05      FL- EMPDEPT     PIC     X(5).
WORKING-STORAGE SECTION.
        EXEC SQL
                INCLUDE SQLCA
        END-EXEC.
        EXEC SQL
                INCLUDE EMPDCL
        END-EXEC.
        EXEC SQL
          DECLARE EMPCUR CURSOR FOR
          SELECT EMPID,EFNAME,EMPSAL,EMPDEPT FROM EMPMAST
      END-EXEC.
PROCEDURE DIVISION.
MAIN-PARA.
        PERFORM 100-OPEN-PARA    THRU 100-OPEN-EXIT.
        PERFORM 200-PROCESS-PARA THRU 200-PROCESS-EXIT UNTIL SQLCODE = 100.
```

```
        PERFORM 300-CLOSE-PARA   THRU 300-CLOSE-EXIT.
        STOP RUN.
100-OPEN-PARA.
        OPEN   OUTPUT  EMPFILE.
        EXEC SQL
                OPEN EMPCUR
        END-EXEC.
        EVALUATE SQLCODE
        WHEN 0
                DISPLAY 'OPEN CURSOR SUCCESSFULLY'
        WHEN OTHER
                DISPLAY 'CURSOR OPEN ERROR'
        END-EVALUATE.
100-OPEN-EXIT.
        EXIT.
200-PROCESS-PARA.
        EXEC SQL
                FETCH EMPCUR
                INTO(:HS-EMPID,:HS-EFNAME,:HS-EMPSAL,:HS-EMPDEPT)
        END-EXEC.
        EVALUATE SQLCODE
        WHEN 0
        PERFORM 250-WRITE-PARA THRU 250-WRITE-EXIT
        WHEN 100
                DISPLAY 'FETCH COMPLETED'
        WHEN OTHER
                DISPLAY 'FETCH ERROR,SQLCODE:' SQLCODE
        END-EVALUATE.
200-PROCESS-EXIT.
        EXIT.
250-WRITE-PARA.
        MOVE    HS-EMPID        TO      FL-EMPID.
        MOVE    HS-EFNAME       TO      FL- EFNAME.
        MOVE    HS-EMPSAL       TO      FL- EMPSAL.
        MOVE    HS-EMPDEPT      TO      FL- EMPDEPT.
        WRITE EMP-REC.
250-WRITE-EXIT
        EXIT.
300-CLOSE-PARA.
        CLOSE EMPFILE.
        EXEC SQL
                CLOSE   EMPCUR
        END-EXEC.
300-CLOSE-EXIT.
        EXIT.
```

- While updating the data by using the cursors if we issue a commit after updating of some records the cursor will be automatically closed.
- In order to keep open the cursor even if we issue commit we have to declare the cursor with the **WITHHOLD** option.

PROGRAM TO UPDATE TABLE USING CURSORS:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UPDCUR.
DATA DIVISION.
WORKING-STORAGE SECTION.
        EXEC SQL
                INCLUDE SQLCA
        END-EXEC.
        EXEC SQL
                INCLUDE EMPDCL
        END-EXEC.
        EXEC SQL
                DECLARE EMPCUR CURSOR WITH HOLD FOR
                SELECT EMPID,EFNAME,EMPSAL,EMPDEPT FROM EMPMAST FOR
                        UPDATE OF EMPSAL
                END-EXEC.
        01      WS-COUNT    PIC     9(2)    VALUE ZEROS.
        PROCEDURE DIVISION.
        MAIN-PARA.
                PERFORM 100-OPEN-PARA    THRU 100-OPEN-EXIT.
                PERFORM 200-PROCESS-PARA THRU 200-PROCESS-EXIT UNTIL SQLCODE = 100.
                PERFORM 300-CLOSE-PARA   THRU 300-CLOSE-EXIT.
                STOP RUN.
        100-OPEN-PARA.
                EXEC SQL
                 OPEN EMPCUR
                END-EXEC.
                EVALUATE SQLCODE
                        WHEN 0
                        DISPLAY 'OPEN CURSOR SUCCESSFULLY'
                        WHEN OTHER
                        DISPLAY 'CURSOR OPEN ERROR, SQLCODE:' SQLCODE
                END-EVALUATE.
        100-OPEN-EXIT.
                EXIT.
200-PROCESS-PARA.
                EXEC SQL
                        FETCH EMPCUR
                        INTO(:HS-EMPID,:HS-EFNAME,:HS-EMPSAL,:HS-EMPDEPT)
                END-EXEC.
        EVALUATE SQLCODE
```

```
            WHEN 0
            PERFORM 250-UPDATE-PARA THRU 250-UPDATE-EXIT
            WHEN 100
            DISPLAY 'FETCH COMPLETED'
            WHEN OTHER
            DISPLAY 'FETCH ERROR, SQLCODE:' SQLCODE
     END-EVALUATE.
200-PROCESS-EXIT.
            EXIT.
250-UPDATE-PARA
            COMPUTE HS-EMPSAL = HS-EMPSAL + 5000.
            ADD 1 TO WS-COUNT
            EXEC SQL
                  UPDATE EMPMAST SET EMPSAL = :HS-EMPSAL      WHERE
                  CURRENT OF EMPCUR
            END-EXEC.
            EVALUATE SQLCODE
            WHEN 0
            PERFORM 280-COMMIT-PARA THRU 280-COMMIT-EXIT.
            WHEN OTHER
            DISPLAY 'UPDATE ERROR, SQLERROR:' SQLERROR
            END-EVALUATE.
250-UPDATE-EXIT.
            EXIT.
280-COMMIT-PARA.
            IF WS-COUNT = 5
                  EXEC SQL
                    COMMIT
                  END-EXEC
                  MOVE ZEROS TO WS-COUNT
            END-IF.
280-COMMIT-EXIT.
            EXIT.
300-CLOSE-PARA.
      EXEC SQL
          CLOSE EMPCRS
      END-EXEC.
300-CLOSE-EXIT.
      EXIT.
```

ADDITIONAL PROGRAM:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. RPTPGM.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
```

```
        SELECT RPT-FILE ASSIGN TO RPTDD.
DATA DIVISION.
FILE SECTION.
FD RPT-FILE.
        01      RPT-REC         PIC     X(135).
WORKING-STORAGE SECTION.
        EXEC SQL
                INCLUDE EMPDCL
        END-EXEC.
        EXEC-SQL
                INCLUDE SQLCA
        END-EXEC.
        EXEC SQL
                DECLARE EMPCRS CURSOR FOR
                SELECT EMP_NO,EMP_NAME,EMP_DEPT,EMP_SAL FROM EMPTAB
                                WHERE EMP_SAL > 1500
        END-EXEC.
        01      HEADER-1.
                05      FILLER          PIC     X(32)   VALUE  SPACES.
                05      SUB-HEA-1       PIC     X(16)   VALUE  'EMPLOYEE DETAILS'.
                05      FILLER          PIC     X(32)   VALUE  SPACES.
        01      HEADER-2.
                05      FILLER          PIC     X(2)    VALUE  SPACES.
                05      SUB-HEA-2.1     PIC     X(15)   VALUE  'EMPLOYEE NUMBER'.
                05      FILLER          PIC     X(3)    VALUE  SPACES.
                05      FILLER          PIC     X(4)    VALUE  SPACES.
                05      SUB-HEA-2.2     PIC     X(13)   VALUE  'EMPLOYEE NAME'.
                05      FILLER          PIC     X(3)    VALUE  SPACES.
                05      FILLER          PIC     X(4)    VALUE  SPACES.
                05      SUB-HEA-2.3     PIC     X(13)   VALUE  'EMPLOYEE DEPT'.
                05      FILLER          PIC     X(5)    VALUE  SPACES.
                05      FILLER          PIC     X(4)    VALUE  SPACES.
                05      SUB-HEA-2.4     PIC     X(12)   VALUE  'EMPLOYEE SAL'.
                05      FILLER          PIC     X(59)   VALUE  SPACES.
        01      HEADER-3.
                05      FILLER          PIC     X(80)   VALUE ALL '-'.
                05      FILLER          PIC     X(55)   VALUE  SPACES.
        01      WS-REC.
                05      FILLER          PIC     X(07)   VALUE  SPACES.
                05      WS-EMP-NO       PIC     9(5).
                05      FILLER          PIC     X(06)   VALUE  SPACES.
                05      FILLER          PIC     X(02)   VALUE  SPACES.
                05      WS-EMP-NAME PIC X(15).
                05      FILLER          PIC     X(3)    VALUE  SPACES.
                05      FILLER          PIC     X(5)    VALUE  SPACES.
                05      WS-EMP-DEPT PIC X(10).
```

```
        05      FILLER          PIC     X(5)    VALUE   SPACES.
        05      FILLER          PIC     X(4)    VALUE   SPACES.
        05      WS-EMP-SAL      PIC     9(5).
        05      FILLER          PIC     X(63)   VALUE   SPACES.
PROCEDURE DIVISION.
MAIN-PARA.
        PERFORM OPEN-PARA.
        PERFORM WRITE-HEADERS.
        PERFORM PROCESS-PARA UNTIL SQLCODE = 100.
        PERFORM CLOSE-PARA.
        STOP RUN.
OPEN-PARA.
        OPEN OUTPUT RPT-FILE.
EXEC SQL
        OPEN EMPCRS
END-EXEC.
        WRITE RPT-REC FROM HEADER-1.
        WRITE RPT-REC FROM HEADER-3.
        WRITE RPT-REC FROM HEADER-2.
        WRITE RPT-REC FROM HEADER-3.
PROCESS-PARA.
        EXEC SQL
                FETCH EMPCRS
                INTO(:HS-EMP-NO,:HS-EMP-NAME,:HS-EMP-DEPT,:HS-EMP-SAL)
        END-EXEC.
        EVALUATE SQLCODE
        WHEN 0
        PERFORM WRITE-PARA
        WHEN OTHER
        DISPLAY 'FETCH ERROR'
        END-EVALUATE.
        WRITE-PARA.
                MOVE  HS-EMP-NO         TO WS-EMP-NO.
                MOVE  HS-EMP-NAME       TO WS-EMP-NAME.
                MOVE  HS-EMP-DEPT       TO WS-EMP-DEPT.
                MOVE  HS-EMP-SAL        TO WS-EMP-SAL.
        WRITE RPT-REC FROM WS-REC.
CLOSE-PARA.
        EXEC SQL
                CLOSE EMPCRS
        END-EXEC.
```

LOAD JOB, UNLOAD JOB & REPAIR JOB

LOAD JOB

```
//LOAD1       EXEC   PROC=DSNUPROC,PARM='DBTG'
//              INCLUDE MEMBER=X →It Holds Some Libraries In It Like JOBLIB Etc.
//SORTWK01  DD     UNIT=SYSDA,DACLAS=DC010
//SYSPRINT    DD     DSN=FILENAME,
//                   DISP=(NEW,CATLG,DELETE),
//                   DATACLAS=DC010,
//                   RECFM=FB,LRECL=130,
//                   DSORG=PS
//SYSREC00    DD     DSN=STTW.BCP.SYSREC00.UNLD.XXXX,DISP=SHR
                     :
                     :
//SYSRE30     DD     DSN=STTW.BCP.SYSREC30.UNLD.XXXX,DISP=SHR
//SYSIN       DD     DSN=STTW.BCP.SYSREC30.UNLD.RSN,DISP=SHR
/*
//
```

UNLOAD JOB

```
// STEP0020   EXEC   PGM=IKJEFT01,       DYNAMBR=20
//                   INCLUDE=DB2STP
//SYSTSPRT    DD     SYSOUT=*
//SYSTSIN     DD     *
        DSN    SYSTEM(DBAG)
        RUN    PROGRAM(DSNTIAUL)
        PLAN(DSNTIB41)-
        PARMS('SQL' )-
        LIB('DSNDBAG.DSN.RUNLIB.LOAD')
/*
//SYSPRINT    DD     DSN=&HLQ.BBD.SYSPRINT.UNLD.RSN,
//                   DISP=(,CATLG,DELETE),
//                   DATACLASS=DC010,
//                   RECFM=FB,LRECL=121
//                   DSORG=PS
//SYSUDUMP  DD     SYSOUT=*
//SYSPUNCH    DD     DSN=&HLQ.BCP.SYSPUNCH.UNLD.RSN,
//                   DISP=(,CATLG,DELETE),
//                   DATACLAS=DC010,
//                   RECFM=FB,LRECL=80,
//                   DSORG=PS
//SYSREC00    DD     DSN=&HLQ.BCP.SYSREOO.UNLD.DCLADADC,
//                   DISP=(,CATLG,DELETE),
//                   DATACLAS=DC010,
//                   DSORG=PS
                        :
```

```
                              :
//SYSREC30    DD      DSN=&HLQ.BCP.UNLD.DCLADAUT,
//                    DISP=(,CATLG,DELETE),
//                    DATACLAS=DC010,
//                    DSORG=PS
//SYSIN               DD      *
        SELECT * FROM GATWBDA.BD-ACQ-DISP-CASE WITH UR;
                              :
                              :
                              :
        SELECT * FROM GATWBDA.BD-AUTH-LOG
                WHERE AUTH-DATE > '2009-12-31' WITH UR;
/*
//
```

SYSPUNCH HOLDS ALL TABLE STRUCTURTS.

SYSPUNCH: FILE

Load data log no replace INDDN SYSREC00 into table GTTWBDA.BD-ACQ-DISP-CASE
(CASE-NBR POSITION(1) CHAR(11),CASE-DATE POSITION(12)
DATE EXTERNAL(10),.....

REPAIR JOB

```
//STEP020     EXEC    DSNUPROC,PARM='DBPG,YHBDRDA',
//SORTOUT     DD      DSN=&&SORTOUT,UNIT=SYSDA,DATACLAS=DC100
//SORTWK01    DD      UNIT=SYSDA,DATACLAS=DC&SPLG
//SYSREC      DD      DSN=&&SYSREC,UNIT=SYSDA,DATACLAS=DC&SPL0
                      DCB=BUFNO=20
//SYSPRINT    DD      SYSOUT=*
//SYSUDUMP    DD      SYSOUT=*
//SYSOUT      DD      SYSOUT=*
//SYSIN               DD      *
        REPAIR SET    TABLESPACE   DB.TS   NOCOPYPEND
        REPAIR SET    TABLESPACE   DB.TS   NOCOPYPEND
        REPAIR SET    TABLESPACE   DB.TS   NOCOPYPEND
/*
//
```

- SYSPUNCH holds structures of all tables in SYSIN DD *
- SYSREC Should start with SYSREC00 and the DD name has to be gone in ascending order.
- When the load job acts return code as 04, (i.e., get abended with -904 (resource unavailable)) that indicates copy pending status.
- In sysprint / spool we get the table for which it went into copy pending status and we have its related database and tablespace name in the spool.
- Take the database and tablespace name and run the repair job against database and tablespace.
- When the index is not available i.e., resource unavailable during batch jobs run then we use rebuild utility for index.

IMPORTANT SQL ERROR CODES

000 Successful Operation on table.
+100 End of the table or row not found.
-180 While inserting invalid date or time format.
-204 The table specified in operation is not defined in DB2.
-205 Column specified is not defined in DB2.
-305 Indicator (null) variable is not defined in application program.
-501 The cursor specified in fetch or close is not opened.
-502 The cursor specified in open statement is already open.
-503 The column which is updating is not defined in the cursor at for update of clause.
-504 The cursor name is not defined.
-811 In an application program a select statement retrieve more than one row, then we will get this error code.
-818 The timestamp between load module and application plan mismatch, while executing application program.
-805 The Plan/Package not found in specified dataset.
-904 Resource unavailable (i.e. table not find in DB2).
-911 Dead lock(Means if one JOB is read or update the table at the same time another JOB also want to access that table if won't possible then the second JOB will wait some time and abend with the -911 SQL code).
-922 Authorization fails while trying to access any unauthorized resource.
-530 When we are inserting a row (with foreign key value) independent table if there is no value (row) present in the parent table (primary key value) we will get -530.
-531 When we are updating a row in parent table we cannot change the primary key value, if the rows present in a dependent table as foreign key.
-532 When we are deleting a row in parent table, without deleting the row in dependent table we will get -532.
-208
```
SELECT EMPNO, EMP-NAME, EMP-SAL
    FROM EMP-TAB
    ORDER BY EMP-NO = 111
```

- If the order by clause EMP-NO and select clause EMP-NO not matched then the error is occurred.

SAHASRA INFOTECH
&
CONSULTING SERVICES

C I C S
(Customer Information Control System)

www.sahasrainfotech.co.in

Customer Information Control System

BATCH SYSTEM AND ONLINE SYSTEM

Characteristics of Batch and Online Systems

There are two types of computer application systems. One is the batch application system and the other is the online application system. The batch system has a system environment where jobs run one by one in a conventional way, whereas the online system has the system environment where many transactions run concurrently.

	Batch System	Online System
Data Collection	Off-Line	On-Line
Input	Sequential in batch	Random, Concurrent
Job Schedule	At specific interval	Instantaneous
Resources	Not sharable	Sharable
Response Time	Not critical	Critical
Output	Printed reports, Output files, User must wait for batch jobs to produce reports	To terminal, Updated files, system log, Instant feedback
Security	Simple	Complex
Recovery	Simple	Complex
Information	Not Always Current	Always Current
Updation	In Batch	Immediate

Batch System **Vs** Online System

Advantages of the Online System

These days, the online system is so common that virtually every large mainframe installation has at least one online application system. This is because the online system has advantages over the batch systems.

- Up-to-date file (information) can be shared among many users simultaneously and instantly.
- Data validation and editing can be done at the data entry time.

Advantages of the Batch System

On the other hand, in spite of the ever-increasing popularity of the online system, the batch system has not died out, and it still contributes to the large portion of data processing requirements. This is because the batch system has advantage over the online system. Some of the advantages of the batch system are as follows:

- Certain information does not have to be updated or displayed on a real-time base. Users can wait until the next day, the end of a week, or the end of a month. In this case, the batch system is sufficient for the purpose.
- If massive file updates or lengthy calculations are involved, the batch system should be used because the online processing in these areas tends to be very costly in terms of resource consumption.

Brief History of CICS

The advantages of the online system were first recognized in the late 1950s and early 1960s by such businesses as the airline and banking industries, which pioneered the development of the large-scale online systems. Since then, online systems have become explosively popular among all industries.

Prior to CICS

Until the late 1960s, all online systems were developed on a custom-made basis for each application for each installation. However, developing an online system is a complex project involving the operating system (OS), telecommunication access methods, data access methods, and applications programs.

Therefore, instead of developing an online system from scratch each time, the database/data communication (DB/DC) control system was developed in order to provide the control functions of the online system environment. Under a DB/DC control system, an application program can concentrate on the application processing, being free from considerations of OS, hardware, etc., which are not really the interest of the application programs. Developing an online application system under the DB/DC control system as a result became significantly easier and faster.

Early CICS (Macro Level)

The Customer Information Control System (CICS) as developed by IBM in the late 1960s as a DB/DC control system. The initial version was the macro level CICS, under which, as the name "macro" suggests, the application programs used the assembler-macro type CICS macros for requesting CICS services. Under the CICS macro level, application development became significantly easier. However, it was still cumbersome work which required special skills.

CICS Command Level

Over time, CICS was constantly upgraded and functionally enhanced. One significant upgrade was from macro level to the command level. CICS commands are high-level language version of CICS macros, in a sense that one CICS command achieves a CICS service which would have been achieved by a series of CICS macros. Therefore, under CICS command level, application development became much easier than under the CICS macro level.

Current CICS Family

There are currently five CICS products available, each of which is developed for particular operating system. These products are as follows:

Product	Operating System
CICS / MVS Version 2 Release 1	MVS / XA, MVS / ESA
CICS / OS / VS Version 1 Release 7	MVS, MVS / XA
CICS / DOS / VS Version 1 Release 7	VSE
CICS / VM Release 2	VM / SP
CICS OS/2	OS/2

Functionally, all of these CICS products are compatible with each other, with certain exceptions caused by the differences among the corresponding operating systems.

CICS / MVS

CICS/MVS version 2 Release 1 is the current version of the mainstream CICS product, replacing its older version, CICS/Os/VS Version 1 Release 7. It runs under the MVS/XA operating system and provides comprehensive services as a general purpose DB/DC control system in the virtual environment.

Features of CICS

1. DB / DC

CICS makes data communication much easier by supplying all the basic components needed to handle data communication. This allows system designer and programmers to concentrate on developing application programs without having to concern with the details of data transmission, buffer handling or the properties of individual terminal devices.

2. Multi Programming

As far as the operating system is concerned, CICS is an application program that is it runs as a job in one of the system's partition or region, which is a virtual address space. Therefore CICS takes part in multiprogramming environment of the operating system. CICS is a longer running job therefore it remains up during the day time collecting on-line transactions and at night batch jobs are executed for master file updation etc.

3. Multi Tasking

Multitasking means that the operating system (OS) allows more than one task to be executed concurrently, regardless of whether the tasks use the same program or different programs. Therefore, this is not a unique concept of CICS. But, CICS manages multitasking of CICS task within its own region. That is CICS provides a multitasking environment where more than one CICS task run concurrently.

4. Scheduling

When user requests a transaction, normally by logging on and keying in a transaction code, CICS checks the status of the user and the terminal. This ensures the validity of user, terminal and transaction-id. Then task control creates a task for the transaction. CICS tries to give the best response times to the most important or urgent work. Usually, several tasks are competing for resources, so a transaction, an operator, and a terminal are each assigned a priority related to the importance of the function they carry out. CICS sums these priorities to give the overall priority of the task and uses this priority to decide the order in which to process competing tasks. Since transactions are not normally processed through to completion in a single, uninterrupted option, CICS makes such decisions every time a task returns control to CICS.

SYSTEM COMPONENTS

The CICS / MVS System consists of

- Management Modules (Control Programs).
- Tables (Control tables).
- Control Blocks.

Management modules are the CICS/MVS programs that interface between the operating system and the application programs. Each management module performs a particular function. For example, when an application program issues a request to read record, the File Control Program management module processes the request. Input/output requests are made to CICS/MVS, instead of the operating system as they are in batch processing environment. CICS takes over the difficult part of I/O operations, leaving only the execution logic to the application program.

Tables define the CICS/MVS system's environment. The tables are functionally associated with the management modules. For example all file definitions are in the File Control Table. So definition may be shared by all application programs and tasks. For this reason, the files are not defined in the application program as they are in a batch program. The Terminal Control Table defines each terminal in the network; thus the application program need not be concerned with the physical attributes of various terminals in the system. Not all management modules have associated tables.

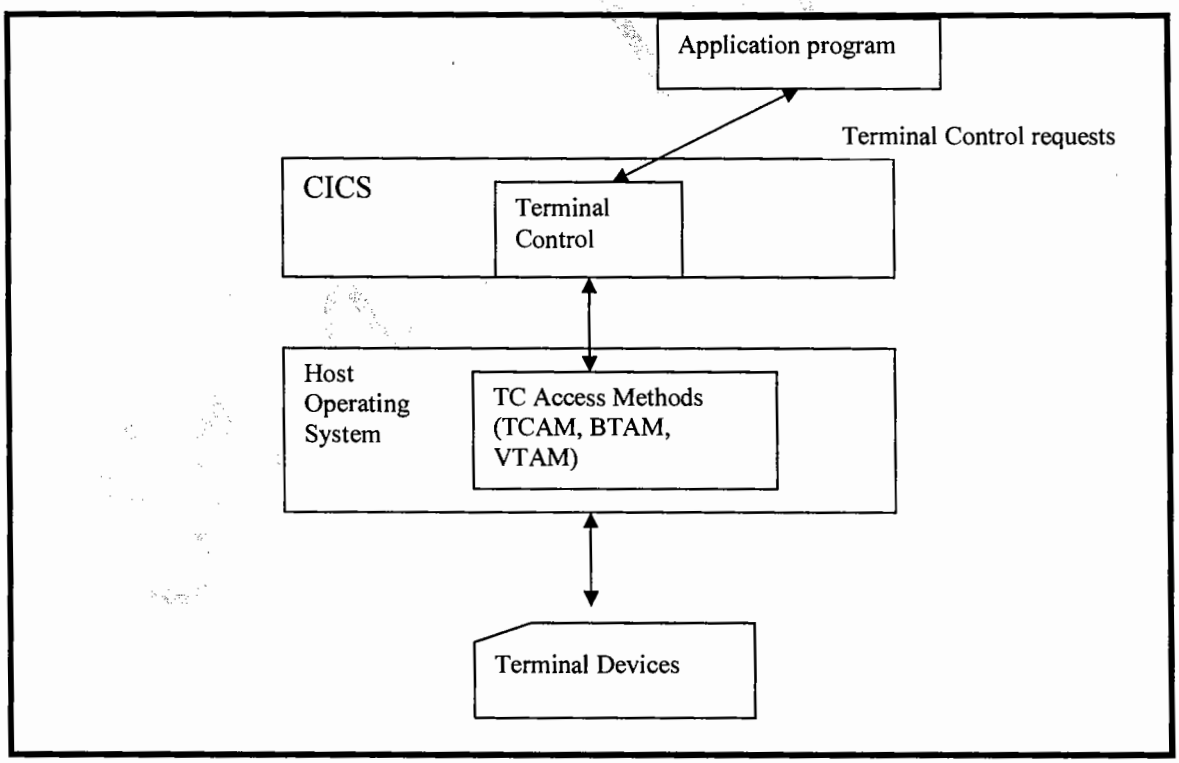
Control Blocks contain system type information. When a transaction is initiated a control block called a Task Control Area contains information pertinent to the task. For example Task Control area contains pointers to the application program and to the terminal's entry in the Terminal Control Table.

Terminal Control Program Management Module (TCP)

Terminal Control Program (TCP) is the management module that provides communications interface to the application program. The application programs invoke TCP which then controls communication in the network.

The Terminal Control Program performs the following functions:

- Requests terminal to send their transactions
- Transfers data between the application program and the terminal. All data transfers are requested by the application program but are executed by TCP.
- Handles hardware communications requirements. TCP is responsible for the actual transmission of a message over communication lines.
- Searches the Terminal Control Table (TCT). Each terminal in the network is identified by a unique entry in the Terminal Control Table. The entry is called Terminal Control Table Terminal Entry (TCTTE). When data is ready to be sent to the terminal, a flag is set ON in the terminal's TCTTE. TCP searches the TCT periodically and causes the data to be sent to the respective terminals.
- Requests initiation of a new task.
- All functions carried out by TCP are transparent to application program. Programmer simply needs to code RECEIVE and SEND commands.



Terminal Control Program Management Module

Task Control Program Management Module (KCP)

Task control keeps control of the status of all CICS tasks. Many of them will be processed concurrently and task control allocates processor time among them. This is called multitasking.

When an operator requests a transaction, normally by logging on and keying in a transaction code, CICS checks the status of the operator and the terminal. This ensures that the operator is known to the system and that the transaction is valid for that user and the terminal. Task Control then creates a task for that transaction.

Transaction request can be made either by entering transaction-id at the terminal or by another CICS transaction. A transaction is processed up to an instruction involving input from a file or a terminal, for instance. Then while the transaction waits for its input, another waiting task begins or resumes execution.

Program Control Program Management Module (PCP)

Program Control program (PCP) is the CICS/MVS management module responsible for locating application programs and if necessary load them into storage for execution. It is also responsible for transferring control to them and returning control back to CICS when the application program have completed execution. PCP also facilitates one application program transferring control to another application program via LINK and XCTL commands.

The validity of transaction-id supplied by the user is checked from the Program Control Table by Program Control Program management module. In carrying out its program control functions, PCP makes use of a table called the Processing Program Table (PPT). There is an entry in the table for each application program along with its name, host language and address.

At the time a task requires a given application program, that program may or may not be in storage. There are three reasons why a program may already be in storage at the time it is needed by a task.

- It is a resident program; that is, it was loaded into storage when CICS was initiated
- Another task is using the program.
- Another task used the program and storage area was not needed for something else the program remains in storage.

PCP knows if there are any tasks currently using the same program, from the value of USE COUNTER in PPT. If USE COUNTER is zero, the storage area is made available for other purpose.

File Control Program Management Module (FCP)

The File Control Program (FCP) facilitates the accessing of files on a direct access basis. FCP provides command to READ, WRITE, REWRITE and DELETE records. It uses the standard access methods provided by IBM. It manages concurrent operations by preventing simultaneous updating of a given record.

In carrying out its file control functions, FCP makes use of a table called the File Control Table (FCT). FCT includes an entry for each file that is to be used during the operation of CICS. Each entry in the FCT contains all the descriptive information for the file that it represents, thereby freeing the application programmer from defining the physical organization and other file attributes. The application programmer simply refers to the symbolic name that identifies the file entry in the FCT.

Storage Control Program Management Module (SCP)

Storage Control acquires, controls, and frees dynamic storage during execution of a task. Maintains full control over the virtual storage. Controls requests for main storage.

TABLES

The tables define the resources that CICS controls. For example, Terminal Control table (TCT) defines the terminals in the network. File Control Table describes the data files the CICS application programs access.

Tables also define the operating environment. For example, limits may be defined for the number of transactions CICS handles concurrently in System Initialization Table (SIT).

The tables are brought into storage when CICS is initialized. Whenever an application program requires access to a resource, CICS checks the appropriate table to get the information necessary to process the request.

There are sixteen tables used to describe the CICS environment. PCT, PPT, SIT, TCT are required. Others are optional.

System Initialization Table

The System Initialization table (SIT) contains parameters used by the system initialization process. In particular, the SIT identifies (by suffix character) the version of the CICS system control programs and CICS tables that you have specified and that are to be loaded.

Program Control Table

The program control table (PCT) contains the control information to be used by CICS for identifying and initializing a transaction. This table is required by CICS to verify incoming requests to start transactions and to supply information about the transaction.

Processing Program Table

The processing program table (PPT) defines the programs and mapsets.

Terminal Control Table

The terminal control table (TCT) describes a configuration of terminals, logical units and other CICS systems.

File Control Table

The file control table (FCT) describes files that are processed by file control management module. The CICS system definition (CSD) file is defined in the FCT, in addition to your own files. The files defined in the FCT can be VSAM or BDAM.

Control Blocks

In execution, CICS is dynamic. To keep track of what is going on in the system CICS uses several different control blocks. Most of these control blocks are dynamically managed. They are created when needed and disposed of when no longer needed.

Primarily CICS management modules access the Control blocks. They provide various kinds of information. For example, a control block called Task Control Area (TCA) represents each transaction. Information about current task is stored in to control block named Execute Interface Block (EIB).

DATA COMMUNICATION OPERATION IN CICS

Three methods are available to the CICS application programmer for communicating with the terminals and logical units in the subsystems of the network that forms part of the CICS system. The methods dealt with are:

- Basic mapping support (BMS)
- Terminal Control
- Batch Data Interchange (BDI)

Basic Mapping Support provides commands and options that can be used to format data in a standard manner. BMS converts data streams provided by the application program to conform to the requirements of the devices. Conversely data received from a device is converted by BMS to a standard form.

Terminal Control is the basic method for communicating with devices; where as both BMS and batch data interchange extend the facilities of terminal control to simplify further the handling of data streams. Both BMS and BDI use terminal control facilities when invoked by an application program.

Batch Data Interchange provides commands and options that may be used possibly in conjunction with BMS commands, to communicate with batch logical units.

Basic Mapping Support (BMS)

Once a record requested by the operator is retrieved, it has to be formatted for display. BMS removes the responsibility of formatting the screen from the application program. BMS places the data with necessary control characters in the TIOA. The control characters are removed before the message is displayed on the screen.

It allows repositioning of fields without modifying application programs. BMS acts as an interface between the application program and Terminal Control Program.
BMS makes the application program Format Independent and Device Independent.
Format Independence:

- Application programs are written without concern for physical position of data fields within a display.
- In an application program symbolic labels are used to refer the fields.
- BMS relates the label to the actual position in the display.

Device Independence:

- Application programs can be developed without any concern for physical characteristics of the terminals.
- BMS prepares the message based on the terminal type being used.

BMS Maps

A Screen defined through BMS is called a "map". There are two types of maps: a physical map, which is primarily used by CICS, and a symbolic map, which is primarily used by the application programs.

BMS map is nothing but a program written in the Assembler Language. However, you do not have to program the BMS map as such, because a set of assembler macros (BMS macros) are provided by CICS for the BMS map coding.

Based on how a map or a group of maps is linkedited, there are two units of the map: map, which represents a BMS coding for a screen panel, and mapset, which represents a load module.

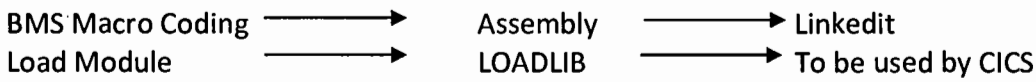
Physical Map and Symbolic Map

Physical Map

The primary objective of the physical map is to ensure the device independence in the application programs. More concretely, for input operations, the physical map defines maximal data length and starting position of each field to be read and allows BMS to interpret an input data stream. For output operations, the physical map defines starting position, length, field characteristics (Attribute Bytes) and default data for each field, and allows BMS to add control characters and commands for output in order to construct an output data stream.

Physical Map Generation

The physical map is a program in a form of load module. Therefore, the physical map is coded using BMS macros, assembled separately, and linkedited into the CICS load library



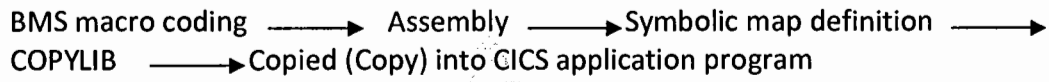
Symbolic Map

The primary objective of the BMS symbolic map is to ensure the device and format independence to the application programs. Therefore, through the symbolic map, a layout change in the formatted screen can be done independent of the application program coding as long as the field name and length remain same.

It is used by the application program which issues a COBOL COPY statement in order to include a symbolic map in the program.

Symbolic Map Generation

A symbolic map is a copy library member, which is to be included in the application program for defining the screen fields. Therefore, the symbolic map is coded using BMS macros, assembled separately, cataloged into a copy library (COPYLIB), and at the time of application program compile, it will be copied into the application program.



MAP AND MAPSET

Map

A representation of one screen format is called a map. One or group of maps makes up a mapset.

Mapset

A group of maps, which are link-edited together, is called a mapset. Each mapset must be registered in PPT, since CICS considers the BMS mapset as a program coded in Assembler. The mapset name consists of two parts as follows:

Generic Name : 1 to 7 characters
Suffix : 1 character

Application program uses only the generic name. The suffix is required at the mapset definition time in order to distinguish the device types if the same mapset is used for different types of terminals. While the application program uses only the generic name of the mapset name, CICS automatically inserts the suffix depending on the terminal in use, thereby ensuring the device independence to the application program.

Components of a Screen

CICS identifies screen into three components:

1. Mapset
2. Maps
    a. Physical Map
    b. Symbolic Map
3. Fields
    a. Unnamed (Literals)
    b. Named (Data Fields)
    c. Stopper

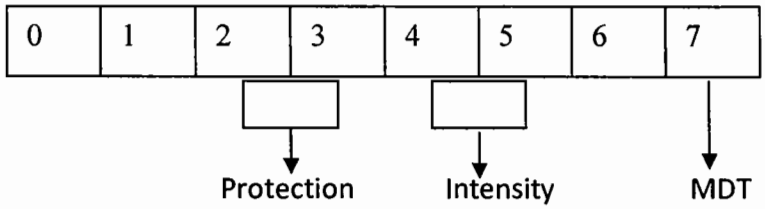
The first step in designing screen layout is to divide the screen into functional areas such as title area, application data area and message area. The title area of a screen should identify the program that displayed data. The application data area comprises the main portion of the screen. Three kinds of fields are usually found in this area:

1. Keyword / Unnamed / Literal – Contains constant data sent by program to identify the contents of the data field.
2. Data Field / Named – Contains data that the application program retrieves from files and displays. The data may appear exactly as stored in a file, or it may be edited, or it may be left blank for operator to enter data.
3. Stopper Field – On data entry screen restricts the length of the data field. Stopper field containing no data is used to stop operator from entering too many characters in a field.

Attribute Byte Format

The attribute character is always the first character of a field. It occupies a character position on the screen but appears as a blank.

Positions	Functions	Bit Settings	
0-1	Information about bits2 thru 7	-	
2-3	Protection	00	Unprotected, Alphanumeric
		01	Unprotected, Numeric
		10	Protected
		11	Autoskip
4-5	Intensity	00	Normal
		01	Normal
		10	Bright
		11	No-display
6	Must be 0	-	
7	MDT	0	Field has not been modified
		1	Field has been modified



Attributes bytes can convey the following field attributes.

- Field Protection
  - Unprotected: can enter any keyboard character into unprotected field.
  - Protected: Data cannot be entered in a protected field. If the operator attempts to enter data, the keyboard is locked.
  - Autoskip: An autoskip field is a protected field that automatically skips the cursor to the next unprotected field.
- Field Intensity
  - Normal: a normal intensity field displays the data at the normal operating intensity.
  - Bright: a bright intensity field displays the data at a brighter than normal intensity. This is often used to highlight keywords, errors, or operator messages.
  - Nodisplay: A nodisplay field does not display the data on the screen for operator viewing. This might be used to enter security data
- Shift

Modified Data Tag (MDT)

In order to understand the concept of data transfer from a terminal to the application program, it is important to know the concept of "Modified Data Tag" (MDT).

MDT is one bit of the attribute character. If it is off (0), it indicates that this field has not been modified by the terminal operator. If it is on (1), it indicates that this field has been modified by the operator. Only when MDT is on, will the data of the field be sent by the terminal hardware to the host computer (i.e., to the application program).

Effective use of MDT drastically reduces the amount of data traffic in the communication line, thereby improving performance significantly.

Following are the three ways to set MDT on:

- By data-entry
- By setting attribute byte in the physical map
- By moving MDT attribute in the application program.

MAP DEFINITION MACROS

Available Macros

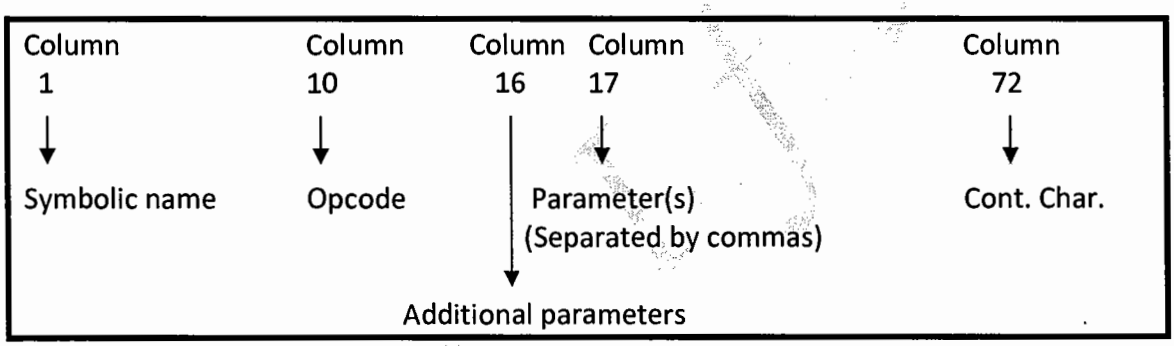
As a program must be coded using a programming language, a BMS map also must be coded. For this purpose, BMS provides Assembler macros as follows:

DFHMSD	:	To define a mapset.
DFHMDI	:	To define a map.
DFHMDF	:	To define a field.

General Format

The BMS map definition macros are purely Assembler macros. Therefore, the following coding must be maintained:

Format of BMS Macros



Example of BMS Macros

```
MENU    DFHMSD    TYPE=&SYSPARM,                X
                  MODE=INOUT, LANG=COBOL
```

Order of Macros

There is rule for the order of BMS macros which must be followed. That is, within one mapset definition, the map definition can be specified as many times as you wish. Within one map definition, the field definition can be specified as many times as you wish

BMS Macro Definition

```
Print Nogen          ----> Assembler Command
Label  DFHMSD        ----> Mapset Definition
Label  DFHMDI        ----> Map Definition
Label  DFHMDF        ----> Field Definition
   |
   |
      DFHMDF
      DFHMDI
      DFHMDF
   |
   |
      DFHMDF
      DFHMSD Type = Final ----> Mapset Definition
  End      ------------------------> Assembler Command
```

DFHMSD Macro

Function

The DFHMSD macro is used to define a mapset and its characteristics or to end a mapset definition. Only one mapset definition is allowed within one assembly run.

Format

```
Label  DFHMSD     [TYPE=DESECT | MAP | &SYSPARM | FINAL]
                  [MODE=IN | OUT | INOUT]
                  [LANG=ASM | COBOL | PLI]
                  [STORAGE=AUTO | BASE=name]
                  [CTRL=([FREEKB] , [ ALARM] , [ FRSET])]
                  [HIGHLIGHT=OFF | BLINK | REVERSE | UNDERLINE]
                  [VALIDN=MUSTFILL | MUSTENTER]
                  [TERM=TYPE | SUFFIX=n]
                  [TIOAPFX=YES | NO]
                  [DATA=FIELD | BLOCK]
```

Format of DFHMSD Macro

Commonly Used Options

The following are some of the commonly used options of the DFHMSD macro:

TYPE = To define the map type

DSECT	For symbolic map.
MAP	For physical map.
&SYSPARM	For symbolic as well as physical map.
FINAL	To indicate the end of the mapset coding.

MODE = To indicate input/output operation.

IN	For the input map.
OUT	For the output map.
INOUT	For input/output map.

LANG = To define the language of the application program.

STORAGE = AUTO To acquire a separate symbolic map area for each mapset.

TIOAPFX = YES To reserve the prefix space (12 bytes) for BMS commands to access TIOA properly.

CNTL= To define the device control requests.

FREEKB	To unlock the keyboard.
FRSET	To reset MDT to zero.
ALARM	To set an alarm at screen displaytime.
PRINT	To indicate the mapset to be sent to the printer.

TERM = Type Required if other than the 3270 terminal is used. This ensures device independence by means of providing the suffix.

End of Mapset Definition

One mapset definition must be ended with another DFHMSD macro with the TYPE=FINAL option as follows:

```
DFSMSD          TYPE=FINAL
```

DFHMDI Macro

Function

The DFHMDI macro is used to define a map and its characteristics in a mapset.

Format of DFHMDI Macro

```
Label   DFHMDI      [SIZE=(LINE , COLUMN)]
                    [LINE=number | NEXT | SAME]
                    [COLUMN=ASM | COBOL | PLI]
                    [STORAGE=AUTO | BASE=name]
                    [CTRL=([FREEKB], [ ALARM], [ FRSET])]
                    [HIGHLIGHT=OFF | BLINK | REVERSE | UNDERLINE]
                    [VALIDN=MUSTFILL | MUSTENTER]
                    [TERM=TYPE | SUFFIX=n]
                    [TIOAPFX=YES | NO]
                    [DATA=FIELD | BLOCK]
```

The LABEL must be specified as the symbolic name to the DFHMDI macro. SIZE (ll, cc) is used to define the size of the map by the line size (ll) and column size (cc). LINE and COLUMN indicates the starting position of the map in line and column numbers, respectively. In addition, the DFHMDI macro has the same options used in the DFHMSD macro, such as CNTL and TIOAPFX. If the DFHMDI macro has the same options specified in the DFHMSD macro, the options specified in the DFHMDI macro override the ones specified in the DFHMSD macro.

DFHMDF Macro

Function

The DFHMDF macro is used to define a field in map and its characteristics. The DFHMDF macro can be issued as many times as you wish within one DFHMDI macro.

Format of DFHMDF Macro

```
Label           DFHMDF      [POS=(Line | Column)]
                            [LENGTH=number]
                            [JUSTIFY=([LEFT | RIGHT])
                            [ATTRB=(ASKIP | PROT | UPROT,[ NUM]]
                            [BRT | NORM | DRK]
                            [IC],[ FSET]) ]
                            [HILIGHT=OFF | BLINK | REVERSE | UNDERLINE]
                            [VALIDN=([MUSTFILL | MUSTENTER])]
                            [OCCURS=number]
                            [INITIAL='value']
                            [PICIN='value']
                            [PICOUT='value']
```

If the field name is required, the name of the field (1 to 7 chars) must be specified as the symbolic name to the DFHMDF macro. POS=(line, column) indicates the starting position of the field in the line and column number including the attribute character. INITIAL defines the initial value of the field (if any). LENGTH indicates the length of the field excluding the attribute character. PICIN and PICOUT defines the PICTURE clauses of the symbolic map in COBOL, which is useful for numeric field editing.

Format of Symbolic Map

A symbolic description map is a source language data structure that the assembler or compiler uses to resolve source program references to fields in the map. Hence it must be copied into any application program that refers to fields in the map.
A COBOL program must contain a COBOL COPY statement for each symbolic map definition in working storage section.

The symbolic map starts with the 01 level definition of FILLER PIC X(12), which is the TIOA prefix created by TIOAPFX=YES of the DFHMSD macro, and this is required by BMS under the CICS command level.
When designing a map, names are assigned to fields that contain variable data. The symbolic map data structure contains extended versions of these fields, each one consisting of sub-fields. Each kind of sub-field has a different suffix.

NameL: The halfword binary (PIC S9(4) COMP) field. For the input field, the actual number of characters typed in the field will be placed by BMS when the map is received. For the output field, this is used for the dynamic cursor positioning.

NameF: Flag byte. For an input field, it will be X'80' if field has been modified but no data is sent (i.e., the field is cleared). Otherwise, this field is X'00'.

NameA: The attribute byte for both input and output fields.

NameI: The input data field. X'00' will be placed if no data in entered.

NameO: The output data field.

Example Of Symbolic Map Definition:

```
01 MORMAP11.
     02 FILLER                 PIC X(12).
     02 AMOUNTL COMP           PIC S9 (4).
     02 AMOUNTF                PIC X.
     02 FILLER REDEFINES AMOUNTF
        03 AMOUNTA             PICTURE X.
     02 AMOUNTI                PIC 9(6)V99.
     02 RATEL          COMP    PIC S9(4).
     02 RATEF                  PICTURE X.
     02 FILLER REDEFINES RATEF.
        03 RATEA               PICTURE X.
```

```
    02 RATEI                PIC V9(4)
    02 PAYMENTL COMP  PIC S9(4).
    02 PAYMENTF          PICTURE X.
    02 FILLER REDEFINES PAYMENTF.
      03 PAYMENTA        PICTURE X.
    02 PAYMENTI          PIC X(9).
    02 MESSAGEL  COMP  PIC S9(4).
    02 MESSAGEF          PICTURE X.
    02 FILLER REDEFINES MESSAGEF.
      03 MESSAGEA        PICTURE X.
    02 MESSAGEI          PIC X(79).
    02 ERRORL     COMP  PIC S9(4).
    02 ERRORF            PICTURE X.
    02 FILLER REDEFINES ERRORF.
      03 ERRORA          PICTURE X.
    02 ERRORI            PIC X(77).
    02 DUMMYL    COMP  PIC S9(4).
    02 DUMMYF            PICTURE X.
    02 FILLER REDEFINES DUMMF.
      03 DUMMYA          PICTURE X.
    02 DUMMYI            PIC X(1).
01 MORMAP10 REDEFINES MORMAP11
    02 FILLER            PIC X(12).
    02 FILLER            PICTURE X(3).
    02 AMOUNTO           PIC X(8)
    02 FILLER            PICTURE X(3).
    02 RATEO             PIC X(4).
    02 FILLER            PICTURE X(3).
    02 PAYMENTO          PIC ZZ,ZZ9.99.
    02 FILLER            PICTURE X(3).
    02 MESSAGEO          PIC X(79).
    02 FILLER            PIC X(3).
    02 ERRORO            PIC X(77).
    02 FILLER            PICTURE X(3).
    02 DUMMYO            PIC X(01)
```

CHARACTERISTICS OF CICS

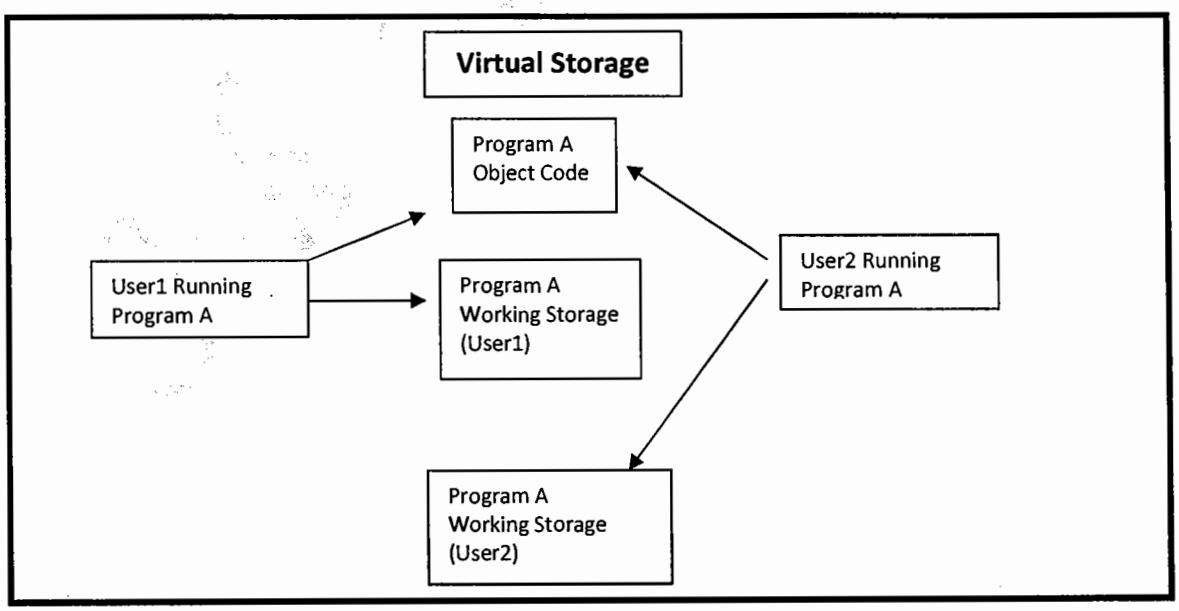
Multithreading

Multithreading is the system environment where the tasks are sharing the same program under the multitasking environment. Multithreading is a subset of multitasking, since it concerns tasks which use the same program. Under the multithreading environment, a program is shared by several tasks concurrently. For each task, the program must work as if it were executing instructions exclusively for each task. Therefore, it requires special considerations such as reentrancy.

Contrary to the multithreading environment, under the single threading environment, a program is used by only one job (or task) at a time. Single threading is the execution of a program from beginning to completion. Processing of one transaction is completed before another transaction is started. A typical example is a batch job. Although multithreading is not a unique concept to CICS, CICS manages multithreading of CICS tasks within its own region. That is, CICS provides the multithreading environment where more than one CICS task, which uses the same program, run concurrently.

Quasi-Reentrancy

In order to make multithreading possible, an application program must be "reentrant". A completely reentrant program doesn't change itself in any way i.e. a reentrant program cannot modify data in working storage. As a result, any user can enter a reentrant program at any point without affecting other users who are also running it. To make things easier, command level CICS allows you to use working storage by treating all programs as quasi-reentrant.



Quasi-Reentrant Program

Figure above shows how quasi-reentrant programs work under CICS, Here, two users are running the same application program, Program A. They share the same storage for the program's object code. I.e. the procedure division; but each is given a separate working storage area. That way, each can use working storage in the normal fashion. When you write a command level CICS programs, you don't need to worry about a quasi-reentrant, CICS handles that for you automatically.

Pseudo-Conversational Programming

An interactive program that waits for data to be input from the user on the terminal is called a Conversational program. On single user system this is not a problem. But with the multi-user system it is a problem. E.g. in a 30 users system, when one program is accepting and processing data, the other 29 are waiting taking up space in main storage. Hence in multi user environment, conversational programs can be tremendously inefficient. The solution Is Pseudo-conversational programming i.e. to return the control to CICS when the program is waiting for any input. As a result the program remains in storage only when it is processing data.

How a Pseudo-Conversational Program Works?

As the operator keys in data, it's displayed on the screen. That's a function of the terminal, and not of the program. After the operator has filled in all of the required data, the program needs to be reloaded to process it. But how does CICS know when to restart a pseudo-conversational program?

Basically, an operator signals CICS to restart a pseudo-conversational program by pressing one of the terminal's AID keys. (AID keys are the enter key, the clear key, the PF keys and the PA keys). Of course, most operators don't think in terms of restarting their program when they press an AID keys. Usually, an operator just knows that he presses specific AID key to get his program to do specific things. On the other hand, as a CICS programmer, you do need to think of an AID key as causing a program to be loaded and executed. A CICS program retrieves the data the operator entered, by issuing a CICS command: RECEIVE MAP. Command simply transfers data from the screen to the symbolic map in a program.

After the program retrieves the data the operator entered, it processes it. Next, the program issues a CICS SEND MAP command to display the results. After the program issues a SEND MAP command, it issues a RETURN command. The RETURN command ends the program just as a STOP RUN does in a batch COBOL program. In addition, the RETURN command specifies what trans-id CICS should invoke the next time the operator presses an AID key. Suppose the operator presses the CLEAR key to end the terminal session. Since the CLEAR key is an AID key, CICS restarts the program automatically. When program sees that the operator pressed the CLEAR key rather than the ENTER key, it sends the message to the terminal. Then it issues a RETURN command without a trans-id. That ends the pseudo-conversational cycle.

Transaction Driven

A transaction is a logical unit of work that a terminal user can invoke. CICS is called to be transaction driven, because every transaction or program can be invoked with the help of a transaction-id. In short, transaction-id and the program to which transaction-id is related are predefined in the PCT table. Each trans-id is a unique four character code. Program-name is the program-id as defined in COBOL program. When the user types the trans-id on the terminal and presses the enter (AID) key, CICS locates the trans-id in PCT and refers to PPT. Loads the program into storage and a TASK is initiated i.e. CICS passes control to the application program. Every task is assigned a unique task-no by CICS.

The initiated task continues till it encounters any i/o and passes control to CICS. Task can also be initiated by CICS, by providing the trans-id while returning control to CICS. When the user presses an AID key, CICS initiates the task again related to the trans-id returned. This time the task gets a different task no. So one application is divided into many tasks during the process of execution.

CICS Program Development

Application programming in CICS can be done with many host languages that IBM supports, like COBOL, PL/1, and Assembler etc. The coding can be done either with Macro level or using CICS Command level library.

Macro Level Programming

Macro Level Programming involves calling macros in the host language using the program areas for accessing CICS resources. This style of programming is cryptic and complex as the programmer is required to remember all the macros and their parameters.

Command Level Programming

Command Level Programming is a high-level language which provides set of CICS commands for accessing the resources. Programming in this style is comparatively easier. Command level Preprocessor translates the CICS commands into equivalent macro calls.

CICS Command Format & Application Development with COBOL

- CICS commands are embedded into the Host language, e.g. in the Procedure Division of COBOL program.
- CICS commands in a COBOL program are delimited by
  EXEC CICS.....END EXEC
- EXEC CICS is coded in margin B of COBOL program.
- The command level translator replaces these statements by COBOL "MOVE" statements followed by COBOL CALL statement.

- The translated module is complied & linked to produce an executable load module
- The Translator also includes copy books into the program.
- CICS Command format:

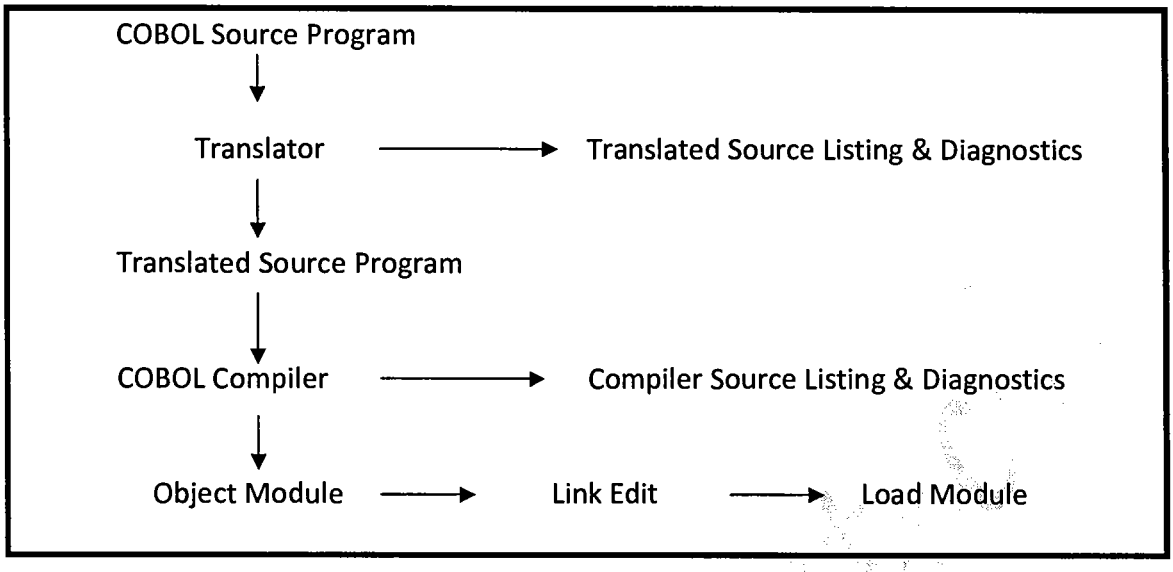
```
EXEC CICS Function
                [Option (argument)
                 Option (argument)
                 ---------
                 ---------
                ]
END-EXEC.
```

  - "Function" describes the operation required e.g. SEND/RECEIVE etc.
  - "Option" describes any of the many optional facilities available with each function.
  - Options can be coded in any order but preferably one option per line. Options may be followed by an argument in parentheses.
  - "Argument" is a value, can be a data-name or literal.

Command Language Translator

- Separate translators are available for Assembly, COBOL and PL/1 languages with embedded CICS.
- The translator is executed in a separate job step.
- The job step sequence is :
  - Translate --- Compile (or assemble) --- Link edit
- Each translator is host language oriented and accepts the source program as input from SYSIN.
- The translator writes the source listing and error messages to SYSPRINT.
- The translated source program is accepted as input by the COBOL compiler and link edited to generate a load module.
- The translator modifies the linkage section by inserting the EIB structure as the first parameter and inserts declaration of the temporary variables that it requires into working storage section.
- For COBOL application program, each command is replaced by one or more COBOL move statements followed by a COBOL CALL statement.
- MOVE statement assign constants to COBOL data variables.
- The constants are coded as arguments to options in the commands.
- Declarations for variables in working storage section is coded as copy file name DFHIVAR.

Command Level Translator



COBOL/CICS Program Structure

IDENTIFICATION DIVISION.
PROGRAM-ID.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
Variable Declaration
File Layout
Symbolic Map
Copy Books
DFHEIVAR
LINKAGE SECTION.
DFHCOMMAREA
EIBBLOCK
BLL CELSS
PROCEDURE DIVISION.

The following COBOL statements are prohibited in a CICS application program:

- ACCEPT, CURRENT-DATE, DATE, DAY, DISPLAY, EXHIBIT, STOP RUN, TRACE
- Any I/O statements (OPEN, CLOSE, READ, WRITE, REWRITE, DELETE, START)
- REPORT WRITER feature
- SORT feature

The equivalent statements (except Report Writer and Sort) are prepared in the form of CICS commands.

The CICS application program must end with the CICS RETURN command and/or GOBACK statement.

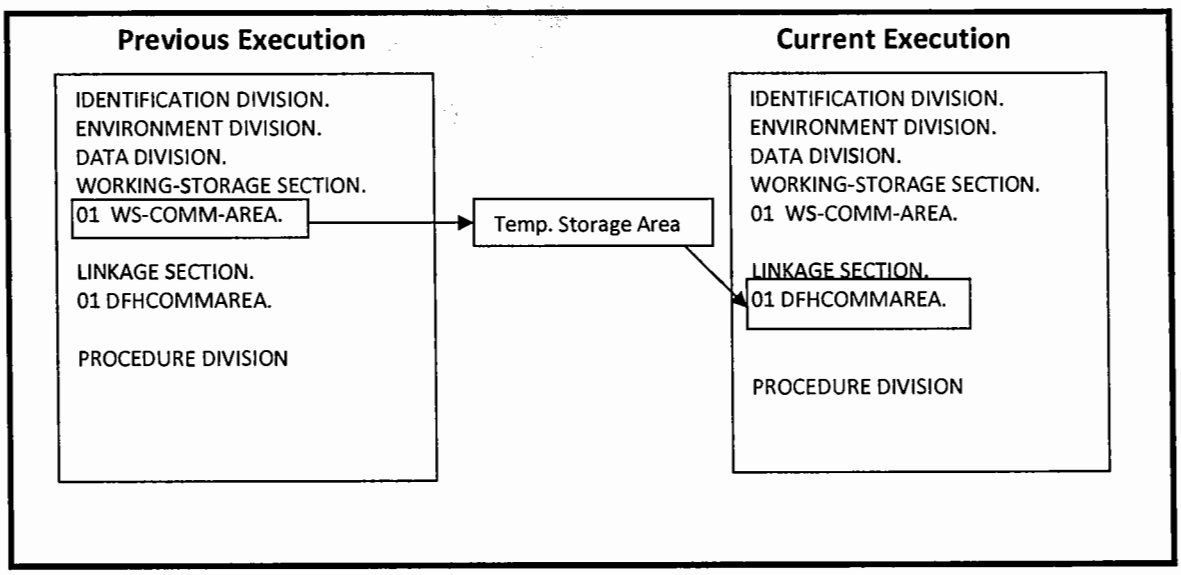
The CALL statement is allowed if the called program does not issue any CICS commands or inhibited COBOL statements mentioned above and if it is written as a reentrant program. The CALL statement in this case can be issued as the following example:

CALL subprog USING xxx yyy zzz

Passing the data using COMMAREA

- Passing of data from one program to another is achieved by COMMAREA defined in working-storage section.
- The receiving program will receive the data in its linkage section, under DFHCOMMAREA
- May be used in functions like RETURN/XCTL/LINK
- The length of COMMAREA must be specified in the LENGTH parameter of the LINK or XCTL command in the calling program. The LENGTH parameter must be defined as a half-word binary field (S9(4) COMP). The maximum length which can be specified is 65,536 bytes.
- Called program will receive data in LINKAGE SECTION
- Called program can find length of data passed using the EIB field EIBCALEN
- If no data is passed then EIBCALEN = 0
- If data is passed then EIBCALEN = Length of COMMAREA

Using COMMAREA in Pseudo-Conversational Cycle



DFHCOMMAREA Usage

CICS COPY BOOKS

Exec Interface Block (EIB) In Linkage Section

- Keeps record of system related information. For example task number, terminal id, date, time etc.
- One EIB is created per task.
- EIB for a task contains information about the task, which is initiated.
- Copy of DFHEIBLK as EIB is automatically included in linkage section of application program during translation.
- Program can only access data using EIB field names.
- User should not update data in EIB fields.

- Some of the EIB Fields are:

  - **EIBAID**: Contains the Attention Identifier (AID) associated with the last input operation.
  - **EIBCALEN**: Contains the length of the communication area passed to the application program.
  - **EIBCPOSN**: Contains the cursor address (position) associated with BMS
  - **EIBDATE**: Contains the date the task is started.
    Format: 00YYDDD PIC S9(7) COMP-3
  - **EIBTIME**: Contains the time at which the task is started.
    Format: HHMMSS PIC S9(7) COMP-3
  - **EIBDS**: Contains the symbolic identifier of the dataset referred to in a file control request
  - **EIBRCODE**: Contains the CICS response code returned after the function requested by the task has been completed
  - **EIBREQID**: Contains the request identifier assigned to an interval control command by CICS
  - **EIBRESP**: Contains a binary number corresponding to the condition that has been raised. For example 11-TERIDERR/13-NOTFND/15-DUPKEY
  - **EIBTRMID**: Contains the symbolic terminal identifier associated with the task.
  - **EIBTRNID**: Contains the symbolic transaction identifier of the task.
  - **EIBTASKN**: Contains the task number assigned to the task by CICS

Standard Attribute Byte List for DFHBMSCA

Constants	Meaning
DFHBMPEM	Printer end-of-message
DFHBMPNL	Printer new line
DFHBMASK	Auto-Skip
DFHBMUNP	Unprotected
DFHBMUNN	Unprotected & num
DFHBMPRO	Protected
DFHBMPRY	Bright
DFHBMDAR	Dark
DFHBMFSE	MDT set
DFHBMPRF	Protected and MDT set
DFHBMASF	Auto-Skip & MDT set
DFHBMASB	Auto-Skip & bright

Installation Defined Attribute Constants

```
01 Field-Attribute-Definition.
      05 FAC-UNPROT                  PIC X  VALUE ' '.
      05 FAC-UNPROT-MDT              PIC X  VALUE 'A'.
      05 FAC-UNPROT-BRT              PIC X  VALUE 'H'.
      05 FAC-UNPROT-BRT-MDT          PIC X  VALUE 'I'.
      05 FAC-UNPROT-DARK             PIC X  VALUE '<'.
      05 FAC-UNPROT-DARK-MDT         PIC X  VALUE '('.
      05 FAC-UNPROT-NUM              PIC X  VALUE '&'.
      05 FAC-UNPROT-NUM-MDT          PIC X  VALUE 'J'.
      05 FAC-UNPROT-NUM-BRT          PIC X  VALUE 'Q'.
      05 FAC-UNPROT-NUM-BRT-MDT      PIC X  VALUE 'R'.
      05 FAC-UNPROT-NUM-DARK         PIC X  VALUE '*'.
      05 FAC-UNPROT-NUM-DARK-MDT     PIC X  VALUE ')'.
      05 FAC-PROT                    PIC X  VALUE '-'.
      05 FAC-PROT-MDT                PIC X  VALUE '/'.
      05 FAC-PROT-BRT                PIC X  VALUE 'Y'.
      05 FAC-PROT-BRT-MDT            PIC X  VALUE 'Z'.
      05 FAC-PROT-DARK               PIC X  VALUE '%'.
      05 FAC-PROT-DARK-MDT           PIC X  VALUE '_'.
      05 FAC-PROT-SKIP               PIC X  VALUE '0'.
      05 FAC-PROT-SKIP-MDT           PIC X  VALUE '5'.
      05 FAC-PROT-SKIP-BRT           PIC X  VALUE '8'.
      05 FAC-PROT-SKIP-BRT-MDT       PIC X  VALUE '9'.
      05 FAC-PROT-SKIP-DARK          PIC X  VALUE '@'.
      05 FAC-PROT-SKIP-DARK-MDT      PIC X  VALUE  QUOTE.
```

Installation Defined Message File

```
01 WS-MESSAGES.
   05 WS-FATAL-ERROR-MSG      PIC X(30)  VALUE 'YOU CAN ONLY START FROM MNMU'.
   05 WS-NORMAL-EXIT-MSG     PIC X(30)  VALUE 'SESSION ENDS HAVE A NICE DAY'.
   05 WS-INVALID-KEY-MSG      PIC X(30)  VALUE 'INVALID AID KEY PRESSED'.
   05 WS-NOT-UNIQUE-MSG       PIC X(30)  VALUE 'ISSUE NOTE NO EXISTING'.
   05 WS-ITEM-NOT-FND-MSG  PIC X(30)  VALUE 'ITEM NOT FOUND'.
   05 WS-INVALID-QTY-MSG     PIC X(30)  VALUE 'INVALID QUANTITY ENTERED'.
   05 WS-DUP-REC-MSG          PIC X(30)  VALUE 'DUPLICATE RECORD'.
   05 WS-GENERAL-ERR-MSG    PIC X(30)  VALUE 'ERROR IN HIGHLIGHTED FIELDS'.
   05 WS-SUCCESS-MSG          PIC X(30)  VALUE 'RECORD SUCCESSFULLY ADDED'.
   05 WS-TRAN-CANCEL-MSG    PIC X(30)  VALUE 'TRANSACTION CANCELLED'.
   05 WS-INVALID-DATE-MSG    PIC X(30)  VALUE 'INVALID DATE'.
   05 WS-RECORD-ERR-MSG      PIC X(30)  VALUE 'ERROR : RECORD NOT FOUND'.
   05 WS-NOT-STOCK-MSG       PIC X(30)  VALUE 'ITEM NOT IN STOCK'.
```

Attention Identifier (AID)

Attention Identifier (AID) indicates which method the terminal operator has used to initiate the transfer of information from the terminal device to CICS. AID keys are: PF keys, PA keys, ENTER key, and CLEAR key. The EIBAID field in EIB contains the AID code of the most recently used AID. Therefore, the EIBAID field can be tested after each Terminal Control (or BMS) input operation. CICS provides the standard AID list in a form of copy library member (DFHAID), so that a program can use this list by specifying in the program:

COPY DFHAID, The DFHAID member contains such AID codes as:

DFHENTER	:	ENTER key
DFHCLEAR	:	CLEAR key
DFHPA1-3	:	PA1 to PA3 keys
DFHPF1-24	:	PF1 to PF24 keys

The following is an example of using AID information in a program:

```
IF EIBAID = DFHPF3
        PERFORM 2100-END-ROUTINE
ELSE IF EIBAID = DFHPA1
        PERFORM  2200-CANCEL-ROUTINE
ELSE IF EIBAID = DFHENTER
        PERFORM 2300-NORMAL-ROUTINE
ELSE
        PERFORM 2400-WRONG-ROUTINE.
```

APPLICATION PROGRAM HOUSEKEEPING

Exceptional Conditions

An abnormal situation during execution of a CICS command is called an "exceptional condition". Checking exceptional conditions is similar to checking return codes after executing COBOL input/output statements in non-CICS programs. It is strongly recommended to check exceptional conditions for every CICS commands. Each CICS command has its own set of possible exceptional conditions

Command Code and Response Code

The CICS command executed last is shown in a system field EIBFN, while the response code for the last CICS command is shown in another system field EIBRCODE. The exceptional condition is nothing but the interpretation of the response code in the EIBRCODE field based on the command code in the EIBFN field.

ABEND Codes

If an exceptional condition occurs during execution of a CICS application program, and if program does not check the exceptional condition, CICS will by default terminates the task. This kind of termination is known as abnormal termination and CICS will issue the abnormal termination (ABEND) code, which is associated with the exceptional condition on subject.

Exceptional conditions can be handled using any of the following:

- HANDLE CONDITION Command
- IGNORE CONDITION Command
- NOHANDLE Option

Handle Condition Command

Function

The HANDLE CONDITION command is used to transfer control to the procedure label specified if the exceptional condition specified occurs. Once a HANDLE CONDITION command request has been made, it remains active until the end of the program or another HANDLE CONDITION request overrides it. In order to avoid the confusion over which HANDLE CONDITION request is active, it is strongly recommended that a HANDLE CONDITION request always be paired with a CICS command

The HANDLE CONDITION request is effective only within the program, which issues the HANDLE CONDITION command. That is, if the control is passed to another program, the HANDLE CONDITION request in the calling program will no longer be honored in the called program.

Format

The format of the HANDLE CONDITION command is as follows:

```
EXEC CICS HANDLE CONDITION
                Condition(Label)
                [Condition(Label)]
                [ERROR(Label)]
END-EXEC.
```

The "condition" represents an exceptional condition. If a label is specified, control will be passed to the labeled paragraph of the program when the condition specified occurs. If no label is specified, it has the effect of canceling the previously set HANDLE CONDITION request and the default action will be taken.

The general error condition (ERROR) can be specified within the same list to specify that all other conditions cause control to be passed to the label specified. Although more than one HANDLE CONDITION command can be issued in the program, not more than 16 conditions can be specified in a single HANDLE CONDITION command.

Example of HANDLE CONDITION Command

```
2110-RECEIVE-CHOICE SECTION.
    EXEC CICS HANDLE CONDITION
                        MAPFAIL(2110-MAP-FAIL)
    END-EXEC.
    EXEC CICS RECEIVE MAPSET('MENUMAP')
                        MAP('MENUMAP')
                       INTO(MENUMAPI)
    END-EXEC.
    GO TO 2110-EXIT.
 2110-MAP-FAIL.
    MOVE 'MAP FAIL SESSION ENDED' TO WS-END-SESSION-MESSAGE.
    MOVE 'Y' TO WS-END-SESSION.
 2110-EXIT.
    EXIT.
```

Ignore Condition Command

Function

The IGNORE CONDITION command causes no action to be taken if the condition specified occurs in the program. That is, control will be returned to the next instruction following the command, which encountered the exceptional condition. The request by the IGNORE CONDITION command is valid until the subsequent HANDLE CONDITION command for the same condition.

Format Of IGNORE CONDITION Command

```
EXEC CICS IGNORE CONDITION
                Condition
                 [Condition]
END-EXEC.
```

The "condition" indicates an exceptional condition. No more than 12 conditions are allowed in the same command.

Example Of IGNORE CONDITION Command

```
EXEC CICS IGNORE CONDITION
                 LENGERR
END-EXEC.
EXEC CICS RECEIVE MAP('MAP1')
                MAPSET('MAP1')
                 INTO(MAP1I)
                LENGTH(80)
END-EXEC.
```

NoHandle Option

The NOHANDLE Option can be specified in any CICS command, and it will cause no action to be taken for any exceptional condition occurring during execution of this command. This option is very useful to prevent a loop on the exceptional condition. However, this option should be used for the special purpose only, because the excessive use of this option defeats the purpose of the exceptional conditions and the HANDLE CONDITION command.

Example of NOHANDLE Option

```
EXEC CICS SEND
                MAP(----)
                MAPSET(------)
                FROM(------)
               LENGTH(---)
               NOHANDLE
END-EXEC.
```

RESP Option

The RESP option can be specified in many CICS command. Its function is similar to the return code in the batch program. If RESP option is specified in a command, CICS places a response code at a completion of the command. The application program can check this code, and then proceed to the next processing. If the RESP option is specified in a command, the NOHANDLE option is applied to this command. Therefore, the HANDLE CONDITION requests will have no effect in this case.

The following are the procedures to utilize the RESP option in a CICS command:

1. Define a full word binary field (S9(8) COMP) in the Working Storage Section as the response field.
2. Place the RESP option with the response field in a command (any CICS command).
3. After command execution, check the response code in the response field with DFHRESP(xxxx), where xxxx is:
    a. NORMAL for normal completion.
    b. Any exceptional condition.

Example Of RESP, NOHANDLE

```
EXEC CICS RECEIVE
          MAP(----)
          MAPSET(------)
          INTO(------)
          LENGTH(---)
          NOHANDLE
           RESP(WS-RESP-FIELD)
END-EXEC.
IF WS-RESP-FIELD = DFHRESP(NORMAL)
                 ----------
ELSE IF  WS-RESP-FIELD = DFHRESP(MAPFAIL)
                ----------
```

Note:

- Use of RESP implies NOHANDLE
- NOHANDLE overrides HANDLE CONDITION and HANDLE AID. Hence PF key responses might be ignored if used with RECEIVE command.
- NOHANDLE also suspends execution till the specified resources become available (e.g.: TSQ/TDQ etc.).

Handle Aid Command

Function

The HANDLE AID command is used to specify the label (i.e., paragraph name) to which control is to be passed when the specified AID is received. After the completion of any terminal input commands such as RECEIVE command, control will be passed to the specified paragraph in the program. This is one way of substituting the EIBAID checking approach. AID keys are PA keys, PF keys, ENTER and CLEAR key. CLEAR and any of the PA keys do not transmit data.

Format Of HANDLE AID Command

```
EXEC CICS  HANDLE AID
                Option(Label)
END-EXEC.
```

- Where "Label" is the paragraph name of the program to which control is to be passed.
- Use of AID key is detected by RECEIVE MAP command.
- HANDLE AID has higher precedence than HANDLE CONDITION.
- Up to 12 options can to can be coded in one HANDLE AID

Commonly used Options

- Any key name (PA1 to PA3, PF1 to PF24, ENTER, CLEAR)
- ANYKEY (Any of the above except ENTER key)

Example Of HANDLE AID Command

```
EXEC CICS HANDLE AID
                PF3(2100-END-ROUTINE)
                PA1(2100-CANCEL-ROUTINE)
                ENTER(2100-NORMAL-ROUTINE)
                ANYKEY(2100-WRONG-KEY-ROUTINE)
END-EXEC.
EXEC CICS RECEIVE MAP('MAP1')
                MAPSET('MAP1')
                INTO(MAP1I)
                LENGTH(WS-MAP-LENGTH)
END-EXEC.
```

Example (Showing Disadvantage of using HANDLE AID Command)

```
2100-RECEIVE-MAP SECTION.
        EXEC CICS HANDLE CONDITION
                MAPFAIL(2100-MAPFAIL-ROUTINE)
        END-EXEC.
        EXEC CICS HANDLE AID
                PF3(2100-END-ROUTINE)
                PA1(2100-CANCEL-ROUTINE)
                ENTER(2100-NORMAL-ROUTINE)
                ANYKEY(2100-WRONG-KEY-ROUTINE)
        END-EXEC.
        EXEC CICS RECEIVE MAP('MAP1')
                MAPSET('MAP1')
                INTO(MAP1I)
                LENGTH(WS-MAP-LENGTH)
        END-EXEC.
2100-MAPFAIL-ROUTINE.
        ---------
2100-NORMAL-ROUTINE.
        ---------
21000-EXIT.
        EXIT.
```

EIBAID

- The 3270 terminal transmits the AID character to the EIBAID field of EIB, at the time of task initiation.
- EIBAID can be used to find which AID Key has been pressed by user
- CICS copy book DFHAID fields are used to identify the key pressed

  IF EIBAID = DFHAID(ENTER)

  Or

  IF EIBAID = DFHENTER
- EIBAID value can be checked even without receiving map.

BMS Programming Considerations

Dynamic Attribute Character Assignment:

CICS provides the standard attribute character list (DFHBMSCA) in the form of a COPYLIB member. This list covers most of the required attribute characters for an application program. You can use this list by copying into the Working Storage Section of the application program through the COPY statement as follows:

COPY DFHBMSCA.

Following are some useful standard attribute characters in DFHBMSCA:

DFHBMASK	:	Auto-skip
DFHBMFSE	:	Unprotected, MDT on
DFHUNNUM	:	Unprotected, MDT on, numeric
DFHUNIMD	:	Unprotected, high intensity, MDT on
DFHUNINT	:	Unprotected, high intensity, MDT on, numeric

You can assign a default attribute character in a BMS map. But, in the cases like an edit error in a field, you might wish to highlight the field indicating an error in a field. This can be accomplished by dynamically assigning a new attribute character for the field on subject.

For the dynamic attribute character assignment, you place the predefined attribute character to the fieldname+A of the field to which you wish to dynamically assign the attribute character.

The attribute character to be dynamically assigned must be chosen very carefully based on the requirement and the characteristics of the field (e.g., alphanumeric or numeric). When the map is sent through SEND MAP command, the new attribute will be in effect on the field on subject, overriding the original attribute defined at the map definition time.

Example Of Dynamically Attribute Value Assignment

```
WORKING-STORAGE SECTION.
       -------
COPY 'MAPSETA'.
       -------
COPY 'DFHBMSCA'.
       -------
PROCEDURE DIVISION.
       -------
       MOVE DFHBMBRY TO CUSTNO-A.
       MOVE DFHDMPRO TO CUSTNAME-A.
       MOVE DFHDMPRO TO AMOUNT-A.
       EXEC CICS      SEND
                      MAP('MAPNAME')
                       MAPSET('MAPSETA')
       END-EXEC.
```

CURSOR POSITIONING TECHNIQUES

Placing a cursor in a particular position of a screen is sometimes very useful in order to make the system user-friendlier. There are three approaches to cursor positioning.

Static Cursor Positioning:

In this approach, you define a cursor position in a map by placing 'IC' in the ATTRB parameter of the DFHMDF macro for a particular field. When the map is sent, the cursor will appear in this field. If there were more than one field (DFHMDF macro) with IC specified in one map (DFHMDI macro), the last IC would be honored.

The following is an example of the static cursor positioning:

```
DFHMDF      POS=(3,16),
            ATTRB=(UNPROT, FSET, IC)
            LENGTH=8
```

Dynamic/Symbolic Cursor Positioning:

In this approach, you dynamically position a cursor through an application program using a symbolic name of the symbolic map by placing –1 into the field length field (i.e., fieldname+L) of the field where you wish to place the cursor. The SEND MAP command to be issued must have the CURSOR option (without value). Also, the mapset must be coded with MODE=INOUT in the DFHMSD macro.

Then, at the completion of the SEND MAP command, the map will be displayed with the cursor at the position dynamically specified in the application program, irrespective of the static cursor position defined in the map definition time.

The following is an example of dynamic/symbolic cursor positioning:

```
MOVE –1 TO CHOICEL.
EXEC CICS SEND MAP('MAP1')
               MAPSET('MAP1')
               CURSOR
               ERASE
END-EXEC.
```

Dynamic/Relative Positioning:

In this approach, you dynamically position a cursor through an application program using the CURSOR(data-value) option in the SEND MAP command with the value of the relative position (starting from zero) of the terminal. At the completion of the SEND MAP command, the map will be displayed with the cursor at the specified position, overriding the static cursor position defined at the map definition time. Data-value is calculated as: (row – 1) * 80 + (column - 1).

The following is an example of the dynamic/Relative cursor positioning:

```
EXEC CICS SEND
               MAP(------)
               MAPSET(-------)
               CURSOR(100)
               ERASE
END-EXEC.
```

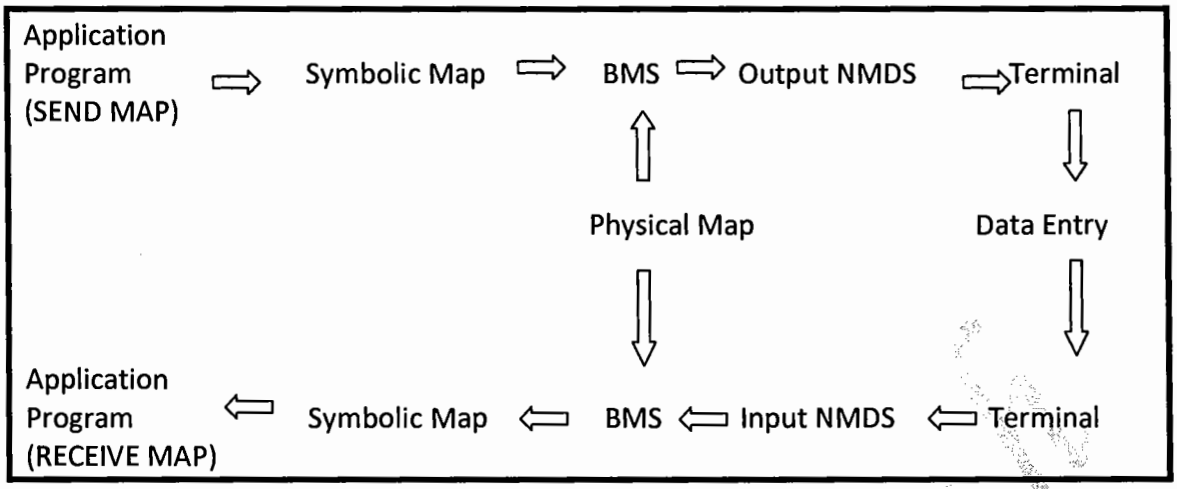
Although this approach makes the application program able to position the cursor dynamically, since the relative position of the terminal is device dependent, this approach is device dependent and format dependent. Whenever the terminal type is changed or the screen layout is changed, the application program will have to be modified to reflect the change. This lessens the program maintainability.

TERMINAL CONTROL OPERATIONS

Introduction: This chapter will discuss how BMS maps are used in the application program for the actual terminal input/output operations, which are performed by a set of CICS commands for BMS.

Functions: The CICS commands for BMS perform the following three basic functions:

1. *Map Sending Function:* Using the data in the symbolic map, BMS prepares the output Native Mode Data Stream (NMDS) the corresponding physical map, and sends to the terminal as illustrated in below figure.
2. *Map Receiving Function:* Using the input NMDS from the terminal, BMS prepares data in the symbolic map through the corresponding physical map as illustrated in below figure.
3. *Text Handling Function:*
   BMS prepares text without using a map and sends to the terminal.



Concept of BMS Operations

The following commands are available for the basic BMS input/output operations:

- RECEIVE MAP : Formatted data transfer. To receive a map
- SEND MAP : Formatted data transfer. To send a map
- RECEIVE : Unformatted data transfer. To receive a text
- SEND TEXT : Unformatted data transfer. To send a text

RECEIVE MAP Command (Formatted Data Transfer)

Function

The RECEIVE MAP command is used to receive a map from a terminal. At the completion of the command, the symbolic map of the specified map will contain the valid data from the terminal in the following 3 fields per each field defined by the DFHMDF macro:

Fieldname+L: The length field, which contains actual number of characters typed in the screen

Fieldname+F: The flag byte field, which is normally X'00'. It will be X'80' if the screen field has been modified but cleared.

Fieldname+I: The actual input data field.

Format Of RECEIVE MAP Command

```
EXEC CICS RECEIVE MAP (map name)
                      MAPSET(map name)
                       INTO(data-area)
END-EXEC.
```

MAP defines the name of the map, which was defined in the corresponding DFHMDI macro. MAPSET defines the name of the mapset which was defined in the corresponding DFHMSD macro. If nothing else is specified, the INTO option is assumed, and BMS automatically finds the symbolic map area to place the data from the terminal. This command when executed will place all modified data items in the symbolic work area.

Common Exceptional Condition

MAPFAIL

This occurs

- If the data to be mapped has the length of zero.
- If user presses the ENTER (or any AID key) without typing data if no FSET is specified in DFHMDF macros.
- If map is received by pressing CLEAR or any of the PA keys.

Data Validity Checking

After the completion of the RECEIVE MAP command, the data from the terminal are placed in the fields of the symbolic map. However you cannot assume that all data in the symbolic map are valid, since BMS has nothing to do with the quality of data contents in the fields which were entered by the terminal user. Therefore, you must validate the data in each field before you use it.

One of the online system's advantages is its capability of real-time (i.e., instant) data validation and feedback to the user. Therefore, you should perform data validity checking as thoroughly as possible right after the RECEIVE MAP command.

Procedures

The following is a set of procedures for the effective data validity checking after the completion of the RECEIVE MAP command:

1. Perform the data validity checking in detail as soon as the program receives data from the terminal.
2. Check the fieldname+L:
    a. If it is zero, the user has entered no data. Therefore, take the default action, or consider an error if it is the required field.
    b. If it is a positive value, some data of that length has been entered. Therefore validate the field data in the fieldname+I.
3. Alternatively, check directly the fieldname+I:
    a. If it is LOW-VALUES or SPACE, no data has been entered or space key has been pressed. Therefore, take the default action, or consider an error if it is the required field.
    b. If it is not LOW-VALUE or space, some data has been entered. Therefore, validate the field data in the fieldname+I.
    c. Note that CICS considers spaces as data. Therefore, space may be valid data, depending on the application specifications.
4. If an error is detected, prepare an error message. For this, each map should have the error message fields.
5. Do not stop validity checking at the first error detection. Go to the next field for another validation.
6. Repeat this until all data fields have been verified.
7. At completion of the data validation, if there are errors, send error message(s) to the terminal for the user to correct them and reenter.

SEND MAP Command (Formatted Data Transfer)

Function

The SEND MAP command is used to send a map to a terminal. Before issuing this command, the application program must prepare the data in the symbolic map of the map to be sent, which has the following three fields per each field defined by the DFHMDF macro:

fieldname+L: The length field, to which the application program does not have to prepare the data except for the dynamic cursor positioning.

fieldname+A: The attribute character filed, to which the application program does not have to prepare the data except for the dynamic attribute character assignement.

fieldname+O: The actual output data field, to which the application program must place the data.

Format Of SEND Command

```
EXEC CICS SEND
          MAP(MAP-NAME)
          MAPSET(MAPSET-NAME)
          [FROM(DATA-AREA)]
          [CURSOR/CURSOR(DATA-VALUE)]
          [ERASE/ERASEAUP]
          [FREEKB]
          [ALARM]
          [FRSET]
          [DATAONLY/MAPONLY]
END-EXEC.
```

MAP

Defines the name of the map defined in the corresponding DFHMDI macro.

MAPSET

Defines the name of the mapset defined in the corresponding DFHMSD macro.

FROM

Even if not specified, this option is assumed, and BMS automatically finds the symbolic map area to take the data to the terminal.

CURSOR

Makes the application program able to position the cursor dynamically on any part of the screen.

ERASE

If this is specified, the current screen will be erased before the map specified appears on the screen. If this is not specified, the map specified will be overwritten onto the current screen. Therefore, double image screen might

appear which may be useful for sending error messages into the current screen, because the current screen will be kept as it is.

ERASEAUP

To erase all unprotected fields. The protected fields or attribute fields remain as they are in the current screen.

FREEKB

To free the Keyboard.

ALARM

To make an Alarm Sound.

FRSET

To reset MDT to zero (i.e., not modified) for all unprotected fields of the screen.

DATAONLY

Only application program data in the symbolic map is to be sent to the terminal.

MAPONLY

Only the default data from the physical map is to be sent to the terminal.

[If the above two options are not given, data from both the maps is sent to the terminal.]

Example Of SEND Command

```
WORKING-STORAGE SECTION
 COPY MAPA.
PROCEDURE DIVISION
EXEC CICS HANDLE CONDITION
                INVMPSZ (BIG-MAP)
 END EXEC.
        EXEC CICS SEND MAP('MAPA')
                MAPSET('MAPA')
                FROM(MAPAI)
 END-EXEC.
BIG-MAP
 *ERROR PROCESSING
```

RECEIVE Command (Unformatted Data Transfer)

Function

The command is used to receive an unformatted message from the terminal. In this data transfer maps are not involved.

Format Of RECEIVE Command

```
EXEC CICS RECEIVE
                INTO(DATA-AREA)
                LENGTH(DATA-VALUE)
END-EXEC.
```

Example Of RECEIVE Command

```
IDENTIFICATION DIVISION.
  ----
WORKING-STORAGE SECTION.
01  MSG-LENGTH     PIC S9(4) COMP.
01  INPUT-MSG.
    02 TRANS-ID    PIC X(4).
    02 FILLER      PIC X(1).
    02 MESSAGE     PIC X(5).
  ----
PROCEDURE DIVISION.
  MOVE 10 TO MSG-LENGTH
  EXEC CICS HANDLE CONDITION
            LENGERR(LENGTH-ERROR-RT)
  END-EXEC
  -----
  EXEC CICS RECEIVE INTO(INPUT-MSG)
            LENGTH(MSG-LENGTH)
  END-EXEC
    ----
 LENGTH-ERROR-RT.
  -----
```

SEND TEXT Command (Unformatted Data Transfer)

Function

The command is used to send unformatted message (i.e., data without any maps). This message will always get displayed at row 1, column 1 position on the screen.

Format Of SEND TEXT Command

```
EXEC CICS SEND TEXT
            FROM(data-area)
            LENGTH(data-value)
            [ERASE]
            [FREEKB]
END-EXEC.
```

Example Of SEND TEXT Command

```
IDENTIFICATION DIVISION.
 ----
WORKING-STORAGE SECTION.
OUTPUT-MESSAGE.
 -----
01 WS-MESSAGE     PIC X(30) Value Spaces
PROEDURE DIVISION.
-------
MOVE 'END OF SESSION' TO WS-MESSAGE.
   EXEC CICS SEND TEXT
             FROM(WS-MESSAGE)
             LENGTH(30)
             ERASE
             FREEKB
   END-EXEC.
 ----------
```

INTERVAL CONTROL OPERATIONS

Introduction

The CICS Interval Control Program provides application program with time-controlled functions, such as the time-oriented task synchronization, providing current date and time, and automatic initiation of the time-ordered tasks

ASKTIME Command

Function

The ASKTIME command is used to request the current date and time. The EIBDATE and EIBTIME fields have the values at the task initiation time. Up on completion of this command, these fields will be updated to have the current date and time.

Format Of ASKTIME Command

```
EXEC CICS ASKTIME
             [ABSTIME(data-area)]
END-EXEC.
```

ABSTIME option is required if you want to store current date and time for further processing. Without this option ASKTIME will just update EIBDATE and EIBTIME field. The data-area is defined in Working-Storage Section with picture clause of S9(15) COMP. "data-area" after the execution gets the value in milliseconds.

FORMATTIME Command

Function

The FORMATTIME command is used to receive the information of date and time in various formats.

Format Of FORMATTIME Command

```
EXEC CICS FORMATTIME
          ABSTIME(data-area)
          [YYDDD(data-area)]
          [YYMMDD(data-area)]
          [YYDDMM(data-area)]
          [MMDDYY(data-area)]
          [DDMMYY(data-area)]
          [DATESEP(data-value)]
          [DAYOFWEEK(data-area)]
          [DAYOFMONTH(data-area)]
          [MONTHOFYEAR(data-area)]
          [YEAR(data-area)]
          [TIME(data-area) [TIMESEP(data-value)]]
END-EXEC.
```

"data-area" for date and time are defined as X(08). Time separator by default is " : " and date separator by default is " / " (for this specify TIMESEP and DATESEP options without argument). If these two options are omitted, no separator is provided.

PROGRAM CONTROL OPERATIONS

Introduction

The CICS Program Control Program (PCP) governs a flow of control among CICS application programs and CICS itself. The name of a CICS application program must be registered in the Processing Program Table (PPT), otherwise the program will not be recognized by CICS

The following commands are available for the Program control services.

RETURN	: To return to the next higher-level program or CICS.
LINK	: To pass control to another program at the lower level, expecting to be returned.
XCTL	: To pass control to another program at the same level, not expecting to return.
LOAD	: To load a program.
RELEASE	: To release a loaded program.

Application Program Levels

The application programs under CICS run at various logical levels, the first CICS application program to receive control from CICS within a task runs at the highest logical level 1. A LINKed program runs at the next lower logical level from the LINKing program, while an XCTL'ed program runs at the same logical level as the XCTL'ing program. The RETURN command always passes control back to the program at one logical level higher.

RETURN Command

Function: The RETURN command is used to return control to the next higher logical level, or CICS itself.

Format Of RETURN Command

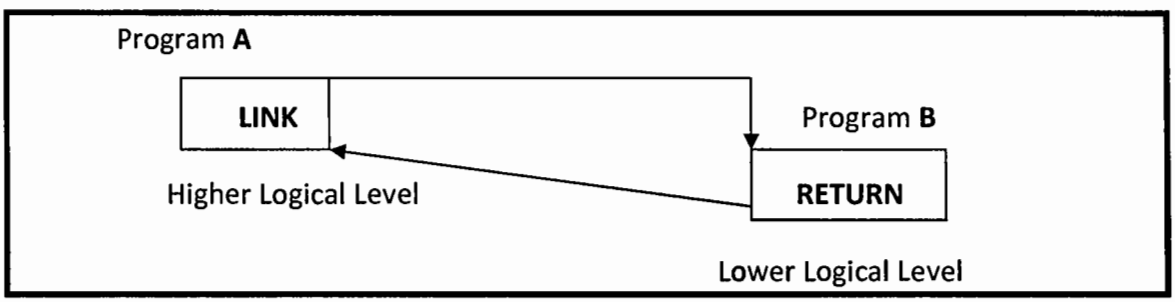
```
EXEC CICS RETURN
                [TRANSID(name)
                [COMMAREA(data-area)
                [LENGTH(data-value)]]]
END-EXEC.
```

If none of TRANSID, COMMAREA, or LENGTH is specified, and if COMMAREA had been passed by a calling program, the RETURN command makes the data in COMMAREA available to the calling program. That is, the called program does not have to use the COMMAREA option in the RETURN command. If the TRANSID option is used, the specified transaction identifier will be the transaction identifier for the next program to be associated with the terminal. . This is allowed only in the program at the highest logical level. If the TRANSID option is specified, the COMMAREA and LENGTH option can be used to pass data to the next task

LINK and XCTL Command

Function of LINK Command

The LINK command is used to pass control from an application program at one logical level to another application at the next lower logical level. The calling program expects control to be returned to it. The control is returned at the next sentence when called program completes execution.



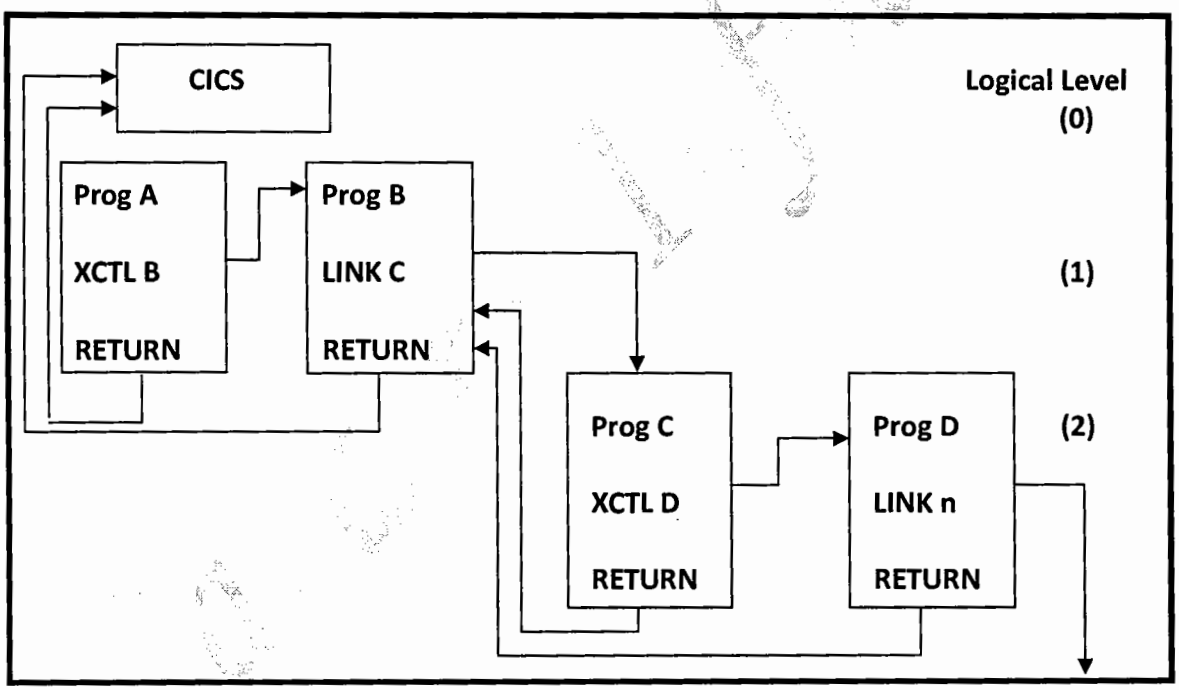
Format Of LINK Command

```
EXEC CICS LINK
                [PROGRAM(name)]
                [COMMAREA(data-area)]
                [LENGTH( data-value)]
END-EXEC.
```

The name of the called program to which control is passed must be specified in PROGRAM.

Function of XCTL command

The XCTL command is used to pass control from one application program to another application program at the same logical level. The calling program does not expect control to be returned. Data can be passed to the program through a special communication area called COMMAREA. Since this command requires less overhead, performance is relatively better in comparison to the LINK command.



Application Program Logical Levels

Format Of XCTL Command

```
EXEC CICS XCTL
                PROGRAM(name)
                [COMMAREA(data-area)]
                [LENGTH(data-value)]
END-EXEC.
```

Example Of LINK

```
*Code in Program 'Menu'
EXEC CICS HANDLE CONDITION
                PGMIDERR(4000-PGM-NOT-FND)
                ERROR(4000-GENERAL-ERROR)
END-EXEC.
EXEC CICS LINK
                PROGRAM('FILE-MAINT')
                COMMAREA(WS-COMM-AREA)
                LENGTH(20)
END-EXEC.
*Code in Program 'FILE-MAINT'
EXEC CICS RETURN ------------------> Takes control back in 'Menu' program
END-EXEC.
```

Example Of XCTL

```
*Code in Program 'Menu'
EXEC CICS HANDLE CONDITION
                PGMIDERR(4000-PGM-NOT-FND)
                ERROR(4000-GENERAL-ERROR)
END-EXEC.
EXEC CICS XCTL
                PROGRAM('FILE-MAINT')
                COMMAREA(WS-COMM-AREA)
                LENGTH(20)
END-EXEC.
*Code in Program 'FILE-MAINT'
EXEC CICS RETURN ------------------> Takes control back to CICS
END-EXEC.
```

Common Exceptional Conditions

The following are the exceptional conditions common to both LINK and XCTL command:

NOTAUTH	:	A resource security check has failed.
PGMIDERR	:	The program specified is not found in PPT.

Data Passing Through COMMAREA

Data can be passed to a called program using the COMMAREA option of the LINK or XCTL command in a calling program. In case of the LINK the called program may alter the data content of COMMAREA and the changes will be available to the calling program after the RETURN command is issued in the called program.

This implies that the called program does not have to specify the COMMAREA option in the RETURN command. If the COMMAREA is used in the calling program, the area must

be defined in the Working Storage Section of the program, whereas, in the called program, the area must be defined as the first area in the Linkage Section, using the reserved name DFHCOMMAREA.
The length of the COMMAREA must be specified in the LENGTH parameter of the LINK or XCTL command in the calling program. The LENGTH parameter must be defined as a half word binary field (S9(04) COMP). The maximum length which can be specified is 65,536 bytes. EIBCALEN determines whether the calling program sent COMMAREA. If EIBCALEN is greater than 0, COMMAREA was sent. This should always be checked.

Difference between XCTL and LINK

XCTL	LINK
Calling program is released from the main memory	Calling program is not released from the main memory
Control is transferred to the same logical level	Control in transferred to the lower logical level
Return can to CICS or to the application program	Return is always to the application program
COMMAREA option can be used to pass data to the invoked program	COMMAREA option is not required in RETURN because the invoked program does pass pointer to the communication area of the calling program.

FILE CONTROL OPERATIONS (RANDOM ACCESS)

The CICS File Control Program (FCP) provides application programs with services to read, update, add, and delete records in a file. In addition, it makes application programs independent of the structure of the database, while it manages exclusive control over the records in order to maintain the data integrity during record updates.

Supported Access Methods

CICS File Control supports the VSAM and BDAM data access methods, of which VSAM is the primary data access method under CICS. There are three types of VSAM file, all of which are supported by CICS. These are as follows:

- Key-Sequenced Dataset (KSDS)
- Entry-Sequenced Dataset (ESDS)
- Relative-Record Dataset (RRDS)

Available Commands

For random access of VSAM dataset following commands are available:

READ	:	To read a record directly
WRITE	:	To newly write a record
REWRITE	:	To update an existing record
DELETE	:	To delete a record
UNLOCK	:	To release exclusive control acquired for update.

Special Services of File Control

Data Independence

Data independence is a concept of a program being independent of the structure of database or the data access methods. The CICS file Control provides data independence to application program, so that the application programmer does not have to be concern with such data-dependent COBOL parameters or JCL as:

- INPUT-OUTPUT SECTION
- SELECT Statement
- FD Statement
- OPEN/CLOSE
- JCL

The system programmer (or application programmer) defines the File Control Table (FCT) to specify the characteristics of files to be used under CICS, while application programmer codes application program using CICS commands. In this way, it is almost transparent to the application programmer which data access method is being used.

Exclusive Control during Updates

If a task is updating a record, the other tasks must be excluded from updating the record; otherwise data content will be updated incorrectly. This control is called exclusive control over the resources during updates. This is important because in the CICS environment, many tasks might be concurrently accessing the same file (possibly the same record). CICS locks the entire control interval where that record under subject is residing.

File Open/Close

When an application program accesses a file, the file must be open under CICS. For this, FCT defines an initial file open/close status. If the file is closed, you must open the file using the Master Terminal transaction (CEMT) before you initiate an application program which uses this file.

READ Command Using Full Key

Function

The READ command with the INTO option using the full key of a record is used to read a specific record specified by the full key. The data content of the record will be moved into the specified field defined in the Working Storage section of the program.

Format Of 'READ with Full Key' Command

```
EXEC CICS READ
          DATASET(name)
          INTO(data-area)
          RIDFLD(data-area)
          LENGTH(data-value)
END-EXEC.
```

This is the basic format for the READ command. As the other options are introduced, the format will vary slightly.

DATASET names the file, which you wish to read. The file name specified must be defined in FCT. INTO names the field in the Working Storage Section to which the record content is to be placed. RIDFLD is used to specify the key field, which identifies the record to be read. LENGTH indicates the maximum length of the record to be read. This is optional, if it is specified, at the completion of the command; CICS will place the actual length of the record into the LENGTH field.

Example Of 'READ with Full Key' Command

```
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION
01 LEN                PIC S9(04) COMP VALUE 25.
01 RECORD-KEY         PIC X(05) VALUE 'A0001'.
01 FILE-AREA.
   02 REC-KEY         PIC X(05).
   02 REC-DESC        PIC X(20).
PROCEDURE DIVISION.
2500-READ SECTION.
  EXEC CICS HANDLE CONDITION
        LENGERR(2500-LENGERR)
        NOTFND(2500-NOTFND)
        ERROR(2500-OTHER)
  END-EXEC.
  EXEC CICS READ
        INTO(FILE-AREA)
        DATASET('MASTER')
        RIDFLD(RECORD-KEY)
        LENGTH(LEN)
  END-EXEC.
2500-LENGERR
* Error processing (the record read is longer than number of bytes specified in length field (len))
        ------
2500-NOTFND
* Error processing (record with specified key is not in the data set)
        -------
2500-OTHER
* Error processing (other error has occurred)
        -------
```

- CICS will read the record 'A0001' and place the data in the INTO field (FILE-AREA).
- The actual record length will be placed in LEN.

Common Exceptional Conditions

DUPKEY	The duplicate record is found in the specified key (In case of Alternate Record Key). The first record in the file, which has that key, will be read. If you want to read other records of the same key, you have to use the Browse operation
NOTFND	No record with the key specified is found.
LENGERR	The specified length is shorter than the actual record length. The record will be truncated at the length specified and moved into the INTO field. The actual length will be placed in the LENGTH field.
NOTOPEN	The file specified is not open.

READ Command with GENERIC Option

Function

The READ command with the GENERIC option is used to read a nonspecific record based on the generic key (i.e. a partial key) specified, instead of the full key. This is useful when you do not know the complete information of the key.

Format

The format of the READ command with the GENERIC option using the INTO option is shown in the example below. For this option, in addition to specifying explicitly GENERIC as the option, the length of the generic key must be specified in the KEYLENGTH field. The format related to the other options or parameters remains the same.

Example Of 'READ with generic Key' Command

```
MOVE 35 TO WK-LEN.
MOVE 'NY' TO REC-A-KEY.
EXEC CICS READ
          DATASET('FILE2')
          INTO(FILE-OAREA)
          RIDFLD(REC-A-KEY)
          KEYLENGTH(2)
          GENERIC
          LENGTH(WK-LEN)
END-EXEC.
```

Execution Results

Suppose that file FILE2 has records in the following order:

BC001
DC001

DC001
NY000 *
NY001
NY002
PH001
PH002

Then, the record (NY000) will be read, because this is the first record of the generic key "NY".

Common Exceptional Conditions

NOTFND	No record with the key specified is found.
LENGERR	The specified length is shorter than the actual record length. The record will be truncated at the length specified and moved into the INTO field. The actual length will be placed in the LENGTH field
INVREQ	The keylength specified is greater than the actual keylength of the record.

READ Command with GTEQ Option

Function

The READ command with the GTEQ option is used to read a nonspecific record whose key is equal to or greater than the full key data specified. This is useful when you know the full key, but you are not sure that the key exists in the file.

Format

The format of the READ command with the GTEQ option using the INTO option is shown in the example below.

Example Of 'READ with GTEQ option' Command

```
MOVE 35 TO WK-LEN.
MOVE 'NY003' TO REC-KEY.   ← = Must be a Full Key
EXEC CICS READ
          DATASET('FILE2')
          INTO(FILE-IOAREA)
          RIDFLD(REC-KEY)
          GTEQ
          LENGTH(WK-LEN)
END-EXEC.
```

READ / UPDATE and REWRITE Commands

Function

A combination of the READ command with the UPDATE option and the REWRITE command is used to update a record. Between these two commands, exclusive control over the record will be maintained for this task, so that no other tasks can access this record for updates.

Format

The format of the READ command with the UPDATE option is shown in the example below. For this option, UPDATE must be explicitly specified as the option. The format of the REWRITE command is also shown in the example below. DATASET must name the same file read by the prior READ/UPDATE command. The FROM data area must be the same record area used by the READ/UPDATE command. LENGTH indicates the length of the new record.

Example Of 'READ/UPDATE and REWRITE' Command

```
MOVE 35 TO WK-LEN.
MOVE 'NY001' TO REC-A-KEY.
EXEC CICS      READ
               DATASET('FILE2')
               INTO(FILE-IOAREA)
               RIDFLD(REC-A-KEY)
               UPDATE
               LENGTH(WK-LEN)
END-EXEC.                          ← = Read for Updating
Set of MOVE Statements.....
EXEC CICS      REWRITE
               DATASET('FILE2')
               FROM(FILE-IOAREA)
               LENGTH(WK-LEN)
END-EXEC.
```

Execution Results

- At the completion of the READ/UPDATE command, record NY001 will be read and reserved for the subsequent update. That is, exclusive control over the record is maintained for this task.
- At the completion of the REWRITE command, the same record NY001 will be rewritten and the record will be released from exclusive control.

Exceptional Conditions

INVREQ The REWRITE command is issued without a prior READ command with the UPDATE option.

UNLOCK Command

Function

The READ command with the UPDATE option normally maintains exclusive control over the record read until:

- The record is updated by the REWRITE command.
- The transaction is normally or abnormally completed.

However, there are occasions when after reading a record through the READ command with the UPDATE option. It is found that the update is no longer required or if the REWRITE commands itself fails the execution. In this case, the application program should release exclusive control from the record, so that other tasks can access the same record. For this purpose, the UNLOCK command is used. That is, the UNLOCK command is used to release the exclusive control from the record.

Format

The format of the UNLOCK command is shown in the example below. DATASET names the file from which exclusive control is to be released.

Example Of 'UNLOCK' Command

```
EXEC CICS UNLOCK
          DATASET('FILE2')
END-EXEC.
```

WRITE Command

Function

The WRITE Command is used to write a record directly into a file based on the key specified.

Format Of 'WRITE' Command

```
EXEC CICS WRITE
                DATASET(name)
                FROM(data-area)
                LENGTH(data-value)
                RIDFLD(data-area)
END-EXEC.
```

Example Of 'WRITE' Command

```
MOVE 35 TO WK-LEN.
MOVE 'NY004' TO REC-A-KEY.
Move symbolic map fields to the record area...
EXEC CICS WRITE
                DATASET('FILE2')
                FROM(FILE-IOAREA)
                RIDFLD(REC-A-KEY)
                LENGTH(WK-LEN)
END-EXEC.
```

Exceptional Conditions

DUPREC The duplicate record is found.

NOSPACE No disk space is available for the record addition.

LENGERR The length specified is greater than the maximum length specified in the VSAM cluster.

WRITE Command with MASSINSERT Option

Function

The WRITE command with the MASSINSERT option is used to add a group of records whose keys are in ascending order into a file. If there are many records to be added as a group, this option will provide high performance in writing these records.

Since the MASSINSERT option causes exclusive control over the file, the file must be released by the UNLOCK command after the completion of the WRITE command.

Procedures

The following are the procedures for the mass-insert operations using the WRITE command with the MASSINSERT option:

1. Establish the first record key.
2. Issue the WRITE command with the MASSINSERT option.
3. Increment the key in the ascending order.
4. Repeat steps 2 and 3 until all records have been written.
5. Issue the UNLOCK command to release the exclusive control over the file.

Format

The format of the WRITE command with the MASSINSERT option is shown in the example below.

Example Of 'WRITE with MASSINSERT' Command

```
PROCEDURE DIVISION
        ........
        MOVE 'TX000' TO REC-A-KEY'.
MASS-INS-LOOP SECTION.
        ADD 1 TO REC-A-KEY-SEQ.
        .......
        (Prepare the record content)
        .......
        EXEC CICS WRITE
                DATASET('FILE2')
                FROM(FILE-IOAREA)
                RIDFLD(REC-A-KEY)
```

```
            LENGTH(WK-LEN)
            MASSINSERT
        END-EXEC.
        IF REC-A-SEQ < 99
                    GO TO MASS-INS-LOOP.
        EXEC CICS UNLOCK
                    DATASET('FILE2')
        END-EXEC.
```

DELETE Command

Function

The DELETE command is used to delete one record or a group of records from a file. There are three approaches to using the DELETE command.

DELETE after READ/UPDATE Approach

In this approach, the DELETE command is used without the record key information after the READ command with the UPDATE option has been completed. The record which was read by the READ/UPDATE command will be deleted from the file.

Format

The format of the DELETE command in this approach is shown in the example below. The only parameter required is DATASET, which must name the same file used by the prior READ/UPDATE command:

Example Of 'DELETE after READ/UPDATE' Command

```
MOVE 35 TO WK-LEN.
MOVE 'NY001' TO REC-A-KEY.
EXEC CICS       READ
                DATASET('FILE2')
                INTO(FILE-IOAREA)
                RIDFLD(REC-A-KEY)
                UPDATE
                LENGTH(WK-LEN)
END-EXEC.
.....
EXEC CICS DELETE
                DATASET('FILE2')
END-EXEC.
```

Exceptional Condition

INVREQ The DELETE command without the RIDFLD option is issued without a prior READ/UPDATE command.

Direct DELETE Approach

In this approach, the DELETE command is issued with the full key information in the RIDFLD field. The record specified will be directly deleted from the file.

Format

The format of the DELETE command in this approach is shown in the example shown below. DATASET must name the file from which a record is to be deleted. In addition, the full key data must be provided in the RIDFLD field for the record to be deleted.

Example Of 'Direct DELETE (without prior READ/UPDATE)' Command

```
MOVE 'NY001' TO REC-KEY.
EXEC CICS DELETE
          DATASET('FILE2')
          RIDFLD(REC-A-KEY)    <= Required
END-EXEC.
```

Exceptional Conditions

DUPKEY There is more than one record with same key. (In the case of Alt. Index)

NOTFND The record specified is not found.

Group Record DELETE Approach

In this approach, the DELETE command is issued with the GENERIC option. A group of records, which satisfy the generic key specified, will be deleted from the file.

Format

The format of the DELETE command in this approach is shown in the example shown below. In addition to the required parameter for the direct delete, GENERIC must be specified as the option. The KEYLENGTH indicates the length of the generic key. The higher part of the key information must be supplied in the RIDFLD field. A half-word binary field (S9(4) COMP) should be provided to NUMREC, which will contain the number of the records deleted by this generic delete.

Example Of 'Group DELETE' Command

```
PROCEDURE DIVISION.
.....
        MOVE 'NY' TO RE-A-KEY.
        EXEC CICS DELETE
                DATASET('FILE2')
                RIDFLD('REC-A-KEY')
```

```
            KEYLENGTH(2)
            GENERIC
            NUMREC(WK-DEL-COUNT)
        END-EXEC.
```

FILE CONTROL OPERATIONS (SEQ. ACCESS)

Under CICS any of VSAM/KSDS, VSAM/ESDS, and VSAM/RRDS can be accessed randomly as well as sequentially.

Available Commands

Sequential access of VSAM file under CICS is called "browsing", for which the following commands are available:

STARTBR	: To establish a position for a browse operation.
READNEXT	: To read a next record (forward)
READPREV	: To read a previous record (backward)
RESETBR	: To reestablish another position for a new browse.
ENDBR	: To complete a browse operation.

STARTBR Command

Function

The STARTBR command is used to establish a browse starting position for a file. This command is for establishing the position only. The actual record will be read by the next read command (READNEXT or READPREV).

Format Of 'STARTBR' Command

```
EXEC CICS STARTBR
            DATASET(name)
            RIDFLD(data-area)
            GTEQ/EQUAL
            KEYLENGTH(data-area)
            GENERIC
            RBA/RRN
END-EXEC
```

Example Of 'STARTBR' Command

```
WORKING-STORAGE SECTION.
77  WK-LEN                         PIC S9(4) COMP.
01  FILE-IOAREA.
        05  REC-A-KEY.
                10  REC-A-KEY-CITY     PIC XX.
                10  REC-A-KEY-SEQ      PIC 999.
        05  REC-A-DETAIL               PIC X(30).
```

```
PROCEDURE DIVISION.
:
:
        MOVE 'NY000' TO RE-A-KEY.
        EXEC CICS STARTBR
                        DATASET('FILE2')
                        RIDFLD(REC-A-KEY)
                        GTEQ                    ←= Default
        END-EXEC.
```

Execution Results

Suppose the file has the records in the following order:

B0001
DC001
DC002
NY001 ←=Then, Browse Starting Position is Set here (NY001).
NY002
NY003
PH001
PH002

Exceptional Conditions

DSIDERR	The file specified is not found in FCT.
NOTFND	The specified record is not found.

READNEXT Command

Function

The READNEXT command is used to read a record of a file sequentially forward. The STARTBR must have been successfully completed prior to issuing the READNEXT command.

Format Of 'READNEXT' Command

```
EXEC CICS  READNEXT
                DATASET(name)
                INTO(data-area)
                LENGTH(data-value)
                RIDFLD(data-area)
                RBA/RRN
END-EXEC.
```

Example Of 'READNEXT' Command

```
WORKING-STORAGE SECTION.
77  WK-LEN                             PIC S9(4) COMP.
01  FILE-IOAREA.
        05  REC-A-KEY.
                10  REC-A-KEY-CITY     PIC XX.
```

```
            10  REC-A-KEY-SEQ          PIC 999.
        05  REC-A-DETAIL               PIC X(30).
PROCEDURE DIVISION.
        MOVE 'NY000' TO REC-A-KEY.
        EXEC CICS STARTBR
                        DATASET('FILE2')
                        RIDFLD(REC-A-KEY)
                        GTEQ            ← = Default
        END-EXEC.
        MOVE  35 TO WK-LEN.
        EXEC CICS READNEXT DATASET('FILE2')
                        INTO(FILE-IOAREA)
                        RIDFLD(REC-A-KEY)   ←= Required
                        LENGTH(WK-LEN)
        END-EXEC.
```

Execution Results

At the completion of the program logic, the following actions will occur:

- A record will actually be read into the INTO area. That is, the record NY001 will be placed in FILE-IOAREA.
- CICS will place the actual key into the RIDFLD area (REC-A-KEY).
- CICS will place the actual record length into the LENGTH area (WK-LEN).
- If the same READNEXT command is issued again, the record NY002 will be read; that is, the record will be read forward.

Exceptional Conditions

DUPKEY	The key of the record is a duplicate of the next record's key.
ENDFILE	The end of the file is detected.
LENGERR	The actual record length is greater than the length specified.

READPREV Command

Function: The READPREV command is used to read a record of a file backward. The STARTBR command must have been successfully completed prior to issuing the READPREV command.

Format: The format of the READPREV command is shown in the example below. The options and parameters are the same as the READNEXT command.

Example Of 'READPREV' Command

```
WORKING-STORAGE SECTION.
77  WK-LEN                             PIC S9(4) COMP.
01  FILE-IOAREA.
        05  REC-A-KEY.
                10  REC-A-KEY-CITY     PIC XX.
```

```
            10  REC-A-KEY-SEQ          PIC 999.
        05  REC-A-DETAIL               PIC X(30).
PROCEDURE DIVISION.
:
:
        MOVE 'NY000' TO REC-A-KEY.
        EXEC CICS STARTBR
                    DATASET('FILE2')
                    RIDFLD(REC-A-KEY)
                    GTEQ                ←= Default
        END-EXEC.
        MOVE  35 TO WK-LEN.
        EXEC CICS READPREV DATASET('FILE2')
                    INTO(FILE-IOAREA)
                    RIDFLD(REC-A-KEY)   ←= Required
                    LENGTH(WK-LEN)
        END-EXEC.
```

Execution Results

At the completion of this program logic, the following actions will occur:

- Exactly the same results of READNEXT can be applied.
- If the same READPREV command is issued again, the record DC002 will be read. That is, the record will be read backward.

Exceptional Conditions

The exceptional condition common to the READPREV command includes the same conditions mentioned in the READNEXT command. In addition, the following conditions should be watched:

NOTFND The record positioned by the STARTBR command or the RESETBR command is not found. The STARTBR or RESETBR command prior to the READPREV command must specify an existing record as the start key. Otherwise, at the READPREV time, the NOTFND condition will occur, in which case the browse operation must be terminated by the ENDBR command or reset by the RESETBR command.

INVREQ The GENERIC option must not be used in the STARTBR command prior to the READPREV command.

Note: Better option to use READNEXT after STARTBR to avoid NOTFND exceptional condition. Because even if the specified record is not existing, READNEXT will read next record in sequence.

Special Techniques For Sequential Read

There are a few special techniques for sequential read. These are very useful, practical, and convenient techniques in the actual CICS application programming.

Skip Sequential Read

The technique of skip sequential read is used to skip records while continuing the browse operation established by the prior STARTBR command. For the READNEXT command, the new key for skipping must be in the forward direction from the current record. Similarly, for the READPREV command, the new key for skipping must be in the backward direction from the current record.

Example Of Skip Sequential Processing

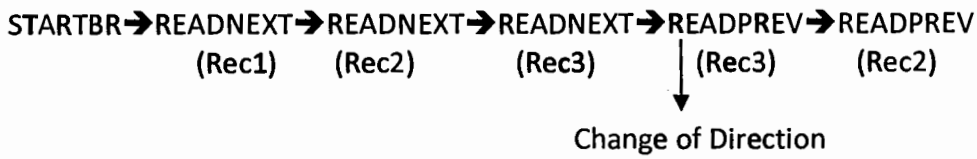
```
MOVE 'NY001' TO REC-A-KEY.
EXEC CICS STARTBR ---------      <= Record NY001 is positioned.
END-EXEC.
:
:
MOVE 35 TO WK-LEN.
EXEC CICS READNEXT --------      <= Record NY001 is read into the file area.
END-EXEC.
:
MOVE 35 TO WK-LEN.
EXEC CICS READNEXT --------      <= Record NY002 is read into the file area.
END-EXEC
:
MOVE 35 TO WK-LEN.
MOVE 'PH000' TO REC-A-KEY.       <= Place a skipping key
EXEC CICS READNEXT --------      <= Record PH000 is read into the file area.
END-EXEC.
```

Execution Results

Record PH000 will be read (that is, NY003 will be skipped), and sequential read can be continued from there.

Changing Direction Of Browse

After the STARTBR command, the direction of browse can be changed from forward to backward by simply switching the READNEXT command to the READPREV command, or vice versa. However, the first READPREV (or READNEXT) command after the direction change will read the same record as the last READNEXT (or READPREV, respectively) command, as illustrated below:



RESETBR Command

Function

The RESETBR command is used to reestablish another starting point within the same browse operation against the same file. The RESETBR command performs exactly the same function as the STARTBR command, except that reposition is much faster because the file is already in the browse mode by the prior STARTBR command. That is without issuing the ENDBR command. The RESETBR command makes it possible to reposition the dataset for a new browse operation.

Similar to the STARTBR command, the RESETBR command does repositioning only. The actual record read will be done by the next READNEXT or READPREV command. The RESETBR command can also be used for changing the characteristics of browse, for example, from generic key positioning to full key positioning, or vice versa.

Format

The format of the RESETBR command is shown in the example below. The options and parameters are the same as the STARTBR command.

Example Of 'RESETBR' Command

```
WORKING-STORAGE SECTION.
77  WK-LEN                          PIC S9(4) COMP.
01  FILE-IOAREA.
        05  REC-A-KEY.
                10  REC-A-KEY-CITY      PIC XX.
                10  REC-A-KEY-SEQ       PIC 999.
        05  REC-A-DETAIL                PIC X(30).
PROCEDURE DIVISION.
:
:
        MOVE 'NY000' TO REC-A-KEY.
        EXEC CICS STARTBR
                        DATASET('FILE2')
                        RIDFLD(REC-A-KEY)
                        GTEQ              ←= Default
        END-EXEC.

        MOVE  35 TO WK-LEN.
        EXEC CICS READNEXT DATASET('FILE2')
                        INTO(FILE-IOAREA)
                        RIDFLD(REC-A-KEY)   ←= Required
                        LENGTH(WK-LEN)
END-EXEC.
**Go for READNEXT until key matches....
**Reset the key
        EXEC CICS RESETBR DATASET('FILE2')
                                RIDFLD(REC-A-KEY)
```

```
                    EQUAL
        END-EXEC.
:
```

Exceptional Conditions

INVREQ The RESETBR command is issued without the prior STARTBR command.

ENDBR Command

Function

At the physical end-of file (no record exists physically further) or logical end-of file (no relevant record exists further), the browse operation must be terminated. The ENDBR command performs this function. That is, ENDBR command is used to terminate the browse operation which was initiated by the prior STARTBR command.

Format

The format of the ENDBR command is shown in the example below. The only required parameter is DATASET, which must name the file to which you wish to terminate the browse operation.

Example Of 'ENDBR' Command

```
EXEC CICS ENDBR
                DATASET('FILE2')
END-EXEC.
```

Multiple Browse Operations

The Multiple Browse Operations are used to perform several concurrent brose operations against the same file. One browse operation is to be identified by the REQID parameter in the browse commands.

As shown in the example below, for multiple browse operations, all browse commands (STARTBR, READNEXT, READPREV, RESETBR, ENDBR) must have the REQID parameter to identify to which browse operation group the command belongs. The other parameters and options of each browse command remain the same as the case of the single browse operation. Once REQID is established, within the same REQID group, the browse operation can be performed in any way as you wish, independently from the other REQID group.

Example Of Multiple Browse.

```
*OPERATION (1)
MOVE 'NY000' TO REC-A-KEY1.
EXEC CICS STARTBR
                DATASET('FILE2')
                REQID(1)
                RIDFLD(REC-A-KEY1)
```

```
                GTEQ
END-EXEC.
*
*OPERATION(2)
MOVE 'DC002' TO REC-A-KEY2.
EXEC CICS STARTBR
                DATASET('FILE2')
                REQID(2)
                RIDFLD(REC-A-KEY2)
                GTEQ
END-EXEC.
*
*OPERATION (3)
MOVE 'B0000' TO REC-A-KEY3.
EXEC CICS STARTBR
                DATASET('FILE2')
                REQID(1)
                RIDFLD(REC-A-KEY3)
                GTEQ
END-EXEC.
*
*OPERATION (1)
MOVE 35 TO WK-LEN1
EXEC CICS READNEXT DATASET('FILE2')
                REDQID(1)
                INTO(FILE-IOAREA1)
                RIDFLD(REC-A-KEY1)
                LENGTH(WK-LEN1)
END-EXEC.
*
* OPERATION (2)
MOVE 35 TO WK-LEN2
EXEC CICS READNEXT DATASET('FILE2')
                REDQID(2)
                INTO(FILE-IOAREA2)
                RIDFLD(REC-A-KEY2)
                LENGTH(WK-LEN2)
END-EXEC.
*
* OPERATION (3)
MOVE 35 TO WK-LEN3
EXEC CICS READNEXT DATASET('FILE2')
                REDQID(3)
                INTO(FILE-IOAREA3)
                RIDFLD(REC-A-KEY3)
                LENGTH(WK-LEN3)
END-EXEC.
```

Execution Result

- For operation 1: NY000 will be read
- For operation 2: DC001 will be read
- For operation 3: B0000 will be read

TEMPORARY STORAGE CONTROL

Introduction

The CICS Temporary Storage Control Program (TSP) provides the application program with an ability to store and retrieve the data in a Temporary Storage Queue (TSQ). Typically TSQ is used for passing of data among transactions.

Characteristics of TSQ

- A Temporary Storage Queue (TSQ) is a queue of stored records (Data).
- It is created and deleted dynamically by an application program without specifying anything in the CICS control tables, as long as data recovery is not intended.
- Therefore, the application program can use TSQ as a scratch pad memory facility for any purposes.
- A TSQ is identified by the queue id (1 to8 bytes), and the relative position number called item number identifies a record within a TSQ.
- The records in TSQ, once written, remain accessible until the entire TSQ is explicitly deleted.
- The records in TSQ can be read sequentially or directly. Also records in TSQ can be updated.
- TSQ may be written in the main storage or the auxiliary storage in the direct access device.
- Regardless of where TSQ resides, TSQ can be accessed by any transactions, in the same CICS region.
- The characteristics of TSQ described above are unique to TSQ in comparison to the Transient Data Queue (TDQ).

Naming Convention for TSQ Queue ID

A TSQ identified by the queue id (1 to 8 bytes) can be accessed by any transactions in the same CICS region. This means that TSQ could be accessed by any of the following transactions:

- Same transaction:
  - From same terminal
  - From different terminal
- Different transaction:
  - From same terminal
  - From different terminal

In order to avoid confusion and to maintain data security, a strict naming convention for QID will be required in the installation. The following is an example of a QID naming convention:

Format: dttttann

Where,

d	:	Division id (E.g., A for Accounts, B for Budget)
tttt	:	Terminal id
a	:	Application code (A, B, C,.....)
nn	:	Queue number(1, 2, ...)

Terminal Id in QID For a terminal dependent task (e.g., pseudo-conversational task), the terminal id should be included in QID in order to ensure the uniqueness of TSQ to the task.

TSQ in MAIN Storage or AUXILIARY Storage

- TSQ can be written in the main storage. In this case, accessing TSQ is a fast and convenient operation. But TSQ in this mode is not recoverable.
- Mutually exclusive to the TSQ in the main storage, TSQ can be written in the auxiliary storage.
- The auxiliary storage is an external VSAM file (DFHTEMP) established by the system programmer.
- It is always available to application programs, which implies that no file open/close is required.
- TSQ in this mode is recoverable. If you which to recover TSQ in this mode, you must specify so in the Temporary Storage Table (TST).

Available Commands

WRITEQ TS	:	To write or rewrite a record in a TSQ.
READQ TS	:	To read a record in a TSQ.
DELETEQ TS	:	To delete a TSQ. All records in the TSQ will be deleted.

WRITEQ Command (No Update)

Function

The WRITEQ TS command is used to write a record (item) in a TSQ.

Format

The format of the WRITEQ TS command is shown in the example below. QUEUE names the queue id (QID). FROM defines the area from which data is to be written. LENGTH indicates the length of the record. A half word binary field (S9(4) COMP) should be provided to the item parameter, to which CICS places the actual item number of the record written. MAIN indicates TSQ to be written in the main storage (not auxiliary storage).

Example Of WRITEQ TS Command

```
EXEC CICS WRITEQ TS
                QUEUE('AttttM01')
                FROM(TSQ-DATA)
                LENGTH(TSQ-LEN)
                ITEM(TSQ-ITEM)
                MAIN
END-EXEC.
```

Execution Results

- If TSQ with this QID does not exist, a TSQ will be created with QID=AttttM01, where tttt is the terminal id. CICS will write a record in the TSQ identified by the QID. CICS will place the actual relative record number (i.e., item number) of the TSQ into TSQ –ITEM. You may have to remember this for later use if more than one record is in the TSQ.
- If TSQ with this QID already exists, CICS will simply write a record in the existing TSQ at the next to the last existing record, and place the relative number of the record into TSQ-ITEM.

Common Options

NOSUSPEND Even if the NOSPACE condition is detected, instead of suspending the task, CICS will return control to the next statement after the WRITEQ command.

AUXILIARY Mutually exclusive to the MAIN option. If specified, the TSQ will be written in the external VSAM file.

Exceptional Conditions

NOSPACE Sufficient space is not available. The default action is to suspend the task until space becomes available.

READQ TS Command (Direct Read)

Function

The READQ TS command is used to read a particular record (item) of a particular TSQ.

Format

The format of the READQ TS command is shown in the example below. QUEUE names the queue id (QID). INTO defines the area to which the record data is to be placed. LENGTH indicates the length of the record to be read. ITEM indicates the item number of the record to be read.

Example Of READQ TS Command (Direct Read)

```
EXEC CICS READQ TS
        QUEUE(TSQ-QID)
```

```
        INTO(TSQ-DATA)
        LENGTH(TSQ-LEN)
        ITEM(TSQ-ITEM)
END-EXEC.
```

Common Options

NUMITEMS If you wish to know the number of items in the TSQ, specify NUMITEMS (data-area), where the data-area is defined as PIC S9(4) COMP. At the completion of the READ command, CICS will place the actual number of records in the TSQ specified.

NEXT Mutually exclusive to the ITEM option, if the NEXT option is specified, the task will read the next sequential record in the TSQ specified.

Exceptional Condition

QIDERR The specified QID is not found

ITEMERR The specified item number is not found. If you are reading TSQ sequentially, this condition occurs at the end of the records of the TSQ.

READQ TS Command (Sequential Read)

Function

The READQ TS command can be used also for reading all records sequentially in the queue. But the sequential read of TSQ should be performed based on the programming techniques, instead of the NEXT option of the READQ TS command.

Format

The format of the READQ TS command for the sequential read is the same as the case of the direct read.

Example Of READQ TS Command (Sequential Read)

```
*ESTABLISH HANDLE CONDITION.
        EXEC CICS HANDLE CONDITION
                ITEMERR(TSQ-EOF)
                ERROR(SQ-ERROR)
        END-EXEC.
LOOP.
        ADD 1 TO TSQ-ITEM.
        MOVE 200 TO TSQ-LEN.
*READ A QUEUE.
        EXEC CICS READQ TS QUEUE(TSQ-QID)
                        INTO(TSQ-DATA)
                        LENGTH(TSQ-LENGTH)
                        ITEM(TSQ-ITEM)
```

```
        END-EXEC.
:
        (PROCESS A RECORD)
:
        GO TO LOOP.
TSQ-EOF.
        (EOF PROCESSING)
        :
```

Execution Results

- At the completion of each READQ TS command, the record will be sequentially read one by one based on the value in TSQ-ITEM.
- When there are no more records to read, the ITEMERR condition (implying that end of TSQ) will be detected through the HANDLE CONDITION command, and control will passed to TSQ-EOF.

Note: The sequential read of TSQ should be done through this approach. This approach is better than specifying the NEXT option in the READQ TS command, which gives the application program the next record to the current record read by any CICS transaction.

This means that if the NEXT option is specified in the READQ TS command instead of the item option for a TSQ, and if other tasks have read several records of the same TSQ after the last read by this task, then this task would read a skipped record (i.e., not next to the record this task read last). Therefore, the NEXT option should be used very carefully, if you have to use it.

WRITEQ TS Command with REWRITE Option

Function

The WRITEQ Ts command with REWRITE option is used to rewrite a record (item) of a TSQ, which has been read.

Format

The WRITEQ TS command with the REWRITE option is shown in the example below. REWRITE must be explicitly specified as the option. Other options and parameters are the same as the case of the WRITEQ TS without update.

Example Of WRITEQ TS Command with REWRITE Option

```
EXEC CICS READQ TS
        QUEUE(TSQ-QID)
        INTO(TSQ-DATA)
        LENGTH(TSQ-LEN)
        ITEM(TSQ-ITEM)
END-EXEC.
```

```
:
*Process data in TSQ-DATA
:
```

Example Of WRITEQ TS Command with REWRITE Option

```
EXEC CICS WRITEQ TS
                QUEUE(TSQ-QID)
                FROM(TSQ-DATA)
                LENGTH(TSQ-LEN)
                ITEM(TSQ-ITEM)
                REWRITE
                MAIN
END-EXEC.
```

Note: The READQ command does not hold exclusive control over TSQ, as you note that there is no UPDATE option in the READQ command. Therefore, if a TSQ is shared by other transactions, it is the application program's responsibility to establish exclusive control over the TSQ during update. In this case, you must use the ENQ command before the READQ command and the DEQ command after the WRITEQ command with the REWRITE option.

DELETEQ TS Command

Function

The DELETEQ TS command is used to delete a TSQ entirely.

Format

The format is shown in the example below. QUEUE names the queue id of TSQ you wish to delete.

Example Of DELETEQ TS Command

```
EXEC CICS DELETEQ TS
        QUEUE(TSQ-QID)
END-EXEC.
```

Exceptional condition

QIDERR The queue id specified is not found.

TRANSIENT DATA CONTROL

Introduction

The CICS transient Data Control Program (TDP) allows a CICS transaction to deal with sequential data called Transient data files. A Transient Data file can be used as either as an input file or an output file, but not both.

The Transient data is sometimes called Transient Data Queue (TDQ), while other times it is called Transient Data Destination. The word "Queue" is used because the records in

the Transient data are put together in the sequential mode. The word "Destination" is used because this sequential data is directed to other transactions. Both terms are used interchangeably and they mean essentially the same.

There are two types of TDQ: Intrapartition TDQ and Extrapartition TDQ. Although the same CICS commands are used for both Intrapartition TDQ and Extrapartition TDQ, the application of these two types of TDQ are very different.
Regardless of the type, a 1-4 characters identifier called "destination-id" identifies each TDQ. All destination ids must be registered in the destination Control table (DCT).

Intrapartition TDQ
An Intrapartition TDQ is a group of sequential records which are produced and processed by the same and/or different transactions within a CICS region. This is why the word "Intrapartition" is used.
All Intrapartition TDQs are stored in only one physical file (VSAM) in a CICS region, which is prepared by the system programmer. From an application program's point of view, however, only sequential access is allowed for a queue. Once a record is read from a queue, the record will be logically removed from the queue; that is the record cannot be read again.
The Intrapartition TDQ is used for the various applications such as:

- Interface among CICS transactions.
  Appl. Program 1 →TDQ->Appl. Program 2→ Report
- Automatic Task Initiation (ATI)
- Message Routing
- Message Broadcast

Extrapartition TDQ
An Extrapartition TDQ is a group of sequential records, which interfaces between the transactions of the CICS region and the systems (or batch jobs) outside of the CICS region. This is why the word "Extrapartition" is used.
Each extrapartition TDQ is a separate physical file, and it may be on the disk, tape, printer, or plotter. This implies that each file (i.e., Extrapartition TDQ) must be open within CICS region when it is used by the CICS transaction.
The Extrapartition TDQ is typically used for the following two applications:

- Interface to batch (or TSO, or PC) jobs.
  CICS Application Program → TDQ → File → Batch Program
- Interface from batch (or TSQ, or PC) jobs.
  Batch Program → File → TDQ → CICS Application Program.

Available Commands

WRITEQ TD:	To sequentially write a record in a TDQ.
READQ TD:	To sequentially read a record in a TDQ.
DELETEQ TD:	To delete an Intrapartition TDQ.

It should be noted that TD represents Transient Data. If this is omitted, Temporary storage (TS) will be assumed.

WRITEQ TD Command

Function

The WRITEQ TD command is used to write a record in a TDQ.

Format

The format of the WRITEQ TD command is shown in the example below. QUEUE names the destination id (1 to 4 characters), which must be defined in DCT. FROM defines the name of the area from which the data is to be written. LENGTH indicates the length of the record.

Example Of WRITEQ TD Command

```
WORKING-STORAGE SECTION.
01 MSG-AREA.
  05 MSG-TO-LOC      PIC (2)  VALUE 'GE'.
  05 FILLER          PIC X    VALUE SPACE.
  05 MSG-OP-FROM     PIC X(5) VALUE 'JIM'.
  05 FILLER          PIC X    VALUE SPACE.
  05 MSG-OP-TO       PIC  X(5).
  05 FILLER          PIC  X   VALUE SPACE.
  05 MSG-MESSAGE     PIC X(40).
77 MSG-LEN           PIC S9(4) COMP.
:
PROCEDURE DIVISION.
:
        MOVE 'JACK' TO MSG-OP-TO.
        MOVE 'HI, THIS IS A TEST MESSAGE SWITCHING' TO MSG-MESSAGE.
        MOVE 55 TO MSG-LEN.
        EXEC CICS WRITEQ TD
                          QUEUE('MSGS')      ← Destination ID
                          FROM(MSG-AREA)
                          LENGTH(MSG-LEN)
        END-EXEC.
```

Exceptional Condition

QIDERR: The destination id specified cannot be found in DCT.

LENGERR: The length specified is greater than the maximum record length specified in DCT.

NOSPACE: No space is available in the TDQ.

READQ TD Command

Function

The READQ TD command is used to read a particular record of a particular TDQ.

Format

The format of the READQ TS command is shown in the example below. QUEUE names the destination id (1 to 4 characters). INTO defines the area to which the record data is to be placed. LENGTH indicates the length of the record to be read.

Example Of READQ TD Command

```
EXEC CICS READQ TD
        QUEUE('MSGS')
        INTO(MSG-AREA)
        LENGTH(MSG-LEN)
END-EXEC.
```

Exceptional Condition

QIDERR: The specified Destination Id is not found

LENGERR: The length specified is shorter than the actual record length. The record will be truncated at the length specified, while that actual length will be placed in th4e LENGTH field.

QZERO: The queue is empty. This often means the end of the file.

DELETEQ TD Command

Function

The DELETEQ TD command is used to delete an Intrapartition TDQ entirely.

Format

The format is shown in the example below. QUEUE names the destination id of TDQ you wish to delete.

Example Of DELETEQ TD Command

```
EXEC CICS DELETEQ TD
        QUEUE('MSGS')
END-EXEC.
```

Exceptional Condition

QIDERR: The destination ID specified cannot be found in DCT.

Destination Control Table (DCT)

Function

The primary function of the Destination Control Table (DCT) is to register control information of all TDQs. The CICS destination Control program (DCP) uses this table for identifying all TDQs and performing input/output operations against them.

Format of DCT Entry for Intrapartition TDQ

```
DFHDCT      TYPE=INTRA,
            DESTID=name,
```

```
[TRANSID=name,],
[TRIGLEV=number,],
[REUSE=YES|NO]
```

TYPE=INTRA indicates that TDQ is the Intrapartition TDQ. DESTID defines the destination id (1 to 4 characters). REUSE option defines space availability after a TDQ record has been read. This is useful for better disk space management. For the Automatic Task Initiation, TRANSID and TRIGLEV must be specified. TRANSID names the transaction id of the transaction to be initiated. TRIGLEV indicates the number of the records in TDQ, which triggers the transaction to be initiated.

Note: One a record of Intrapartition TDQ is read by a transaction, the record is logically removed, but it still occupies the space. If REUSE=YES is specified, this space for the logically deleted record will be used for other TDQ records. But, in this case, records are not recoverable. If REUSE=NO is specified, the logically deleted records are recoverable.

Automatic Task Initiation (ATI)

The Automatic Task Initiation (ATI) is a facility through which a CICS transaction can be initiated automatically. The number of the records in an Intrapartition TDQ triggers the transaction initiation.
The transaction id (task) must be defined in the DCT entry of the Intrapartition TDQ, with nonzero trigger level.

Applications

Following are some of the practical applications of ATI:

- **Message Switching**: messages can be accumulated in an Intrapartition TDQ, and at a certain level, a transaction can be initiated automatically through ATI in order to route the messages.
- **Report Print**: reports can be accumulated in an Intrapartition TDQ, and at a certain level, a transaction can be automatically initiated through ATI in order to print the reports.

INTERVAL CONTROL AND TASK CONTROL

START Command

Function

The START command is used to start a transaction at the specified terminal and at the specified time or interval. Optionally, data can be passed to the to-be-initiated transaction.

Format

The basic format of the START command is shown in the example below. TRANSID defines the transaction id of a transaction, which you wish to initiate. TERMID defines the terminal id of a terminal against which you initiate the transaction. If TERMID is omitted, the specified transaction will be initiated against the terminal with which the current transaction is associated. TIME or INTERVAL indicates the time or the time interval of the transaction initiation, respectively, in the form of hhmmss.

Example of START Command

```
EXEC CICS START
          TRANSID('TRN1')
          TERMID('TRM1')
          TIME(083000)          <= Or INTERVAL(001500)
END-EXEC.
```

Execution Results

Transaction 'TRN1' will be started on terminal (or printer) 'TRM1' at 8:30 (if TIME is specified) or 15 minutes later (if INTERVAL is specified).

Other Options

FROM:	To pass a field in the working storage section
LENGTH:	To specify the length of the 'FROM' field.
QUEUE:	To pass the queue id (8 bytes) of a TSQ

These data must be retrieved by the RETRIEVE command in the to-be-initiated transaction.

RETRIEVE Command

Function

The RETRIEVE command is used to retrieve the data passed by the START command, which was issued in the other transaction in order to initiate this transaction.

Format

The format of the RETRIEVE command is shown in the example below. INTO defines the field in the Working Storage Section to which the data passed by the FROM option of the START command is to be placed. LENGTH indicates the length of the INTO field. QUEUE defines the QID to which the QID passed by the START command to be placed.

Example of RETRIEVE Command

```
1]
EXEC CICS START
          TRANSID('TRN1')
          INTERVAL(001500)
          TERMID('TRM1')
          FROM(DATAFLD)
```

```
                LENGTH(100)
                QUEUE(QNAME)
END-EXEC.
2] In the transaction (TRN1) which is to be initiated by the START command:
EXEC CICS RETRIEVE
                INTO(DATAFLD)
                LENGTH(100)
                QUEUE(RETQUID)
END-EXEC.
```

Execution Results

- At the completion of the START command, the original transaction will initiate the new transaction (TRN1) based on the time specified and the data DATAFLD, transaction-id, terminal-id, and QID will be passed to the new transaction (TRN1).
- 15 minutes later, CICS will actually initiate the INQ1 transaction.
- At the completion of the RETRIEVE command, the INQ1 transaction will receive all specified data passed from the original transaction, which has initiated this INQ1 transaction.

CANCEL Command

Function

The CANCEL command is used to cancel the Interval Control command START, which have been issued. The Interval control commands to be cancelled are identified by the REQID parameter of these commands.

Format

The format of the CANCEL command is shown in the example below. REQID defines the request id (up to 8 chars), which identifies the interval control command to be cancelled.

Example of CANCEL Command

```
EXEC CICS START
                REQID('START1')
                ---------------
END-EXEC.
:
:
:
EXEC CICS CANCEL
                REQID('START1')
END-EXEC.
```

Note: Effective only prior to the expiration time specified in the START command.

SUSPEND Command

Function

The SUSPEND command is used to suspend a task. During the execution of this command, the task will be suspended, and control will be given to other tasks with higher priority. As soon as all higher priority tasks have been executed, control will be returned to the suspended task.

Format of SUSPEND Command

```
EXEC CICS SUSPEND
END-EXEC.
```

Applications

Because of the quasi-reentrancy of the CICS programs, if a CPU intensive processing is performed in a program, it is a good practice to issue the SUSPEND command from time to time in the program in order to allow other tasks to proceed.

ENQ and DEQ Commands

Function

The ENQ command is used to gain exclusive control over a resource. The DEQ command is used to free that exclusive control from the resource.

Format

The format of the ENQ and DEQ are shown in the example below. RESOURCE defines the name of the resource to be ENQ'ed or DEQ'ed.

Example of ENQ and DEQ Command

```
EXEC CICS ENQ RESOURCE(TSQ-QID)
END-EXEC.
EXEC CICS READQ TS QUEUE(TSQ-QID)
END-EXEC.
:
(Update the contents of TSQ)
:
EXEC CICS WRITEQ TS QUEUE(TSQ-QID)
                 REWRITE
END-EXEC.
EXEC CICS DEQ
                 RESOURCE(TSQ-QID)
END-EXEC.
```

Other Options

LENGTH If the resource is a character string, the LENGTH parameter must be specified in both ENQ.DEQ commands.

NOSUSPEND If this is specified, even if the ENQBUSY condition occurs, control will be returned to the statement after the ENQ command.

Exceptional Condition

ENQBUSY The resource specified is reserved to another task. The task will be suspended until the resource is freed by the other task.

TESTS AND DEBUGGING

Introduction

For an effective debugging and/or better management of abnormal termination of a program, CICS provides useful control functions over these areas. The Abnormal Termination Recovery function (ABEND Control) manages an abnormal termination (ABEND) of a task. The Dump Control program (DCP) manages dumping the main storage areas. The CICS Trace Control Program makes use of the Trace Table for debugging aid purposes.

Commands

- Abend Control Commands:
  - **HANDLE ABEND**: To detect an ABEND.
  - **ABEND**: To force an ABEND
- Dump Control Command:
  - **DUMP**: To dump the main storage area

Other Facilities

In addition, in order to facilitate effective tests and production operations, CICS provides the following facilities:

- Transaction Dump
- Execution Diagnostic Facility (EDF)
- Command Level Interpreter (CECI)
- Temporary Storage Browse (CEBR)
- Master Terminal Transaction (CEMT)
- Dynamic File Open/Close (DFOC)

HANDLE ABEND Command

Function

The HANDLE ABEND command is used to intercept an abnormal termination (ABEND) within a program, and to activate, cancel, or reactivate an exit for the ABEND processing. The HANDLE ABEND command is similar to, but different from, the HANDLE CONDITION command, which intercepts only the abnormal conditions of the CICS command execution.

Format of HANDLE ABEND Command

```
EXEC CICS HANDLE ABEND
                [PROGRAM(name)]  | LABEL(label) | CANCEL |  RESET]
END-EXEC.
```

PROGRAM or LABEL is used to activate an exit to a program or a paragraph, respectively, for the ABEND processing. CANCEL is used to cancel the previously established HANDLE ABEND request. RESET is to reactivate the previously cancelled HANDLE ABEND request.

ABEND Command

Function

The ABEND command is used to terminate a task intentionally, causing an ABEND.

Format of ABEND Command

```
EXEC CICS ABEND
        [ABCODE(name) ]
END-EXEC.
```

ABCODE is used to specify the user abend code (1 to 4 characters).

Example of ABEND Command

```
EXEC CICS HANDLE CONDITION
        ERROR(ERROR-RTN)
END-EXEC.
:
:
ERROR-RTN.
        EXEC CICS ABEND
                ABCODE('1234')
        END-EXEC.
```

Execution Results

- If the general error condition occurs, control will be passed to ERROR-RTN through the HANDLE CONDITION command.
- At the completion of the ABEND command, the task will be forcefully terminated with the user ABEND code 1234.

DUMP Command

Function

The DUMP command is used to dump the main storage areas related to the task.

Format of DUMP Command

```
EXEC CICS DUMP
        DUMPCODE(name)
        [FROM(data-area)
         LENGTH(data-value)]
        [TASK]
```

```
        :
        (Other options)
        :
END-EXEC.
```

DUMPCODE names the user dump code (1 to 4 characters). FROM and LENGTH are used to define the name of a particular area of the program which you wish to dump and the length of the area, respectively. TASK indicates that the task-related system areas are to be dumped. There are other options for dumping system areas. If no option is specified, the option TASK is assumed, in which case the following areas will be normally dumped: *TCA, CSA, Trace Table, Program Storage, General Registers.*

CICS TRANSACTIONS

There are total approximate 22 CICS transactions available. Following is the list of few:

- **CSSN** Sign-On
- **CEBR** Temporary Storage Browse
- **CECI** Command Level Interface
- **CEDF** Execution Diagnostic Facility
- **CEMT** Master Terminal Transaction
- **CEDA** Resource Definition Online
- **CSSF** Sign-Off

CSSN

- Enables CICS to associate user with the terminal
- Name and password should be defined in the SignON Table (SNT).

CEBR

The Temporary Storage Browse (CEBR) is a CICS-supplied transaction, which browses Temporary Storage Queue (TSQ). It is a very convenient tool if you wish to display the content of TSQ when you are monitoring an application program through EDF.
Invoking CEBR is simple. You type CEBR and press the ENTER key. The CEBR initial screen will be displayed. Then, you type:
QUEUE xxxxxxxx
Where, xxxxxxxx is the Queue ID of TSQ to be browsed. After pressing ENTER key, the content of the TSQ will be displayed.

CECI

The Command Level Interpreter (CECI) is a CICS-supplied transaction which performs syntax checking of a CICS command. If the syntax is satisfied, it will actually execute the command. This may be useful for interactive patching into the application system. CECS just checks the syntax.

CEMT

The CEMT (Enhanced Master Terminal) transaction is a CICS supplied transaction, which manipulates the CICS environment, such as transactions, programs, files, TSQs, and tasks. It is a menu-driven and easy-to-use transaction, but due to its nature of manipulating the CICS environment, it is usually a restricted transaction which an application programmer or the end-users cannot use freely, but helps application programmers for program testing, monitoring, and/or trouble shooting.

Major Functions

The CEMT transaction performs the following major functions:

- **INQUIRE:** To inquire about the status of CICS environments.
- **SET:** To update the status of CICS environments
- **PERFORM:** For further system operations.

CSSF

- Sign-off Transaction
- Breaks the connection to CICS
- Sends the message to message log
- Writes sign-off message to terminal