COBOL (**CO**mmon **B**usiness **O**riented **L**anguage)

History.

Developed by 1959 by a group called COnference on Data Systems Language (CODASYL). First COBOL compiler was released by December 1959.

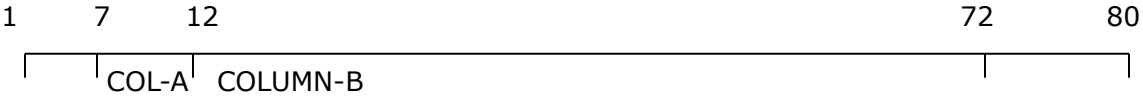
First ANSI approved version – 1968
Modified ANSI approved version – 1974 (OS/VS COBOL)
Modified ANSI approved version – 1985 (VS COBOL 2)

This book is written based on IBM COBOL for OS/390 V2R2.

Speciality.

1. First language developed for commercial application development, which can efficiently handle millions of data.
2. Procedure Oriented Language - Problem is segmented into several tasks. Each task is written as a Paragraph in Procedure Division and executed in a logical sequence as mentioned.
3. English Like language – Easy to learn, code and maintain.

Coding Sheet.



1-6 Page/line numbers – Optional (automatically assigned by compiler)
7 Continuity (-), Comment (*), Starting a new page (/)
Debugging lines (D)
8-11 Column A –Division, Section, Paragraph, 01,77 declarations must begin here.
12.72 Column B –All the other declarations/statements begin here.
73-80 Identification field. It will be ignored by the compiler but visible in the source listing.

Language Structure.

Character	Digits (0-9), Alphabets (A-Z), Space (b), Special Characters (+ - * / ( ) = $; " > < . ,)
Word	One or more characters- User defined or Reserved
Clause	One or more words. It specifies an attribute for an entry
Statement	One or more valid words and clauses
Sentence	One or more statements terminated by a period
Paragraph	One or more sentences.
Section	One or more paragraphs.
Division	One or more sections or paragraphs
Program	Made up of four divisions

Divisions in COBOL.

There are four divisions in a COBOL program and Data division is optional.

1.Identification Division.
2.Environment Division.
3.Data Division.
4.Procedure Division.

Identification Division.

This is the first division and the program is identified here. Paragraph PROGRAM-ID followed by user-defined name is mandatory. All other paragraphs are optional and used for documentation. The length of user-defined name for IBM COBOL is EIGHT.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.       PROGRAM NAME.
AUTHOR.           COMMENT ENTRY.
INSTALLATION.     COMMENT ENTRY.
DATE-WRITTEN.     COMMENT ENTRY.
DATE-COMPILED.    COMMENT ENTRY.
SECURITY.         COMMENT ENTRY.
```

Security does not pertain to the operating system security, but the information that is passed to the user of the program about the security features of the program.

Environment Division.

Only machine dependant division of COBOL program. It supplies information about the hardware or computer equipment to be used on the program. When your program moves from one computer to another computer, the only section that may need to be changed is ENVIRONMENT division.

Configuration Section.

It supplies information concerning the computer on which the program will be compiled (SOURCE-COMPUTER) and executed (OBJECT-COMPUTER). It consists of three paragraphs – SOURCE COMPUTER, OBJECT-COMPUTER and SPECIAL-NAMES. This is OPTIONAL section from COBOL 85.

SOURCE-COMPUTER. IBM-4381 (Computer and model # supplied by manufacturer)
WITH DEBUGGING MODE clause specifies that the debugging lines in the program (statements coded with 'D' in column 7) are compiled.

OBJECT-COMPUTER. IBM-4381 (Usually same as source computer)

SPECIAL-NAMES. This paragraph is used to relate hardware names to user-specified mnemonic names.

1. Substitute character for currency sign. (CURRENCY SIGN IS litearal-1)
2. Comma can be used as decimal point. (DECIMAL-POINT IS COMMA)
3. Default collating sequence can be changed. It will be explained later.
4. New class can be defined using CLASS keyword. (CLASS *DIGIT* is "0" thru "9")

Input-Output Section.

It contains information regarding the files to be used in the program and it consists of two paragraphs FILE-CONTROL & I-O CONTROL.
FILE CONTROL. Files used in the program are identified in this paragraph.
I-O CONTROL. It specifies when check points to be taken and storage areas that are shared by different files.

Data Division.

Data division is used to define the data that need to be accessed by the program. It has three sections.

FILE SECTION	describes the record structure of the files.
WORKING-STORAGE SECTION	is used to for define intermediate variables.
LINKAGE SECTION	is used to access the external data. Ex: Data passed from other programs or from PARM of JCL.

Literals, Constants, Identifier,

1. Literal is a constant and it can be numeric or non-numeric.
2. Numeric literal can hold 18 digits and non-numeric literal can hold 160 characters in it. (COBOL74 supports 120 characters only)
3. Literal stored in a named memory location is called as variable or identifier.
4. Figurative Constant is a COBOL reserved word representing frequently used constants. They are ZERO/ZEROS/ZEROES, QUOTE/QUOTES, SPACE/SPACES, ALL, HIGH-VALUE/HIGH-VALUES, LOW-VALUE/LOW-VALUES.

Example: 01 WS-VAR1 PIC X(04) VALUE 'MUSA'.
'MUSA ' is a non-numeric literal. WS-VAR1 is a identifier or variable.

Declaration of variable

Level# $ Variable $ Picture clause $ Value clause $ Usage Clause $ Sync clause.
FILLER

Level#

It specifies the hierarchy of data within a record. It can take a value from the set of integers between 01-49 or from one of the special level-numbers 66 77 88

01 level.	Specifies the record itself. It may be either a group item or an Elementary item. It must begin in Area A.
02-49 levels.	Specify group or elementary items within a record. Group level items must not have picture clause.
66 level.	Identify the items that contain the RENAMES clause.
77 level.	Identify independent data item.
88 level.	Condition names.

Variable name and Qualifier

Variable name can have 1-30 characters with at least one alphabet in it. Hyphen is the only allowed special character but it cannot be first or last letter of the name. Name should be unique within the record. If two variables with same name are there, then use OF qualifier of high level grouping to refer a variable uniquely.
Ex: MOVE balance *OF* record-1 TO balance *OF* record-2.

FILLER

When the program is not intended to use selected fields in a record structure, define them as FILLER. FILLER items cannot be initialized or used in any operation of the procedure division.

PICTURE Clause

Describes the attributes of variable.

Numeric	9 (Digit), V (Implied decimal point), S (Sign)
Numeric Edited	+ (Plus Sign), - (Minus Sign), CR DB (Credit Debit Sign) . (Period), b (Blank), ‘,’(comma), 0 (Zero), / (Slash) BLANK WHEN ZERO (Insert blank when data value is 0), Z (ZERO suppression), * (ASTERISK), $(Currency Sign)
Non Numeric	A (alphabet), B (Blank insertion Character), X(Alpha numeric), G(DBCS)
Exclusive sets	1. + - CR DB 2. V ‘.’ 3. $ + - Z * (But $ Can appear as first place and * as floating. $***.**)

DBCS (Double Byte Character Set) is used in the applications that support large character sets. 16 bits are used for one character. Ex: Japanese language applications.

Refreshing Basics

Nibble. 4 Bits is one nibble. In packed decimal, each nibble stores one digit.
Byte. 8 Bits is one byte. By default, every character is stored in one byte.
Half word. 16 bits or 2 bytes is one half word. (MVS)
Full word. 32 bits or 4 bytes is one full word. (MVS)
Double word. 64 bits or 8 bytes is one double word. (MVS)

Usage Clause

DISPLAY Default. Number of bytes required equals to the size of the data item.
COMP Binary representation of data item.
PIC clause can contain S and 9 only.
S9(01) – S9(04) Half word.
S9(05) – S9(09) Full word.
S9(10) - S9(18) Double word.
Most significant bit is ON if the number is negative.
COMP-1 Single word floating point item. PIC Clause should not be specified.
COMP-2 Double word floating-point item. PIC Clause should not be specified.
COMP-3 Packed Decimal representation. Two digits are stored in each byte.
Last nibble is for sign. (F for unsigned positive, C for signed positive and D for signed negative)
Formula for Bytes: Integer ((n/2) + 1)) => n is number of 9s.
INDEX It is used for preserve the index value of an array. PIC Clause should not be specified.

VALUE Clause

It is used for initializing data items in the working storage section. Value of item must not exceed picture size. It cannot be specified for the items whose size is variable.

Syntax: VALUE IS literal.
VALUES ARE literal-1 THRU | THROUGH literal-2
VALUES ARE literal-1, literal-2

Literal can be numeric without quotes OR non-numeric within quotes OR figurative constant.

SIGN Clause

Syntax SIGN IS (LEADING) SEPARATE CHARACTER (TRAILING).

It is applicable when the picture string contain 'S'. Default is TRAILING WITH NO SEPARATE CHARACTER. So 'S' doesn't take any space. It is stored along with last digit.

+1=A +2=B +3=C +4=D +5=E +6=F +7=G +8=H +9=I
-0=}, -1= J, -2= K, -3=L, -4=M, -5=N, -6=O, -7=P, -8=Q, -9=R

Number	TRAILING SIGN (Default)	LEADING SIGN	LEADING SEPARATE.
-125	12N	J25	-125
+125	12E	A25	+125

SYNC Clause and Slack Bytes

SYNC clause is specified with COMP, COMP-1 and COMP-2 items. These items are expected to start at half/full/double word boundaries for faster address resolution. SYNC clause does this but it may introduce slack bytes (unused bytes) before the binary item.

```
01 WS-TEST.
    10 WS-VAR1 PIC X(02).
    10 WS-VAR2 PIC S9(6) COMP SYNC.
```

Assumes WS-TEST starts at relative location 0 in the memory, WS-VAR1 occupies zero and first byte. WS-VAR2 is expected to start at second byte. As the comp item in the example needs one word and it is coded with SYNC clause, it will start only at the next word boundary that is 4th byte. So this introduces two slack bytes between WS-VAR1 and WS-VAR2.

REDEFINES

The REDEFINES clause allows you to use different data description entries to describe the same computer storage area. Redefining declaration should immediately follow the redefined item and should be done at the same level. Multiple redefinitions are possible. Size of redefined and redefining need not be the same.

Example:

```
    01 WS-DATE PIC 9(06).
    01 WS-REDEF-DATE REDEFINES WS-DATE.
        05   WS-YEAR   PIC 9(02).
        05   WS-MON    PIC 9(02).
        05   WS-DAY    PIC 9(02).
```

RENAMES

It is used for regrouping of elementary data items in a record. It should be declared at 66 level. It need not immediately follows the data item, which is being renamed. But all RENAMES entries associated with one logical record must immediately follow that record's last data description entry. RENAMES cannot be done for a 01, 77, 88 or another 66 entry.

```
01 WS-REPSONSE.
    05 WS-CHAR143      PIC X(03).
    05 WS-CHAR4        PIC X(04).
    66 ADD-REPSONSE RENAMES WS-CHAR143.
```

CONDITION name

It is identified with special level '88'. A condition name specifies the value that a field can contain and used as abbreviation in condition checking.

```
01 SEX PIC X.
    88 MALE    VALUE '1'
    88 FEMALE VALUE '2' '3'.
```

IF SEX=1 can also be coded as IF MALE in Procedure division.

'SET FEMALE TO TRUE ' moves value 2 to SEX. If multiple values are coded on VALUE clause, the first value will be moved when it is set to true.

JUSTIFIED RIGHT

This clause can be specified with alphanumeric and alphabetic items for right justification. It cannot be used with 66 and 88 level items.

OCCURS Clause

OCCURS Clause is used to allocate physically contiguous memory locations to store the table values and access them with subscript or index. Detail explanation is given in Table Handling section.

LINKAGE SECTION

It is used to access the data that are external to the program. JCL can send maximum 100 characters to a program thru PARM. Linkage section MUST be coded with a half word binary field, prior to actual field. If length field is not coded, the first two bytes of the field coded in the linkage section will be filled with length and so there are chances of 2 bytes data truncation in the actual field.

```
01 LK-DATA.
05 LK-LENGTH         PIC S9(04) COMP.
05 LK-VARIABLE       PIC X(08).
```

LINKAGE section of sub-programs will be explained later.

Procedure Division.

This is the last division and business logic is coded here. It has user-defined sections and paragraphs. Section name should be unique within the program and paragraph name should be unique within the section.

Procedure division statements are broadly classified into following categories.

Statement Type	Meaning
Imperative	Direct the program to take a specific action. Ex: MOVE ADD EXIT GO TO
Conditional	Decide the truth or false of relational condition and based on it, execute different paths. Ex: IF, EVALUATE
Compiler Directive	Directs the compiler to take specific action during compilation. Ex: COPY SKIP EJECT
Explicit Scope terminator	Terminate the scope of conditional and imperative statements. Ex: END-ADD END-IF END-EVALUATE
Implicit Scope terminator	The period at the end of any sentence, terminates the scope of all previous statements not yet terminated.

MOVE Statement

It is used to transfer data between internal storage areas defined in either file section or working storage section.

Syntax:

MOVE identifier1/literal1/figurative-constant TO identifier2 (identifier3)

Multiple move statements can be separated using comma, semicolons, blanks or the keyword THEN.

Numeric move rules:

A numeric or numeric-edited item receives data in such a way that the decimal point is aligned first and then filling of the receiving field takes place.

Unfilled positions are filled with zero. Zero suppression or insertion of editing symbols takes places according to the rules of editing pictures.

If the receiving field width is smaller than sending field then excess digits, to the left and/or to the right of the decimal point are truncated.

Alphanumeric Move Rules:

Alphabetic, alphanumeric or alphanumeric-edited data field receives the data from left to right. Any unfilled field of the receiving filed is filled with spaces.

When the length of receiving field is shorter than that of sending field, then receiving field accepts characters from left to right until it is filled. The unaccomodated characters on the right of the sending field are truncated.

When an alphanumeric field is moved to a numeric or numeric-edited field, the item is moved as if it were in an unsigned numeric integer mode.

CORRESPONDING can be used to transfer data between items of the same names belonging to different group-items by specifying the names of group-items to which they belong.

MOVE CORRESPONDING group-1 TO group-2

Group Move rule

When MOVE statement is used to move information at group level, the movement of data takes place as if both sending and receiving fields are specified as alphanumeric items. This is regardless of the description of the elementary items constituting the group item.

Samples for understanding MOVE statement (MOVE A TO B)

Picture of A	Value of A	Picture of B	Value of B after Move
PIC 99V99	12.35	PIC 999V99	012.35
PIC 99V99	12.35	PIC 9999V9999	0012.3500
PIC 99V999	12.345	PIC 9V99	2.34
PIC9(05)V9(03)	54321.543	PIC 9(03)V9(03)	321.543
PIC 9(04)V9(02)	23.24	PIC ZZZ99.9	23.2
PIC 99V99	00.34	PIC $$$.99	$.34
PIC X(04)	MUSA	XBXBXB	M U S

ARITHMETIC VERBS

All the possible arithmetic operations in COBOL using ADD, SUBTRACT, MULTIPLY and DIVIDE are given below:

Arithmetic Operation	A	B	C	D
ADD A TO B	A	A+B		
ADD A B C TO D	A	B	C	A+B+C+D
ADD A B C GIVING D	A	B	C	A+B+C
ADD A TO B C	A	A+B	A+C	
SUBTRACT A FROM B	A	B-A		
SUBTRACT A B FROM C	A	B	C-(A+B)	
SUBTRACT A B FROM C GIVING D	A	B	C	C-(A+B)
MULTIPLY A BY B	A	A*B		
MULTIPLY A BY B GIVING C	A	B	A*B	
DIVIDE A INTO B	A	B/A		
DIVIDE A INTO B GIVING C	A	B	B/A	
DIVIDE A BY B GIVING C	A	B	A/B	
DIVIDE A INTO B GIVING C REMAINDER D	A	B	Integer (B/A)	Integer remainder

GIVING is used in the following cases:

1.To retain the values of operands participating in the operation.

2.The resultant value of operation exceeds any of the operand size.

ROUNDED option

With ROUNDED option, the computer will always round the result to the PICTURE clause specification of the receiving field. It is usually coded after the field to be rounded. It is prefixed with REMAINDER keyword ONLY in DIVIDE operation.

ADD A B GIVING C ROUNDED.

DIVIDE..ROUNDED REMAINDER

Caution: Don't use for intermediate computation.

ON SIZE ERROR

If A=20 (PIC 9(02)) and B=90 (PIC 9(02)), ADD A TO B will result 10 in B where the expected value in B is 110. ON SIZE ERROR clause is coded to trap such size errors in arithmetic operation.

If this is coded with arithmetic statement, any operation that ended with SIZE error will not be carried out but the statement follows ON SIZE ERROR will be executed.

ADD A TO B ON SIZE ERROR DISPLAY 'ERROR!'.

COMPUTE

Complex arithmetic operations can be carried out using COMPUTE statement. We can use arithmetic symbols than keywords and so it is simple and easy to code. + For ADD, - for SUBTRACT, * for MULTIPLY, / for DIVIDE and ** for exponentiation.

Rule: Left to right – 1.Parentheses
2.Exponentiation
3.Multiplication and Division
4.Addition and Subtraction

Caution: When ROUNDED is coded with COMPUTE, some compiler will do rounding for every arithmetic operation and so the final result would not be precise.

77 A PIC 999 VALUE 10
COMPUTE A ROUNDED = (A+2.95) *10.99
Result: (ROUNDED(ROUNDED(12.95) * ROUNDED(10.99)) =120 or
ROUNDED(142.3205) = 142

So the result can be 120 or 142.Be cautious when using ROUNDED keyword with COMPUTE statement.

All arithmetic operators have their own explicit scope terminators. (END-ADD, END-SUBTRACT, END-MULTIPLY, END-DIVIDE, END-COMPUTE). It is suggested to use them.

CORRESPONDING is available for ADD and SUBTRACT only.

INITIALIZE

VALUE clause is used to initialize the data items in the working storage section whereas INITIALIZE is used to initialize the data items in the procedure division.

INITIALIZE sets the alphabetic, alphanumeric and alphanumeric-edited items to SPACES and numeric and numeric-edited items to ZERO. This can be overridden by REPLACING option of INITIALIZE. FILLER, OCCURS DEPENDING ON items are not affected.

Syntax: INITIALIZE identifier-1
REPLACING (ALPHABETIC/ALPHANUMERIC/ALPHA-NUMERIC-EDITED
NUMERIC/NUMERIC-EDITED)
DATA BY (identifier-2 /Literal-2)

ACCEPT

ACCEPT can transfer data from input device or system information contain in the reserved data items like DATE, TIME, DAY.

ACCEPT WS-VAR1 (FROM DATE/TIME/DAY/OTHER SYSTEM VARS).

If FROM Clause is not coded, then the data is read from terminal. At the time of execution, batch program will ABEND if there is no in-stream data from JCL and there is no FROM clause in the ACCEPT clause.

DATE option returns six digit current date in YYYYMMDD
DAY returns 5 digit current date in YYDDD
TIME returns 8 digit RUN TIME in HHMMSSTT
DAY-OF-WEEK returns single digit whose value can be 1-7 (Monday-Sunday respectively)

DISPLAY

It is used to display data. By default display messages are routed to SYSOUT.
Syntax: DISPLAY identifier1| literal1 (UPON mnemonic name)

STOP RUN, EXIT PROGRAM & GO BACK

STOP RUN is the last executable statement of the main program. It returns control back to OS.

EXIT PROGRAM is the last executable statement of sub-program. It returns control back to main program.

GOBACK can be coded in main program as well as sub-program as the last statement. It just gives the control back from where it received the control.

Collating Sequence

There are two famous Collating Sequence available in computers. IBM and IBM Compatible machine use EBCDIC collating sequence whereas most micro and many mainframe systems use ASCII collating sequence. The result of arithmetic and alphabetic comparison would be same in both collating sequences whereas the same is not true for alphanumeric comparison.

EBCDIC (Ascending Order)	ASCII (Ascending Order)
Special Characters	Special Characters
a-z	0-9
A-Z	A-Z
0-9	a-z

Default collating sequence can be overridden by an entry in OBJECT-COMPUTER and SPECIAL NAMES paragraphs.

1. Code the PROGRAM COLLATING SEQUENCE Clause in the Object computer paragraph. PROGRAM COLLATING SEQUENCE IS alphabet-name

2. Map the alphabet-name in the SPECIAL-NAMES paragraph as follows:
ALPHABET alphabet-name is STANDARD-1 | NATIVE
NATIVE stands for computer's own collating sequence whereas STANDARD-1 stands for ASCII collating sequence.

IF/THEN/ELSE/END-IF

The most famous decision making statement in all language is 'IF'. The syntax of IF statement is given below: IF can be coded without any ELSE statement. THEN is a noise word and it is optional.

If ORs & ANDs are used in the same sentence, ANDs are evaluated first from left to right, followed by ORs. This rule can be overridden by using parentheses.

The permitted relation conditions are =, <, >, <=, >=, <>

CONTINUE is no operation statement. The control is just passed to next STATEMENT. NEXT SENTENCE passes the control to the next SENTENCE. If you forgot the difference between statement and sentence, refer the first page.

It is advised to use END-IF, explicit scope terminator for the IF statements than period, implicit scope terminator.

```
IF condition1 AND condition2 THEN
    Statement-Block-1
ELSE
    IF condition3 THEN
        CONTINUE
    ELSE
        IF condition4 THEN
            Statement-Block-2
        ELSE
            NEXT SENTENCE
        END-IF
    END-IF
END-IF
```

Statement-Block-2 will be executed only when condition 1, 2 and 4 are TRUE and condition 3 is FALSE.

Implied operand: In compound conditions, it is not always necessary to specify both operands for each condition. IF TOTAL=7 or 8 is acceptable. Here TOTAL=8 is implied operation.

SIGN test and CLASS test

SIGN test is used to check the sign of a data item. It can be done as follows –

```
IF identifier is POSITIVE/NEGATIVE/ZERO
```

CLASS test is used to check the content of data item against pre-defined range of values. It can be done as follows -

```
IF identifier is NUMERIC/ALPHABETIC/ALPHABETIC-HIGHER/
                ALPHABETIC-LOWER
```

You can define your own classes in the special names paragraph. We have defined a class DIGIT in our special names paragraph. It can be used in the following way.

```
IF identifier is DIGIT
```

Negated conditions.

Any simple, relational, class, sign test can be negated using NOT.

But it is not always true that NOT NEGATIVE is equal to POSITIVE. (Example ZERO)

EVALUATE

With COBOL85, we use the EVALUATE verb to implement the case structure of other languages. Multiple IF statements can be efficiently and effectively replaced with EVALUATE statement. After the execution of one of the when clauses, the control is automatically come to the next statement after the END-EVALUATE. Any complex condition can be given in the WHEN clause. Break statement is not needed, as it is so in other languages.

General Syntax

```
EVALUATE subject-1 (ALSO subject2..)
    WHEN object-1 (ALSO object2..)
    WHEN object-3 (ALSO object4..)
    WHEN OTHER imperative statement
END--EVALUATE
```

1.Number of Subjects in EVALUATE clause should be equal to number of objects in every WHEN clause.

2.Subject can be variable, expression or the keyword TRUE/ FLASE and respectively objects can be values, TRUE/FALSE or any condition.

3.If none of the WHEN condition is satisfied, then WHEN OTHER path will be executed.

Sample

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

PERFORM STATEMENTS

PERFORM will be useful when you want to execute a set of statements in multiple places of the program. Write all the statements in one paragraph and invoke it using PERFORM wherever needed. Once the paragraph is executed, the control comes back to next statement following the PERFORM.

1.SIMPLE PERFORM.

```
PERFORM PARA-1.
DISPLAY 'PARA-1 executed'
STOP RUN.
PARA-1.
        Statement1
        Statement2.
```

It executes all the instructions coded in PARA-1 and then transfers the control to the next instruction in sequence.

2.INLINE PERFORM.

When sets of statements are used only in one place then we can group all of them within PERFORM END-PERFORM structure. This is called INLINE PERFORM. This is equal to DO..END structure of other languages.

```
PERFORM
        ADD A TO B
        MULTIPLE B BY C
        DISPLAY 'VALUE OF A+B*C ` C
END-PERFORM
```

3. PERFORM PARA-1 THRU PARA-N.

All the paragraphs between PARA-1 and PARA-N are executed once.

4. PERFORM PARA-1 THRU PARA-N UNTIL condition(s).

The identifiers used in the UNTIL condition(s) must be altered within the paragraph(s) being performed; otherwise the paragraphs will be performed indefinitely. If the condition in the UNTIL clause is met at first time of execution, then named paragraph(s) will not be executed at all.

5. PERFORM PARA-1 THRU PARA-N N TIMES.

N can be literal defined as numeric item in working storage or hard coded constant.

6. PERFORM PARA-1 THRU PARA-N VARYING identifier1
FROM identifier 2 BY identifier3 UNTIL condition(s)

Initialize identifier1 with identifier2 and test the condition(s). If the condition is false execute the statements in PARA-1 thru PARA-N and increment identifier1 BY identifier3 and check the condition(s) again. If the condition is again false, repeat this process till the condition is satisfied.

7.PERFORM PARA-1 WITH TEST BEFORE/AFTER UNTIL condition(s).

With TEST BEFORE, Condition is checked first and if it found false, then PARA-1 is executed and this is the default. (Functions like DO- WHILE)

With TEST AFTER, PARA-1 is executed once and then the condition is checked. (Functions like DO-UNTIL)

Refer Table session for eighth type of PERFORM.

EXIT statement.

COBOL reserved word that performs NOTHING. It is used as a single statement in a paragraph that indicate the end of paragraph(s) execution.
EXIT must be the only statement in a paragraph in COBOL74 whereas it can be used with other statements in COBOL85.

GO TO Usage:

In a structured top-down programming GO TO is not preferable. It offers permanent control transfer to another paragraph and the chances of logic errors is much greater with GO TO than PERFORM. The readability of the program will also be badly affected.

But still GO TO can be used within the paragraphs being performed. i.e. When using the THRU option of PERFORM statement, branches or GO TO statements, are permitted as long as they are within the range of named paragraphs.

```
PERFORM 100-STEP1 THRU STEP-4
..
100-STEP-1.
        ADD A TO B GIVING C.
        IF D = ZERO DISPLAY 'MULTIPLICATION NOT DONE'
                        GO TO 300-STEP3
        END-IF.
200-STEP-2.
        MULTIPLY C BY D.
300-STEP-3.
        DISPLAY 'VALUE OF C:' C.
```

Here GO TO used within the range of PERFORM. This kind of Controlled GO TO is fine with structured programming also!

TABLES

An OCCURS clause is used to indicate the repeated occurrences of items of the same format in a structure. OCCURS clause is not valid for 01, 77, 88 levels. It can be defined as elementary or group item. Initialization of large table occurrences with specific values are usually done using perform loops in procedure division. Simple tables can be initialized in the following way.

```
01 WEEK-ARRAY VALUE 'MONTUEWEDTHUFRISATSUN'.
    05 WS-WEEK-DAYS OCCURS 7 TIMES PIC X(03).
```

Dynamic array is the array whose size is decided during runtime just before the access of first element of the array.

```
01 WS-MONTH-DAY-CAL.
        05 WS-DAYS OCCURS 31 TIMES DEPENDING ON WS-OCCURENCE.
```

IF MONTH = 'FEB' MOVE '28' to WS-OCCURRENCE.

Array Items can be accessed using INDEX or subscript and the difference between them are listed in the table. Relative subscripts and relative indexes are supported only in COBOL85. Literals used in relative subscripting/indexing must be an unsigned integer.

```
ADD WS-SAL(SUB) WS-SAL(SUB + 1) TO WS-SAL(SUB + 2).
```

Sl #	Subscript	Index
1	Working Storage item	Internal Item – No need to declare it.
2	It means occurrence	It means displacement
3	Occurrence, in turn translated to displacement to access elements and so slower than INDEX access.	Faster and efficient.
4	It can be used in any arithmetic operations or for display.	It cannot be used for arithmetic operation or for display purpose.
5	Subscripts can be modified by any arithmetic statement.	INDEX can only be modified with SET, SEARCH and PERFORM statements.

Sometimes, you may face a question like how to randomly access the information in the sequential file of 50 records that contains all the designation and the respective lower and higher salary information.

Obviously, OS does not allow you to randomly access the sequence file. You have to do by yourself and the best way is, load the file into a working storage table in the first section of the program and then access as you wish.

The table look-up can be done in two ways.

-Sequential search.

-Binary search.

Sequential SEARCH

During SERIAL SEARCH, the first entry of the table is searched. If the condition is met, the table look-up is completed. If the condition is not met, then index or subscript is incremented by one and the next entry is searched and the process continues until a match is found or the table has been completely searched.

```
SET indexname-1 TO 1.
SEARCH identifier-1 AT END display 'match not found:'
  WHEN condition-1 imperative statement-1 /NEXT SENTENCE
  WHEN condition-2 imperative statement-2 /NEXT SENTENCE
END-SEARCH
```

Identifier-1 should be OCCURS item and not 01 item.
Condition-1, Condition-2 compares an input field or search argument with a table argument.
Though AT END Clause is optional, it is highly recommended to code that. Because if it is not coded and element looking for is not found, then the control simply comes to the next statement after SEARCH where an invalid table item can be referred and that may lead to incorrect results / abnormal ends.

SET statement Syntax:

```
SET index-name-1 TO/UP BY/DOWN BY integer-1.
```

Binary SEARCH

When the size of the table is large and it is arranged in some sequence – either ascending or descending on search field, then BINARY SEARCH would be the efficient method.

```
SEARCH ALL identifier-1 AT END imperative-statement-1
        WHEN dataname-1 = identifier-2/literal-1/arithmetic expression-1
        AND dataname-2 = identifier-3/literal-2/arithmetic expression-2
END-SEARCH.
```

Identifier-2 and identifier-3 are subscripted items and dataname-1 and dataname-2 are working storage items that are not subscripted.

Compare the item to be searched with the item at the center. If it matches fine, else repeat the process with the left or right half depending on where the item lies.

Sl #	Sequential SEARCH	Binary SEARCH
1	SEARCH	SEARCH ALL
2	Table should have INDEX	Table should have INDEX
3	Table need not be in SORTED order.	Table should be in sorted order of the searching argument. There should be ASCENDING/DESCENDING Clause.
4	Multiple WHEN conditions can be coded.	Only one WHEN condition can be coded.
5.	Any logical comparison is possible.	Only = is possible. Only AND is possible in compound conditions.
6	Index should be set to 1 before using SEARCH	Index need not be set to 1 before SEARCH ALL.
7	Prefer when the table size is small	Prefer when the table size is significantly large.

Multi Dimensional Arrays

COBOL74 supports array of maximum of three dimensions whereas COBOL85 supports up to seven dimensions. The lowest- level OCCURS data-name or an item subordinate to it is used to access an entry in the array or the table.

If we use SEARCH for accessing multi-dimension table, then INDEXED BY must be used on all OCCURS levels. Expanded nested perform is available for processing multi level tables. The syntax of this perform is given below:

```
PERFORM para-1 thru para-n
  VARYING index-1 from 1 BY 1 UNTIL index-1 > size- of- outer-occurs
  AFTER VARYING index-2 from 1 by 1 until index-2 > size of inner occurs.
```

SEARCH example for multi level tables:

```
01 EMP-TABLE.
        05 DEPTNUMBER OCCURS 10 TIMES INDEXED BY I1.
                10 EMP-DETAIL OCCURS 50 TIMES INDEXED BY I2.
                        15 EMP-NUMBER  PIC 9(04).
                        15 EMP-SALARY   PIC 9(05).
77 EMPNUMBER-IN                         PIC 9(04) VALUE '2052'.
PERFORM 100-SEARCH-EMP-SAL VARYING I1 FROM 1 BY 1
    UNTIL  I1 > 10 OR WS-FOUND
100-SEARCH-EMP-SAL.
        SET I2 TO 1.
        SEARCH EMP-DETAIL AT END DISPLAY 'NOT FOUND'     == > Lowest Occurs
           WHEN EMPNUMBER-IN = EMP-NUMBER(I1,I2)
                    DISPLAY 'SALARY IS:' EMP-SALARY(I1,I2)
                     SET WS-FOUND TO TRUE                        == > Search ends
        END-SEARCH.
```

NESTED PROGRAMS, GLOBAL, EXTERNAL

One program may contain other program(s). The contained program(s) may themselves contain yet other program(s). All the contained and containing programs should end with END PROGRAM statement. PGMB is nested a program in the example below:

```
Example:        IDENTIFICATION DIVISION.
                 PROGRAM-ID. PGMA
                 …
                IDENTIFICATION DIVISION.
                PROGRAM-ID. PGMB
                …
                END PROGRAM PGMB.
                …
                END PROGRAM PGMA.
```

If you want access any working storage variable of PGMA in PGMB, then declare them with the clause 'IS GLOBAL' in PGMA. If you want to access any working storage variable of PGMB in PGMA, declare them with the clause 'IS EXTERNAL' in PGMB. Nested Programs are supported only in COBOL85.

If there is a program PGMC inside PGMB, it cannot be called from PGMA unless it's program id is qualified with keyword COMMON.

SORT and MERGE

The programming SORT is called as internal sort whereas the sort in JCL is called external sort. If you want to manipulate the data before feeding to sort, prefer internal sort. In all other cases, external sort is the good choice. Internal sort, in turn invokes the SORT product of your installation. (DFSORT). In the run JCL, allocate at least three sort work files. (SORT-WKnn => nn can be 00-99).

FASTSRT compiler option makes the DFSORT to do all file I-O operation than your COBOL program. It would significantly improve the performance. The result of the SORT can be checked in SORT-RETURN register. If the sort is successful, the value will be 0 else 16.

Syntax:

```
SORT SORTFILE ON ASCENDING /DESCENDING KEY sd-key-1 sd-key2
  USING file1 file2 / INPUT PROCEDURE IS section-1
  GIVING file3       / OUTPUT PROCEDURE is section-2
END-SORT
```

File1, File2 are to-be-sorted input files and File3 is sorted-output file and all of them are defined in FD.SORTFILE is Disk SORT Work file that is defined at SD. It should not be explicitly opened or closed.

INPUT PROCEDURE and USING are mutually exclusive. If USING is used, then file1 and files should not be opened or READ explicitly. If INPUT PROCEDURE is used then File1 and file2 need to be OPENed and READ the records one by one until end of the file and pass the required records to sort-work-file using the command RELEASE. Syntax: RELEASE sort-work-record from input-file-record.

OUTPUT Procedure and GIVING are mutually exclusive. If GIVING is used, then file3 should not be opened or WRITE explicitly. If OUTPUT procedure is used, then File3 should be OPENed and the required records from sort work file should be RETURNed to it. Once AT END is reached for sort-work-file, close the output file. Syntax: RETURN sort-work-file-name AT END imperative statement.

INPUT PROCEDURE Vs OUTPUT PROCEDURE:

Sometimes it would be more efficient to process data before it is sorted, whereas other times it is more efficient to process after it is sorted. If we intend to eliminate more records, then it would be better preprocess them before feeding to SORT. If we want to eliminate all the records having spaces in the key field then it would be efficient if we eliminate them after sorting. Because the records with blank key comes first after sorting.

MERGE

It is same as sort. USING is mandatory. There should be minimum two files in USING.

```
MERGE Sort-work-file ON ASCENDING KEY dataname1 dataname2
  USING file1 file2
  GIVING file3 / OUTPUT PROCEDURE is section-1
END-MERGE
```

Program sort registers (and its equivalent DFSORT parameter/meaning)

SORT-FILE-SIZE (FILSZ), SORT-CORE-SIZE (RESINV), SORT-MSG(MSGDDN)
SORT-MODE-SIZE (SMS=nnnnn)
SORT-RETURN(return-code of sort) and
SORT-CONTROL (Names the file of control card – default is IGZSRTCD)

STRING MANIPULATION

A string refers to a sequence of characters. String manipulation operations include finding a particular character/sub-string in a string, replacing particular character/sub-string in a string, concatenating strings and segmenting strings. All these functions are handled by three verbs INSPECT, STRING and UNSTRING in COBOL. EXAMINE is the obsolete version of INSPECT supported in COBOL74.

INSPECT- FOR COUNTING

It is used to tally the occurrence of a single character or groups of characters in a data field.

```
INSPECT identifier-1 TALLYING identifier-2 FOR
      ALL|LEADING literal-1|identifier-3
     [BEFORE|AFTER INITIAL identifier-4|literal-2]        - Optional.

INSPECT identifier-1 TALLYING identifier-2 FOR
      CHARACTERS
     [BEFORE|AFTER INITIAL identifier-4|literal-2]        - Optional.
```

Main String is identifier-1 and count is stored in identifier-2. Literal-1 or Identifier-3 is a character or group-of-characters you are looking in the main-string. INSPECT further qualifies the search with BEFORE and AFTER of the initial occurrence of identifier-4 or literal-2.

Example:
WS-NAME – 'MUTHU SARAVANA **S**URYA CHANDRA DEVI'

```
INSPECT WS-NAME TALLYING WS-COUNT ALL 'S'
  BEFORE INITIAL 'SARAVANA' AFTER INITIAL 'CHANDRA'
END-INSPECT
```
Result: WS-COUNT contains – 1

INSPECT- FOR REPLACING

It is used to replace the occurrence of a single character or groups of characters in a data field.

```
INSPECT identifier-1 REPLACING
      ALL|LEADING literal-1|identifier-2 BY identifier-3|literal-2
     [BEFORE|AFTER INITIAL identifier-4|literal-2]        - Optional.

INSPECT identifier-1 REPLCING CHARACTERS
      BY identifier-2 BEFORE|AFTER INITIAL identifier-3|literal-1
```

INSPECT-FOR COUNTING AND REPLACING

It is a combination of the above two methods.

INSPECT identifier-1 TALLYING <tallying part > REPLACING <replacing part>

STRING

STRING command is used to concatenate one or more strings.

Syntax:

```
STRING identifier-1 / literal-1, identifier-2/ literal-2
DELIMITED BY (identifier-3/literal-3/SIZE)
INTO identifier-4
END-STRING.
```

```
01 VAR1 PIC X(10) VALUE 'MUTHU    '
01 VAR2 PIC X(10) VALUE 'SARA     '
01 VAR2 PIC X(20).
```

To get display 'MUTHU,SARA'

```
STRING VAR1 DELIMITED BY ' '
       ',' DELIMITED BY SIZE
         VAR2 DELIMITED BY ' '
   INTO VAR3
END-STRING.
```

The receiving field must be an elementary data item with no editing symbols and JUST RIGHT clause.

With STRING statement, specific characters of a string can be replaced whereas MOVE replaces the full string.

```
01 AGE-OUT PIC X(12) VALUE '12 YEARS OLD'.
STRING '18' DELIMITED BY SIZE INTO AGE-OUT.  => 18 YEARS OLD.
```

Reference Modification – equivalent of SUBSTR

'Reference modification' is used to retrieve or overwrite a sub-string of a string. '**:**' is known as reference modification operator.

Syntax: String(Starting-Position**:**Length)

```
MOVE '18' TO AGE-OUT(1:2) does the same as what we did with STRING command.
```

When it is used in array elements, the syntax is

```
Array-element (occurrence) (Starting-Position:Length)
```

UNSTRING

UNSTRING command is used to split one string to many strings.

Syntax:

```
UNSTRING  identifier-1
[DELIMITED BY (ALL/) identifier2/literal1 [,OR (ALL/) (identifier-3/literal-2),..]]
INTO identifier-4 [,DELIMITER IN identifier-5, COUNT IN identifier-6]
 [,identifier-7 [,DELIMITER IN identifier-8, COUNT IN identifier-9]

 01 WS-DATA PIC X(12) VALUE '10/200/300/1'.
 UNSTRING WS-DATA DELIMITED BY '/'
        INTO WS-FLD1 DELIMITER IN WS-D1 COUNT IN WS-C1
             WS-FLD2 DELIMITER IN WS-D2 COUNT IN WS-C2
             WS-FLD3 DELIMITER IN WS-D3 COUNT IN WS-C3
 END-UNSTRING.
```

Result:

```
WS-FLD1 = 10 WS-FLD2 =200 WS-FLD3=300
WS-C1 = 2 WS-C2=3 WS-C3=3 WS-D1 = '/' WS-D2='/' WS-D3 '/'
```

ON OVERFLOW can be coded with STRING and UNSTRING. If there is STRING truncation then the imperative statements followed ON OVERFLOW will be executed.

COPY Statement

A COPY statement is used to bring a series of prewritten COBOL entries that have been stored in library, into a program.

1.Common routines like error routine, date validation routine are coded in a library and bring into the program by COPY.

2. Master files are used in multiple programs. Their layout can be placed in one copybook and be placed wherever the files are used. It promotes program standardization since all the programs share the same layout and the same data names.

This reduces coding and debugging time. Change in layout needs change in copybook only. It is enough if we just recompile the program for making the new copy effective.

Syntax:

COPY copybook-name [(OF/IN) library name]
[REPLACING string-to-be-replaced BY replacing-string]

Copybooks are stored as members in PDS library and during compilation time, they are included into the program. By default, the copybook library is SYSLIB and it can be changed using IN or OF of COPY statement.

Copybooks can be used in the following paragraphs.

SOURCE-COMPUTER, OBJECT-COMPUTER, SPECIAL-NAMES, FILE-CONTROL, IO-CONTROL, FD SECTION, PARAGRAPHS IN PROCEDURE DIVISION.

If the same copybook is used more than once in the program, then there will be "duplicate data declaration" error during compilation, as all the fields are declared twice. In this case, one copybook can be used with REPLACING verb to replace high-level qualifier of the all the variables with another qualifier.

Example: COPY CUSTOMER REPLACING 'CUST1-' BY 'CUST2-'.

Delimiter '= =' should be used for replacing pseudo texts. The replacing option does not alter the prewritten entries in the library; the changes are made to the user's source program only.

CALL statement (Sub-Programs)

When a specific functionality need to be performed in more than one program, it is best to write them separately and call them into each program. Sub Programs can be written in any programming language. They are typically written in a language best suited to the specific task required and thus provide greater flexibility.

Main Program Changes:

CALL statement is used for executing the sub-program from the main program. A sample of CALL statement is given below:

CALL 'PGM2' USING BY REFERENCE WS-VAR1, BY CONTENT WS-VAR2.

PGM2 is called here. WS-VAR1 and WS-VAR2 are working storage items. WS-VAR1 is passed by reference. WS-VAR2 is passed by Content. BY REFERENCE is default in COBOL and need not be coded. BY CONTENT LENGTH phrase permits the length of data item to be passed to a called program.

Sub-Program Changes:

WS-VAR1 and WS-VAR2 are working storage items of main program. As we have already mentioned, the linkage section is used for accessing external elements. As these working storage items are owned by main program, to access them in the sub-program, we need to define them in the linkage section.

```
LINKAGE SECTION.
01 LINKAGE SECTION.
    05 LK-VAR1 PIC 9(04).
    05 LK-VAR2 PIC 9(04).
```

In addition to define them in linkage section, the procedure division should be coded with these data items for address-ability.

```
PROCEDURE DIVISION USING LK-VAR1,LK-VAR2
```

There is a one-one correspondence between passed elements and received elements (Call using, linkage and procedure division using) BY POSITION. This implies that the name of the identifiers in the called and calling program need not be the same (WS-VAR1 & LK-VAR1) but the number of elements and picture clause should be same.

The last statement of your sub-program should be EXIT PROGRAM. This returns the control back to main program. GOBACK can also be coded instead of EXIT PROGRAM but not STOP RUN. EXIT PROGRAM should be the only statement in a paragraph in COBOL74 whereas it can be coded along with other statements in a paragraph in COBOL85.

PROGRAM-ID. <Program-name> IS INITIAL PROGRAM.

If IS INITIAL PROGRAM is coded along with program-id of sub program, then the program will be in initial stage every time it is called (COBOL85 feature). Alternatively CANCEL issued after CALL, will set the sub-program to initial state.

If the sub program is modified then it needs to be recompiled. The need for main program recompilation is decided by the compiler option used for the main program. If the DYNAM compiler is used, then there is no need to recompile the main program. The modified subroutine will be in effect during the run. NODYNAM is default that expects the main program recompilation.

Difference between Pass-by-reference and Pass-by-content

Sl #	Passl By Reference	Pass By Content
1	CALL ‘sub1’ USING BY REFERENCE WS-VAR1	CALL ‘sub1’ USING BY CONTENT WS-VAR1 (BY CONTENT keyword is needed)
2	It is default in COBOL. BY REFERENCE is not needed.	BY CONTENT key word is mandatory to pass an element by value.
3	Address of WS-VAR1 is passed	Value of WS-VAR1 is passed
4	The sub-program modifications on the passed elements are visible in the main program.	The sub-program modifications on the passed elements are local to that sub-program and not visible in the main program.

Difference between Static Call and Dynamic Call

Sl #	STATIC Call	DYNAMIC Call
1	Identified by Call literal. Ex: CALL 'PGM1'.	Identified by Call variable and the variable should be populated at run time. 01 WS-PGM PIC X(08). Move 'PGM1' to WS-PGM CALL WS-PGM
2	Default Compiler option is NODYNAM and so all the literal calls are considered as static calls.	If you want convert the literal calls into DYNAMIC, the program should be compiled with DYNAM option. By default, call variables and any un-resolved calls are considered as dynamic.
3.	If the subprogram undergoes change, sub program and main program need to be recompiled.	If the subprogram undergoes change, recompilation of subprogram is enough.
4	Sub modules are link edited with main module.	Sub modules are picked up during run time from the load library.
5	Size of load module will be large	Size of load module will be less.
6	Fast	Slow compared to Static call.
7	Less flexible.	More flexible.
8	Sub-program will not be in initial stage the next time it is called unless you explicitly use INITIAL or you do a CANCEL after each call.	Program will be in initial state every time it is called.

INTRINSIC FUNCTIONS:

LENGTH	Returns the length of the PIC clause. Used for finding length of group item that spanned across multiple levels.
MAX	Returns the content of the argument that contains the maximum value
MIN	Returns the content of the argument that contains the minimum value
NUMVAL	Returns the numeric value represented by an alphanumeric character string specified in the argument.
NUMVAL-C	Same as NUMVAL but currency and decimal points are ignored during conversion.
CURRENT DATE	Returns 21 Chars alphanumeric value – YYYYMMDDHHMMSSnnnnnn
INTEGER OF DATE	Returns INTEGER equivalent of Gregorian date passed.
INTEGER OF DAY	Returns INTEGER equivalent of Julian date passed.
DATE OF INTEGER	Returns Gregorian date for the integer passed.
DAY OF INTEGER	Returns Julian date for the integer passed.

Note: FUNCTION INTEGER OF DATE (01-01-1601) returns 1.

FILE HANDLING

A data file is collection of relevant records and a record is collection of relevant fields. The file handling in COBOL program involves five steps.

Steps in file-handing

1.Allocation: The files used in the program should be declared in FILE-CONTROL paragraph of environment division. The mapping with JCL DDNAME is done here. The file is allocated to your program by this statement.

2.Definition. The layout of the file and its attributes are defined in the FILE SECTION of DATA DIVISION.

3.Open: Dataset is connected/readied to your program using OPEN statement. The mode of OPEN decides the operation allowed and the initial pointer in the dataset. For example, EXTEND mode allows only write access and the pointer is kept on the end of file to append.

4.Process: Process the file as per requirement, using the I-O statements provided by COBOL. (READ, WRITE, REWRITE and DELETE)

5. Close: After the processing, close the file to disconnect it from the program.

Allocation of file - SELECT Statement
(ENVIRONMENT-> INPUT-OUTPUT-> FILE-CONTROL)

SELECT [OPTIONAL] *FILENAME* ASSIGN to *DDNAME*	=>ALL Files
ORGANIZATION IS SEQUENTIAL/INDEXED/RELATIVE	=>ALL Files
ACCESS IS SEQUNETIAL/RANDOM/DYNAMIC	=>ALL Files
RECORD KEY IS *FILE-KEY1*	=>KSDS
RELATIVE KEY IS *WS-RRN*	=>RRDS
ALTERNARE RECORD KEY IS *FILE-KEY2* WITH DUPLICATES	=>KSDS with
ALTERNARE RECORD KEY IS *FILE-KEY3* WITHOUT DUPLICATES	=>AIX
FILE STATUS IS *WS-FILE-STAT1*	=>ALL Files
[,*WS-FILE-STAT2*]	=>VSAM Files

SELECT Statement- OPTIONAL Clause

This can be coded only for input files. If OPTIONAL is not coded, then the input file is expected to present in JCL. If not, an execution error will occur.

If OPTIONAL is coded, then if the file is not mapped in JCL, it is considered as empty file and the first read results end of file.

The file can also be dynamically allocated instead of static allocation in JCL.

SELECT Statement- ASSIGN TO

FILENAME is the logical name used inside the program and *DDNAME* is the logical name in the JCL, mapped with physical dataset. *DDNAME* can be prefixed with ‘S-‘ to indicate QSAM file, ‘-AS’ to indicate ESDS file and with no prefix to indicate KSDS/RRDS file.

JCL Step executing the program should have a dataset with *DDNAME* as label
//*DDNAME* DD DSN=BPMAIN.EMPLOYEE.DATA,DISP=SHR

SELECT Statement-ORGANIZATION

It can be SEQUENTIAL (PS or VSAM ESDS), INDEXED (VSAM KSDS), RELATIVE (VSAM RRDS). Default is Sequential.

SELECT Statement-ACCESS MODE

SEQUENTIAL.

It is default access mode and it is used to access the records ONLY in sequential order. To read 100th record, first 99 records need to be read and skipped.

RANDOM.

Records can be randomly accessed in the program using the primary/alternate key of indexed file organization or relative record number of relative organization.100th record can directly be read after getting the address of the record from the INDEX part for INDEXED files.100th record can directly be read for RELATIVE files even without any index.

DYNAMIC.

It is mixed access mode where the file can be accessed in random as well as sequential mode in the program.

Example: Reading the details of all the employees between 1000-2000. First randomly access 1000th employee record, then read sequentially till 2000th employee record. START and READ NEXT commands are used for this purpose in the procedure division.

SELECT Statement-RECORD KEY IS

It is primary key of VSAM KSDS file. It should be unique and part of indexed record structure.

SELECT Statement-ALTERNATE RECORD KEY IS

This phrase is used for KSDS files defined with AIX. Add the clause WITH DUPLICATES if the AIX is defined with duplicates.

Referring to VSAM basics, every alternate index record has an associated PATH and the path should be allocated in the JCL that invokes this program.

The DDNAME of the path should be DDNAME of the base cluster suffixed with 1 for the first alternate record clause, suffixed with n for nth ALTERNATE RECORD KEY clause in SELECT clause.

SELECT Statement-FILE STATUS IS *WS-FILE-STAT1,WS-FILE-STAT2*

WS-FILE-STAT1 should be defined as PIC X(02) in working storage section. After every file operation, the file status should be checked for allowable values.

WS-FILE-STAT2 can be coded for VSAM files to get the VSAM return code (2 bytes), VSAM function-code (1 byte) and VSAM feedback code (3 bytes).
This is a 6- byte field in working storage.

RESERVE Clause.

RESERVE clause [RESERVE integer AREA] can be coded in the SELECT statement. The number of buffers to be allocated for the file is coded here.
By default two buffers will be allocated if the clause is not coded. Since similar option is available in JCL, this is not coded in program.

RESERVE 1 AREA allocates one buffer, for the file in the SELECT statement.

Defining the file in FILE SECTION - FD

FD *FILENAME*
RECORDING MODE IS V/VB/F/FB
RECORD CONTAINS M CHARACTERS (TO N CHARACTERS)
BLOCK CONTAINS X CHARACTERS/RECORDS (TO Y CHARACTERS/RECORDS)
LABEL RECORDS ARE OMITTED/STANDARD
DATA RECORD IS *FILE-RECORD*.
01 *FILE-RECORD* PIC X(nnn).

FD-RECORD CONTAINS

It specifies the length of the record in terms of bytes. (It will be RECORD contains m to n CHARACTERS for variable format files)

FD-BLOCK CONTAINS

It specifies the physical record size. It can be mentioned as number of logical records OR number of characters, that is multiple of logical record length. It is suggested to code BLOCK CONTAINS 0 RECORDS so that system will decide the optimum size for the file based on the device used for storing the file. BLOCK CONTAINS clause is treated as comments for VSAM files.

Advantage of Blocking:
1.I-O time is reduced as n numbers of records are read into main memory buffer during an I-O.
2.Inter record gap is removed and the gap exist only between blocks. So memory wastage due to IRG is avoided.

FD-RECORDING MODE IS

It can be F (FIXED) V(VARIABLE) FB(FIXED BLOCK) VB(VARIABLE BLOCKED)
Variable record file identification:

If there is no recording mode/record contains clause, it is still possible to identify variable length records. If there is an OCCURS depending on clause or there are multiple 01 levels and every 01 level is of different size, then the file would be of variable length. Multiple 01 level in File section is an example for implicit redefinition.

FD-LABEL RECORDS Clause

As a general rule, LABEL RECORDS are STANDARD is coded for Disk and Tape files, LABEL RECORDS ARE OMITTED is coded for printer files. In COBOL74, this clause is a mandatory clause whereas COBOL85 made this as optional.

FD-DATA RECORD IS Clause

It is used to name the data record(s) of the file. More than one record can be coded here.

OPEN STATEMENT
Syntax: OPEN OPENMODE *FILENAME*
OPENMODE can be INPUT OUTPUT I-O EXTEND
INPUT - File can be used ONLY-FOR-READ purpose.
OUTPUT - File can be used ONLY-FOR-WRITE purpose.
I-O - File can be used FOR READ, WRITE and REWRITE purpose.
EXTEND - File can be used FOR appending records using WRITE.

CLOSE statement.

The used files are closed using CLOSE statement. If you don’t close the files, the completion of the program closes all the files used in the program.

Syntax: CLOSE *FILENAME*

OPEN and CLOSE for TAPE files - Advanced

If more than one file is stored in a reel of tape, it is called as multi-file volume. When one file is stored in more than one reel of tape, it is called as multi-volume label. One reel is known as one volume. When the end of one volume is reached, automatically the next volume opens. So there is no special control is needed for multi volume files.

```
OPEN INPUT file-1 [WITH NO REWIND | REVERSED]
OPEN OUTPUT file-2 [WITH NO REWIND]
CLOSE file-3 [{REEL|UNIT} [WITH NO REWIND| FOR REMOVAL]
CLOSE file-3 [WITH NO REWIND|LOCK]
```

UNIT and REEL are synonyms.

After opening a TAPE file, the file is positioned at its beginning. When opening the file if the clause REVERSED is coded, then the file can be read in the REVERESE direction. (Provided hardware supports this feature)

When you close the file, the tape is normally rewound. The NO REWIND clause specifies that the TAPE should be left in its current position.

CLOSE statement with REEL option closes the current reel alone. So the next READ will get the first record of next REEL. This will be useful when you want skip all the records in the first reel after n number of records processing.

Since TAPE is sequential device, if you create multiple files in the same TAPE, then before opening the second file, first file should be closed. At any point of time, you can have only one file is active in the program. In addition to this, you have to code MULTIPLE FILE clause in the I-O control paragraph of environment division.

```
MULTIPLE FILE TAPE CONTAINS     OUT-FILE1 POSITION 1
                                OUT-FILE3 POSITION 3.
```

The files OUT-FILE1 and OUT-FILE3 used in the program are part of a same TAPE and they exist in first and third position in the tape. Alternatively, this information can be passed from JCL using LABEL parameter.

READ statement

READ statement is used to read the record from the file.

Syntax: READ *FILENAME* [INTO ws-record] [KEY IS *FILE-KEY1*]

[AT END/INVALID KEY imperative statement1]
[NOT AT END/NOT INVALID KEY imperative statement2]
END-READ

If INTO clause is coded, then the file is directly read into working storage section record. It is preferred as it avoids another move of file-section-record to working-storage-record followed by simple READ. READ-INTO is not preferred for variable size records where the length of the record being read is not known.

KEY IS clause is used while accessing a record randomly using primary/alternate record key.

AT END and NOT AT END are used during sequential READ of the file.

INVALID KEY and NOT INVALID KEY are used during random read of the file. Before accessing the file randomly, the key field should have a value before READ.

WRITE Statement

Write statement is used to write a new record in the file. If the file is opened in EXTEND mode, the record will be appended. If the file is opened in OUTPUT mode, the record will be added at the current position.

Syntax: WRITE *FILE-RECORD* [FROM ws-record]
[INVALID KEY imperative statement1]
END-WRITE

FROM clause avoids the explicit move of working storage record to file section record before WRITE.

REWRITE Statement

REWRITE is used to update an already read record. To update a record in a file, the file should be opened in I-O mode.

Syntax: REWRITE *FILE-RECORD* [FROM ws-record]
[INVALID KEY imperative statement1]
END-REWRITE

START Statement

START is used with dynamic access mode of indexed files. It establishes the current location in the cluster for READ NEXT statement. START itself does not retrieve any record.

Syntax: START *FILENAME*
KEY is EQUAL TO/NOT LESS THAN/GREATER THAN key-name
[INVALID KEY imperative statement1]
END-START.

DELETE Statement

DELETE is used to delete the most recently read record in the file. To delete a record, the file should be opened in I-O mode.

Syntax: DELETE *FILENAME* RECORD
[INVALID KEY imperative statement1]
END-DELETE

File Error – Handling

There are chances for failure of any file I-O processing. The failure of an I-O operation can be accepted or cannot be tolerated. The severity of failure has to be defined in the program design stage.

Let us assume that we don't have any error handling in our program. In this case, for example, if you don't have a specific record in the file, the random read of that record would immediately terminate the program with error 'record not found'.

Error Handling Clauses Provided by COBOL.

The sudden termination can be avoided by handling this error, with INVALID KEY clause of READ. Based on the importance of the record and business rule, we can continue our program with next record or terminate the program properly.
AT END is another error handling clause provided by COBOL. But there is no way to handle all such errors in this way.

Assign file-status and take the responsibility.

The second method is, assigning file-status to the file in the SELECT clause and checks the file status after each and every I-O and ensures that the value of status code is one of the allowable values. If it is not an allowable return code, then abnormally end the program with error statements that would be easier to debug.

But we have to do this checking after each and every I-O operation.
This is MOST PREFERRED ERROR HANDLING METHOD in structured programming.

Declaratives – USE statement

COBOL provides an option to group all the possible errors of specific operation(s) in a place and that will be automatically invoked during the respective operation(s) of any file. This avoids redundant code.

This is done in DECLARATIVE section of the procedure division. DECLARATIVE should be the first section in the procedure division if coded.

```
PROCEDURE DIVISION.
DECLARATIVES.
USE-PROCEDURE SECTION.
        USE AFTER EXCEPTION PROCEDURE ON INPUT
ERROR-PROCEDURE.
        Check the file-status code for validity.
END-DECLARATIVES.
```

Whenever there is an error in the processing of ANY FILE opened in INPUT mode, then the control comes to ERROR-PROCEDURE. The validity of error should be checked in this paragraph and allow or restrict the process down, based on severity of error code.

The complete syntax of USE statements is:

```
        USE AFTER STANDARD ERROR|EXCEPTION PROCEDURE ON
                                    INPUT|OUTPUT|I-O|EXTEND| file-1
```

If INPUT is coded, the following procedure will be executed for every operation involved in any file that is opened in INPUT mode. OUTPUT, I-O and EXTEND have the same meaning but the mode is different.

If file name (file-1) is coded in the USE statement, then all the input-output operation of that specific file will be checked.

ERROR and EXCEPTION are synonyms.

The Procedure written in a DECLARATIVE section should not refer to any non-declarative procedure written after the end procedure and vice-versa.

I-O-CONTROL - SAME AREA AND SAME RECORD AREA

RESERVE clause of SELECT statement specifies the number of buffers to be allocated for a file. SAME AREA allows more than one file to use the same buffer area. This will be very useful when the program must work with a limited memory

space. But the problem is only one file should be open at a time if SAME AREA is coded.

Syntax: SAME AREA FOR file-1 file-2 file-3.

If SAME RECORD AREA is coded, then the buffer is not shared but only the record area is shared. So more than one file can be in open state. We should be careful while filling in the record area of the output file. This may destroy the record read most recently.

Syntax: SAME RECORD AREA FOR file-1 file-2 file-3.

SAME SORT AREA allows more than one sort/merge work files to use the same area. The sort work files are automatically allocated when file is opened and de-allocated when file is closed. As the sort file is automatically opened and closed during a SORT and two sort files cannot be opened at a time, this clause may not be useful.

Syntax: SAME SORT|SORT-MERGE AREA for file-1 file-2.
File-1 or file-2 should be a SD file.

I-O CONTROL- RERUN Clause

RERUN ON rescue FOR EVERY integer RECORDS on file-1

This will cause checkpoint to be taken for every interger-1 records processing of file-1. If the program ABENDED before the complete processing of the file-1, then the program will restart from integer+1ST record instead of first record. The rescue file details should be mentioned outside the program and it varies from installation to installation.

ENTRY statement

ENTRY statement establishes an alternate ENTRY point in a COBOL called sub-program. When a CALL statement naming the alternate entry point is executed in a calling program, control is transferred to the next executable statement following the entry statement. Except when a CALL statement refers to an entry name, the ENTRY statements are ignored at run-time.

Matching Logic

If you have been given two files of similar type, say master and transaction file and you are requested to update the master file with transaction file information for existing records and prepare a report of new transactions and deleted transactions, then you should go for what is called Matching logic. This is also known as co-sequential processing.

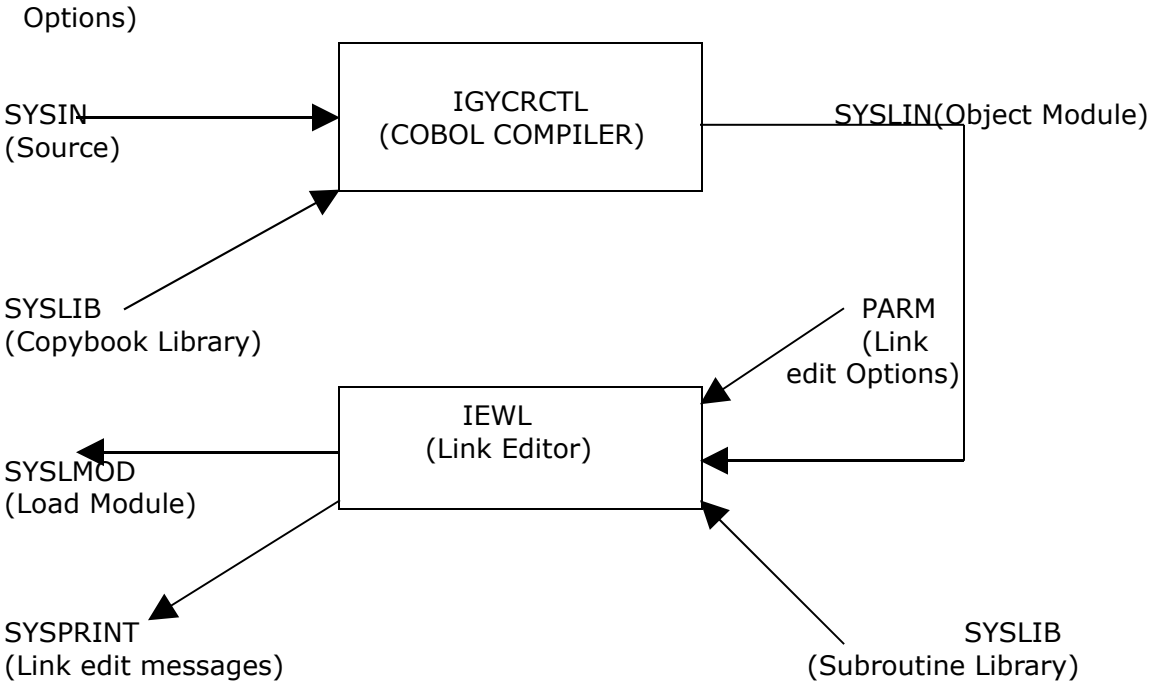
Sort both the files on key and compare the keys. If the keys are matching then update the file. If you find any record that is found in transaction but not in master file, then that is new addition and the reverse is deletion. If the master key is greater than transaction key, then that corresponds to the first case and reverse is the second case.

This can be easily done in JCL using ICETOOL. Refer JCL section.

FILE STATUS CODES

It is a two-byte working storage item. The first byte denotes the general category whereas second byte denotes the particular type of error message under that category.

0		*Successful OPEN/READ/WRITE Operation*
	0	Successful completion
	2	Duplicate key was detected which is allowed as per definition of AIX.
	4	Length of record just READ didn't conform to the fixed length attributes for the file.
	5	Referenced Optional file is not present during OPEN. If open mode is I-O or EXTEND, then file will be created.
	7	Open or Close statement is executed with a phrase that implies a tape file (ex NO REWIND) whereas the file is not in TAPE.
1		*When AT END condition fails*
	0	Sequential READ is attempted on 1.after the end of file is reached 2.optional file that is not present.
	4	Sequential READ was attempted for a relative file and RRN is larger than the maximum that can be stored in the relative key data item.
0		*When INDEX Key fails*
	1	Sequence error exists for sequentially accessed index file.
	2	Attempt was made to write a record that would create a duplicate key.
	3	Record not found.(for keyed random access)
	4	Space not found for WRITE
3		*Permanent Open error*
	5	Opening a non-optional file that was not present.
	7	Open mode is not permitted.
	8	Open issued for a file that was closed previously with lock
	9	File attribute mismatch-Open failed.
4		*Logic error in opening/closing/deleting*
	1	OPEN a opened file.
	2	CLOSE attempted for not opened file.
	3	IO statement before the current REWRITE/DELETE is not successful.
	4	REWRITE attempt with invalid length
	7	READ file which is not opened in INPUT or I-O mode
	8	WRITE file which is not opened in I-O OUPUT or EXTEND mode
	9	DELETE or REWRITE file which is not opened in I-O mode.
9		*Implementation defined*
	1	VSAM password error
	2	Logic error
	3	VSAM resource unavailable
	6	No DD statement specified for VSAM file.
	7	File integrity verified for VSAM file.



COMPILATION JCL:

```
//SMSXL86B JOB ,'COMPILATION JCL', MSGCLASS=Q,MSGLEVEL=(1,1),CLASS=C
//COMPILE1  EXEC  PGM=IGYCRCTL, PARM='XREF,APO,ADV,MAP,LIST),REGION=0M
//STEPLIB     DD  DSN=SYS1.COB2LIB,DISP=SHR
//SYSIN       DD  DSN=SMSXL86.TEST.COBOL(SAMPGM01),DISP=SHR
//SYSLIB      DD  DSN=SMSXL86.COPYLIB,DISP=SHR
//SYSPRINT    DD  SYSOUT=*
//SYSLIN      DD  DSN=&&LOADSET, DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200),
//             DISP=(NEW,PASS),UNIT=SYSDA,SPACE=(CYL,(5,10),RLSE),
//SYSUT1      DD UNIT=&SYSDA,SPACE=(CYL,(1,10))  => Code SYSUT2 to UT7
//LINKEDT1    EXEC  PGM=IEWL,COND=(4,LT)
//SYSLIN      DD  DSN=&&LOADSET, DISP=(OLD,DELETE)
//SYSLMOD     DD  DSN=&&GOSET(SAMPGM01),DISP=(NEW,PASS),UNIT=SYSDA
//             SPACE=(CYL,1,1,1))
//SYSLIB      DD  DSN=SMSXL86.LOADLIB,DISP=SHR
//SYSUT1      DD  UNIT=SYSDA,SPACE=(CYL,(1,10))
//SYSPRINT    DD SYSOUT=*

//*** EXECUTE THE PROGRAM ***
//EXECUTE1  EXEC  PGM=*.LINKEDT1.SYSLMOD,COND=(4,LT),REGION=0M
//STEPLIB     DD  DSN=SMSXL86.LOADLIB,DISP=SHR
//             DD  DSN=SYS1.SCEERUN,DISP=SHR
//SYSOUT      DD  SYSOUT=*
//SYSPRINT    DD  SYSOUT=*
```

Compiler Options

The default options that were set up when your compiler was installed are in effect for your program unless you override them with other options. To check the default compiler options of your installation, do a compile and check in the compilation listing.

Ways of overriding the default options

1.Compiler options can be passed to COBOL Compiler Program (IGYCRCTL) through the PARM in JCL.
2.PROCESS or CBL statement with compiler options, can be placed before the identification division.
3.If the organization uses any third party product or its own utility then these options can be coded in the pre-defined line of the utility panel.

Precedence of Compiler Options

1. (Highest precedence). Installation defaults, fixed by the installation.
2. Options coded on PROCESS /CBL statement
3. Options coded on JCL PARM parameters
4. (Lowest Precedence). Installation defaults, but not fixed.

The complete list of compiler option is in the table:

Aspect	Compiler Option
Source Language	APOST, CMPR2, CURRENCY, DBCS, LIB, NUMBER, QUOTE, SEQUENCE, WORD
Date Processing	DATEPROC, INTDATE, YEARWINDOW
Maps and Listing	LANGUAGE, LINECOUNT, LIST, MAP, OFFSET, SOURCE, SPACE, TERMINAL, VBREF, XREF
Object Deck generation	COMPILE, DECK, NAME, OBJECT, PGMNAME
Object Code Control	ADV, AWO, DLL, EXPORTALL, FASTSRT, OPTIMIZE, NUMPROC, OUTDD, TRUNC, ZWB
Debugging	DUMP, FLAG, FLAGMIG, FLAGSTD, SSRANGE, TYPECHK
Other	ADATA, ANALYZE, EXIT, IDLGEN

ADV: It is meaningful if your program has any printer files with WRITE..ADVANCING keyword. The compiler adds one byte prefix to the original LRECL of printer files for printing control purpose. If you are manually populating printing control character in the program, then you can compile your program with NOADV.

DYNAM: Use DYNAM to cause separately compiled programs invoked through the CALL *literal* statement to be loaded dynamically at run time. DYNAM causes dynamic loads (for CALL) and deletes (for CANCEL) of separately compiled programs at object time. Any CALL *identifier* statements that cannot be resolved in your program are also treated as dynamic calls. When you specify DYNAM, RESIDENT is also put into effect.

LIST/OFFSET: LIST and OFFSET are mutually exclusive. If you use both, LIST will be ignored. LIST is used to produce listing a listing of the assembler language expansion of your code. OFFSET is used to produce a condensed Procedure Division listing. With OFFSET, the procedure portion of the listing will contain line numbers, statement references, and the location of the first instruction generated for each

statement. These options are useful for solving system ABENDS. Refer JCL session for more details.

MAP: Use MAP to produce a listing of the items you defined in the Data Division.

SSRANGE: If the program is compiled with SSRANGE option, then any attempt to refer an area outside the region of the table will abnormally terminate with protection exception, usually S0C4.It also avoids any meaningless operation on reference modification like negative number in the starting position of reference modification expression. If the program is compiled with NOSSRANGE, then the program may proceed further with junk or irrelevant data. So usually the programs are compiled with SSRANGE during development and testing.

RENT: A program compiled as RENT is generated as a reentrant object module. CICS programs should be compiled with RENT option to share the same copy of the program by multiple transactions (Multithreading)

RESIDENT: Use the RESIDENT option to request the COBOL Library Management Feature. (The COBOL Library Management Feature causes most COBOL library routines to be located dynamically at run time, instead of being link-edited with the COBOL program.).CICS Programs should be compiled with RESIENT option.

XREF: Use XREF to get a sorted cross-reference listing. EBCDIC data-names and procedure-names will be listed in alphanumeric order. It also includes listing, where all the data-names that are referenced within your program and the line number where they are defined. This is useful for identifying the fields that are defined but not used anywhere after the development of new program.

TSO Commands from COBOL program

```
CBL APOST,NODECK,OBJECT,BUF(10000),DYNAM         => Compiler option override
     ***************************************************************
     *   FUNCTION = This sample program demonstrates how to invoke    *
     *              TSO commands from a COBOL program using           *
     *              standard TSO services as documented in the        *
```

```
*              TSO/E Programming Services manual.                *
*****************************************************************
 Identification Division.
 Program-ID. SMSTSOEV.

 Data Division.
  Working-Storage Section.
   01 Filler.
     05 ws-dummy        Pic s9(8) Comp.
     05 ws-return-code  Pic s9(8) Comp.
     05 ws-reason-code  Pic s9(8) Comp.
     05 ws-info-code    Pic s9(8) Comp.
     05 ws-cppl-address Pic s9(8) Comp.
     05 ws-flags        Pic X(4) Value X'00010001'.
     05 ws-buffer       Pic X(256).
     05 ws-length       Pic s9(8) Comp Value 256.

 Procedure Division.
*---------------------------------------------------------------*
*          Call IKJTSOEV to create the TSO/E environment         *
*---------------------------------------------------------------*
     CALL 'IKJTSOEV' Using ws-dummy,ws-return-code,ws-reason-code,
                           ws-info-code,ws-cppl-address.
     IF ws-return-code > zero
       DISPLAY 'IKJTSOEV Failed, Return-code=' ws-return-code
                               ' Reason-code=' ws-reason-code
                               'Info-code='    ws-info-code
       MOVE ws-return-code to Return-code
       STOP RUN.
*---------------------------------------------------------------*
*          Build the TSO/E command in ws-buffer                 *
*---------------------------------------------------------------*

     MOVE 'ALLOCATE DD(SYSPUNCH) SYSOUT HOLD' to ws-buffer.

*---------------------------------------------------------------*
*   Call the TSO/E Service Routine to execute the TSO/E command  *
*---------------------------------------------------------------*
     CALL 'IKJEFTSR' Using ws-flags,ws-buffer,ws-length
                           ws-return-code,ws-reason-code,ws-dummy.
     IF ws-return-code > zero
       DISPLAY 'IKJEFTSR Failed, Return-code=' ws-return-code
                               ' Reason-code=' ws-reason-code
       MOVE ws-return-code to Return-code
       STOP RUN.

*---------------------------------------------------------------*
*          Check that the ALLOCATE command worked               *
*---------------------------------------------------------------*
     DISPLAY 'ALLOCATE Worked ! ' Upon Syspunch.

     STOP RUN.
```

Interview Questions(IQ):

* Says importance and possibility of the question in an interview.

1.Difference between Next Sentence and Continue ***

2.Comp, Comp-1, Comp-2 and Comp-3 difference and how many bytes occupied by each. Should know how to read COMP-3 data. *****
3.Identifying and making Static and Dynamic call *****
4.Binary and Sequential search and when you prefer what? *****
5.What are the various ways of passing data from JCL to Program and how to receive them in Program? *****
6.Difference between COBOL74 (OS/VS COBOL) and COBOL85 (VS COBOL2) ****
7.Subscript and Index difference and when you prefer what? ****
8.Reference modification. ****
9.Compiler and Link edit option – SSRANGE MAP LIST OFFSET RENT RESIDENT DYNAM and AMODE/RMODE ***
10.How to make a call by content in COBOL? ***
11.How do you set return code from the program? ***
12.Case structure, Sub-string, Do while, Do Until, BLL equivalent in COBOL ***
13.Difference between section and paragraph *****
14.Can occurs be coded in 01 level? *****
15.Are multiple 01 levels supported in file section? **
16.Various ways of overriding default compiler options **
17.Sort algorithms **
18.How to get the actual length of alphanumeric item? **
19.What is UT-S means with respect to SELECT statement? *
20.Can I rewrite a sequential file in TAPE? *
21.COMP-3 items are always better than COMP in terms of memory. Yes/No **
22.Which VSAM type is fastest? Relative key is part of file section? **
23.How to create a report in COBOL program? ***
24.How to submit a JCL from COBOL program? ****
25.What is SYNC Clause? **
26.What is in line PERFORM and when will you use it? ****
27.What is INSPECT statement? ***
28.To use SEARCH ALL, the table should be in sorted order. I am loading the table from one of the PDS members. The PDS member data is not in sorted order. How will I load the table in SORTED order? You should not sort in JCL. **
29.What is the purpose of USE statement? *
30.What are SAME AREA and SAME RECORD AREA? *
31. Is dynamic allocation possible in COBOL? If yes, How? *
32. What is the difference between ON SIZE ERROR and ON OVERFLOW? *
33.How to swap two variables without third variable? *
34.What is limit of linkage section? *

Answers for selected questions:

What is the limit of working storage and linkage section limit? (IQ 34)

Working storage and Linkage section limit of COBOL85 is 128MB (COBOL74-1MB)

77,01-49 level item limit in COBOL85 is 16MB (COBOL74-1MB)

How to swap the values of two variables without an intermediate variable?(IQ 33)

Let the variables be A and B

```
Way 1:COMPUTE A = A+B        Way 2:    COMPUTE A=A*B
              COMPUTE B = A-B                    COMPUTE B=A/B
              COMPUTE A = A-B                    COMPUTE A=A/B
```

I have retrieved a value from DB2 VARCHAR column. (Ex: WS-VAR = ‘muthu$sara$‘ $ is 1-n spaces.) How to get the length of the WS-VAR in COBOL program? I should not count right hand spaces. (IQ 20)

LENGTH function counts space also as a character. So we cannot use that function for our purpose. INSPECT is also not useful as the string may contain 1- n spaces in between and that needs to be counted. So the logic would be ” Read from right until you read first noon-space character”.

```
PERFORM VARYING WS-SUB-NAME FROM
        LENGTH OF WS-VAR BY -1
        UNTIL END-FOUND OR WS-SUB-NAME = 0
  IF WS-NAME-CHK(WS-SUB-NAME:1) NOT EQUAL TO SPACE
     MOVE 'Y' TO WS-END-OF-FIELD
     DISPLAY 'LENGTH ' WS-SUB-NAME
  END-IF
END-PERFORM
```

How to pass user return code and user ABEND from the COBOL program to the JCL?

RETURN-CODE is a special register and its content is moved to register15 when the control is given back to OS. So move the return code to this register in the program.

Ex: MOVE 1000 to RETURN-CODE.

This sets return code as 1000 for the step that executes this program.

For ABEND, you should call your installation specific assembler routine or ILBOABN0 with the ABEND code you want.

CALL ‘ILBOABN0’ USING WS-AB-CODE.

WS-ABEND-CODE is the variable that has ABEND-CODE. It is a half word binary.

What should be the LRECL of printer files?

Use 133 character records in your program and set the print control character yourself. In this case your JCL would have RECFM=FB, LRECL=133

Use 132 character records in the program and have WRITE ....ADVANCING put in the print control. You need the compiler option ADV for this and the JCL would have RECFM=FBA,LRECL=133.....

What are the sort algorithms? (IQ 17 and 28)

Bubble Sort: Consecutive elements are compared and keys of two elements are not in proper order, they are swapped. In the first pass, the key with largest value will be moved to the last position and n-1 passes needed to sort the whole table.
In between, if any pass results no interchange it implies that the table is in sorted order.

Array: 1 20 9 50 8
First Pass: (Maximum 4 comparisons for 5 elements)
1, 20->no change, 20 & 9 -> 20 is great so swap (1 9 20 50),
20 & 50 -> no change, 50 & 8 -> 50 is great, so swap. (1 9 20 8 50)
Second Pass: (1 9 20 8 50) - (Maximum 3 comparison for 5 elements)
1 & 9-> no change, 9 & 20 -> no change, 20 & 8 -> 20 is great so swap
(1 9 8 20 50)
Third Pass: (1 9 8 20 50) – (Maximum 2 comparisons for 5 elements)
1 & 9 -> no change, 9 & 8-> change (1 8 9 20 50)
Fourth Pass: (1 8 9 20 50) – (Maximum 1 comparison for 5 elements)
1 & 9 -> no change
Note: You can come out of sort when you find 'no change' in all the comparisons of a pass.

Shuttle Sort: In the first pass only first two elements are compared and sorted and in the second pass, third element is compared with two and one and it is placed in the right position. In the ith pass, it assumes that I elements are in already sorted order, proceeds to sort the first (I+1) elements by comparing I+1 th element with I, and I with I-1 and so on until top of the table is reached or no-exchange in a comparison.

Array: 1 20 9 50 8
First Pass: Two elements (1 20) - Maximum 1 comparison
1, 20->no change
Second Pass: Three elements (1 20 9) – Maximum 2 comparisons
9 & 20 -> change (1 9 20) 9 & 1 -> no change
Third Pass: Four elements (1 9 20 50) – Maximum 3 comparisons
50 & 20 -> no change and stop (no need for any other comparison)
Fourth Pass: Five elements (1 9 20 50 8) – Maximum 4 comparisons
8 & 50 -> change (1 9 20 8 50) , 8 & 20-> Change (1 9 8 20 50)
8 & 9 -> Change (1 8 9 20 50) , 8 & 1 -> no change and stop.
Note: You can come out of pass if you find one 'no change'

Shuttle sort is better than bubble sort for sorting arrays with more than 10 elements.

<u>COMP-3 items are always better than COMP with respect to memory usage (IQ 21)?</u>
No. COMP items occupy less space than COMP-3 items at boundaries.
PIC S9(04) COMP occupies 2 bytes whereas PIC S9(04) COMP-3 occupies 3 bytes.
PIC S9(09) COMP occupies 4 bytes whereas PIC S9(09) COMP-3 occupies 5 bytes.
PIC S9(18) COMP occupies 8 bytes whereas PIC S9(18) COMP-3 occupies 10 bytes.

<u>I have a KSDS Students file with 4 bytes key. First two-bytes contain class number and next two-bytes contain student number. I want to read all the students in class '02'. How will you do that?</u>

Allocate the file with dynamic access mode. Move ‘02’ to first two-bytes of the key and low-values to next two-bytes of the key. You can do these moves by reference modification operator or de-grouping the four-byte field into two two-byte fields in the file section.

Issue the START command with KEY IS GREATER THAN clause. Start reading the file in loop until first two-bytes is not equal to 2.

JCL (**J**ob **C**ontrol **L**anguage)

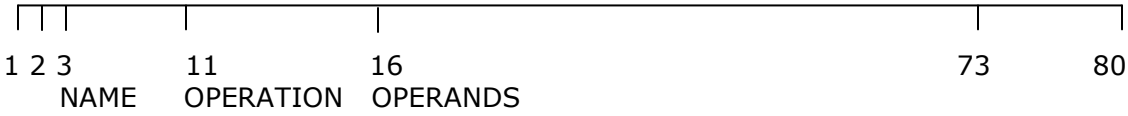
JCL

Any business application is divided into logical modules and these modules are developed using programming languages. These programs should be executed in a pre-defined sequence to achieve the business functionality.

JCL (Job Control language) is used to DEFINE and CONTROL the JOB to the operating system.

Definition involves the program need to be executed, the data for the program and the sequence of programs. CONTROL involves execution or bypassing of a program in the sequence based on the result of the prior program execution.

JCL Coding Sheet



JCL statement should have // in column 1 and 2. STAR ('*') in the third column, indicates that the line is a comment.

NAME is optional field. If coded, it should start at column 3 and can have maximum 8 characters. Alphanumeric and national characters are allowed in the NAME. The first character must not be a number.

OPERATIONAL follows NAME field. There should be at least one space between NAME and OPERATION. IF NAME is not coded then OPERATION can start at fourth column itself. OPERAION field can be JOB, EXEC and PEND.

OPERANDS are the parameter for the operation. OPERANDS follow OPERATION and there should be at least one space between them. A comma separates parameters and there should not any space between parameters. If the OPERANDS are more, then they can be continued in next line. To continue the current line, end the current line before column 72 with ',' and start the next line anywhere between column 4-16.

COMMENT FIELD – Comment field optionally follows OPERAND FIELD, preceded by at least one blank.

End of Job is identified by NULL statement. NULL statement has // in column 1 and 2 with no NAME, OPERATION or OPERAND fields. The statements coded after NULL statement will not be processed.

DELIMITER – Some times we pass the data in the JCL itself. This is called in-stream data. The starting of data is identified by '*' in the operand field of DD operation. DELIMITER indicates the end of data. /* in column 1 and 2 is the default delimiter.

Job Entry Subsystem (JES)

Job Entry Subsystem (JES) is the job processor of MVS operating system.
MVS installation can have either JES2 or JES3. The submitted jobs are taken by JES for processing.

JES2 is a decentralized environment. Every processor processes the incoming jobs individually. JES3 is centralized environment. There is a global processor, which controls all the other processors and assigns the jobs to them.

The datasets are pre-step allocated in JES2. That is, the datasets are allocated just before the execution of that step. The datasets are pre-job allocated in JES3.
That is, all the datasets referred in the Job are allocated before the JOB starts its execution. Most of the installation uses JES2.

JCL Statements

JOB. It should be the first statement in the JCL. It indicates accounting information and JOB related information to the system.

EXEC. The name of the program or procedure to be executed is coded here. Every EXEC statement in a JOB identifies one step. Maximum of 255 EXEC statements can be coded in a JOB.

DD. Data Descriptor. The dataset details are coded here. Dataset contain the data that need to be processed by the program or data that is produced by the program.

Abnormal End (ABEND) & ERROR

Once the work to be done is written in JCL, it can be submitted to the operating system using SUBMIT command.

JCL ERROR:

Submitted work (JOB) can be rejected before the execution and this is called JCLERROR. It is generally due to the syntax error in the JCL coding. The error message will appear in JES MESSAGES. Correct the error and submit the job once again.

If there are no syntax errors, then the job start running step-by-step.
In JES2 system, all the resources needed for a particular step is allocated just before the execution of that step. There are some errors that will be known only just before the execution of the step. One example is duplicate dataset name. You are trying cataloging a dataset, which is already available in the catalog.

ABEND:

ERROR occurs before the execution of JOB or STEP. But ABEND happens during the execution of a program in a step. There may some invalid operation, the operation that cannot be executed by the system in the program. When OS encounters such an operation, the respective JOB is terminated with the ABEND code.
Example: An arithmetic operation on computational field that have non numeric data in COBOL.

JOB Statement

Sample Syntax:

```
//JOBNAME JOB (ACCOUNTING INFO),(PROGRAMMER NAME),
//  TIME=(MINUTES,SECONDS),
//  CLASS=A,MSGCLASS=A,PRTY=14,ADDR=VIRT,
//  REGION=nK,MSGLEVEL=(A,B),COND=(N,OPERATOR),
//  TYPRUN=SCAN
```

JOBNAME

It identifies name of the job. The job is identified in the JES SPOOL using this name. Naming rules are already mentioned in the coding sheet section.

ACCOUNTING INFO (Mandatory. Installation Department.)

1. Resource usages charges are charged to the account mentioned here.
2. If you don't know your installation account, you cannot submit the job. It is like when you don't have account, you cannot withdraw cash in bank.
3. Maximum 142 characters can be coded as accounting information.

PROGRAMMER NAME

Programmer name or program functionality or group can be mentioned. It is used for documentation (Max 20 Chars)

CLASS (Installation Defined)

1. CLASS is coded with single alphanumeric character (A-Z, 0-9).
2. During installation, every CLASS and its properties are defined.
3. Definition describes the job characteristics like CPU time usage, number of tape/cart usage and other resource constraints.
4. Every class is assigned to one or more initiator. The jobs are run in initiator address space. One initiator can process one job at one time.

PRTY

Syntax: PRTY=N (N can be 0 – 15)

1. While selecting the jobs with same class for execution, JES schedules the high priority jobs first. The job coded with PRTY=15 has the highest priority and PRTY=0 has the lowest priority.
2. PRTY works within the JOBCLASS. If there are 2 jobs with CLASS A is submitted and one with PRTY 3 and other with PRTY 4 then PRTY 4 will get into execution queue first.
3. PRTY function is disabled in most of the installations.

MSGLEVEL

Syntax: MSGLEVEL=(X,Y) (X can be 0-2 & Y can be 0-1)

1. It is used to control the lists of information appear in the Job log. To get maximum information in the listing, code MSGLEVEL as MSGLEVEL(1,1)
2. The first parameter controls the statements. (0-Only job statement, 1-JCL, JES statement with expanded procedure, 2-Only JCL and JES statement).
3. The second parameter controls the messages. (0- Only Step execution messages, 1- All JCL, JES, operator and allocation messages).

MSGCLASS (Installation Defined)
Syntax: MSGCLASS=X (X can be A-Z, 0-9)
1. MSGCLASS is coded with single alphanumeric character (A-Z, 0-9).
2. Each MSGCLASS is mapped to a device or location, where the messages are routed.

ADDRSPC
It is used to specify whether the job will run in the Real storage or Virtual storage.
Syntax: ADDRSPC={REAL|VIRT}
REAL – Allocation is done in REAL storage and program is not page-able.
VIRT – Allocation is done in VIRTUAL storage and the program is page-able.

REGION
Syntax: REGION={xK | yM} (x can be 1-2096128 & y can be 1-2047).
1. It is used to specify the amount of central/virtual storage the job requires. It can be requested in the units of kilobytes (xK) or megabytes (yM). If requested in terms of kilobytes, then x should be multiple of 4 or the system will round to nearest 4K allocates for your job.
2. REGION can be coded in EXEC statement also. REGION parameter coded on JOB card overrides the parameter coded on EXEC card.
3. Maximum virtual memory available is 2GB.
4. Region=0M allocates all the available memory in the address space to this job.
5. Region related ABENDS: When the requested region is not available, the JOB will ABEND with S822. When the requested region is not enough for the program to run, you will get ABEND S80A or S804.

RESTART
When a job ABEND at STEP10 and if there are no dependencies, then the job can be restarted from STEP10 instead of once again from STEP1.
RESTART parameter of JOB card provides option for this.
RESTART=STEP10 bypass STEP01 to STEP09 and start execution from STEP10.
If your job has procedures and one of the steps in the procedure is ABENDED, then your restart parameter should be,
RESTART=PROCSTEP.STEPNAME =>
Whereas PROCSTEP=name of the JCL step that invoked the PROC &
STEPNAME=name of the proc step where you want execution to start.

Some installation use RESTART=* in their job card, that means start the execution from step 1. In case real restart, we can just replace * with STEPNAME.

TYPRUN
It is used to request special job processing.
1. TYPRUN=SCAN checks the syntax errors without actual execution.
2. TYPRUN=HOLD checks syntax error and if there is any error, it is notified and if there are no errors, the job is kept in awaiting execution queue and it should be released by user for execution. Release can be done by typing 'A' against the job name in SDSF.
3. TYPRUN=JCLHOLD Function is same as HOLD but the syntax check starts only after the release of the job.

TIME

It defines the maximum allowable CPU time for the JOB. The parameter can be coded at EXEC card also. On EXEC, it defines CPU limit of step.

Syntax: TIME = (MINUTES, SECONDS), MINUTES < = 1440 and SECONDS < 60
TIME-NOLIMIT/1440/MAXIMUM means the job can use unlimited time
TIME=0 will produce unpredictable results.

The EXEC time limit or the JOB time remaining after execution of previous steps, WHICHEVER IS SMALLER will be the time allowed for the job step.

```
Ex: //FIRST JOB (ACCT),TIME=3
    //STEP1 EXEC PGM=PGMONE,TIME=2
    //…
    //STEP2 EXEC PGM=PGMTWO,TIME=2
```

In this job, though we have coded PGMTWO can use the processor for 2 minutes, if PGMONE already used 2 minutes of CPU time, then only one minute is allowed for PGMTWO (as the complete JOB is allowed to run for only three minutes).

If a JOB runs more than allowed time, then it will ABEND with system ABEND code S322. If there is no TIME parameter, then the CPU time limit ore-defined with CLASS parameter will be effective.

NOTIFY

TSO User-id to whom the job END / ABEND/ ERROR status should be notified. NOTIFY=&USERID will send the notification to the user who submitted the job.

COND

1. It is used for conditional execution of JOB based on return code of JOB steps.
2. The return code of every step is checked against the condition coded on JOB card. If the condition is found TRUE, then all the steps following it are bypassed.
3. Maximum eight conditions can be coded in the COND parameter. In case of multiple conditions, if any of the condition id found TRUE then the JOB stops proceeding further.

Syntax: COND=(CODE,OPERATOR,STEPNAME)

STEPNAME is optional. If you code it, then that particular step-name return code is checked against the CODE with the OPERATOR. If omitted, then the return codes of all the steps are checked. On comparison, if the condition found to be true, then all the following steps are bypassed.

CODE can be 0-4095
OPERATOR can be GT, LT, GE, LE, EQ.

It can be coded on EXEC statement. STEP Level control is popular then JOB level control. On EXEC statement, you may find ONLY, EVEN keywords against COND parameter.

COND=ONLY allows the step execution only if any prior step ABENDED.
COND=EVEN allows the step execution independent of prior ABENDS.

Consider the COND parameter coded on EXEC statement,

```
Ex:  //STEP2 EXEC=PGM3,COND-((16,GE),(90,LE,STEP1),ONLY)
```

Step gets executed only if

A preceding step abnormally terminated or
The return codes from all preceding steps are 17 or greater
The return code from STEP1 is 89 or less.

EXEC Statement

It defines the Step and Step level information to the system.

Syntax: //STEPNAME EXEC {PGM=program-name |
PROC=proc-name |
Proc=name}

STEPNAME

It is an OPTIONAL field but it is needed if you want to restart a job from this step and for the same reason, it should be unique within the job.

PGM or PROC

Code the program name or PROC name to be executed. (1-8 characters)

PARM

1. It is used to pass variable information to the processing program, executed by this job step.
2. If there is more than one sub parameter of if there is any special character then enclose them in parentheses/quotes.
3. Maximum 100 characters can be passed in this way. Quotes and brackets are not included in this 100.
4. The program can receive them using linkage section. Linkage section must be coded with half word binary field as first field. This field is populated with length of the PARM passed from the JCL.

Example:

```
//STEP3 EXEC PGM=WORK,PARM=(DECK,LIST,'LINECNT=80',
//     '12+80',NOMAP)
```

DPRTY

PRTY assigns priority to a job and DPRTY assigns dispatching priority to job step. Syntax: DPRTY=(value1,value2). Value1 and value2 can be 0-15. D-Priority is calculated using the formula (value1*16 + value)

IF /THEN/ELSE/END-IF

It is used for conditionally executing one or more steps. Nesting is possible up to 15 levels. The meaning is same as programming IF. If the coded condition is true, the following steps till ELSE will be executed. If the condition is false, then the steps coded on ELSE part will be executed.

Syntax:

```
//name IF (relational operations) THEN
//…Steps...
//        ELSE
//…Steps..
//        END-IF.
```

PROC PEND INCLUDE '/*' '//' '//*' are executed irrespective of their place.

Don't specify JOBLIB, JCLLIB, JOBCAT, STEPCAT, JOB, SYSCHK within the THEN or ELSE Scope of IF statement.

Relational condition can be also coded as:

STEPNAME.ABEND=TRUE, STEPNAME.RUN=TRUE, STEPNAME NOT RUN
STEPNAME.ABENDCC = any-abend=code or

DD Statement

It defines the data requirements of the program. It is coded for every file used in the program. If the employee details are stored in a file and catalogued with the name 'SMSXL86.EMPLOYEE.DETAILS', one of he program (EMPPGM) in the application reads this file, then JCL card for the DD looks like.

```
//STEP EXEC PGM=EMPPGM,REGION=1M
//EMPFILE DD DSN=SMSXL86.EMPLOYEE.DETAILS,DISP=SHR
```

When the program opens EMPFILE, it would open SMSXL86.EMPLOYEE.DETAILS. The mapping is done as follows in the program. So in future, if you process the same program with another file SMSXL86.EMPLOYEE.DETAILS2, then change of dataset name in the JCL is enough. Because of logical mapping, program need not be changed.

```
SELECT file ASSIGN TO EMPFILE.  (COBOL Program)
DECLARE (EMPFILE) FILE;         (PL/I Program)
DCB DDNAME=EMPFILE              (ASSEMBLER)
FOPEN(EMPFILE,mode)             (C)
```

Maximum 3273 DD statements can be coded in a single EXEC step.

DSNAME(DSN)

The name of the dataset is coded in the DSN parameter. Dataset name can contain 444 characters including the periods in between qualifier. Each qualifier can have 8 characters and there can be 22 qualifiers. But usually we don't code more than 4 qualifiers.

DSN=XXXX.YYYY.ZZZZ,DISP=SHR

Temporary datasets are indicated by && in the DSN (DSN=&&temp).

If DSN is not specified, then the system assigns the specific name to dataset.

If DSN=NULLFILE or DUMMY is coded, then all the I/O s against this file are bypassed.

DISP

It is used to describe the status of a dataset to the system and instruct the system what to do with the dataset after successful/unsuccessful termination of the step or job.

DISP=(current-status, normal-termination-status, abnormal-termination-status)

Current-status can be:

NEW	Dataset does not exist. It will be created in this step
SHR	Dataset already exist and this step need it without any exclusive access
OLD	Dataset already exist and this step need it with exclusive access
MOD	If the dataset does not exist, it is to be created. If it already exists, records are to be added to the end of the dataset. In both the cases, exclusive is needed.

Normal-termination-status:

CATLG	System is to place an entry, pointing to the dataset in the system/user catalog
DELETE	Dataset is no longer required. Space available for use by another dataset but existing dataset not physically erased until overwritten by another dataset.
PASS	Dataset is passed to subsequent steps in the same job and each step can use the dataset only once.
KEEP	Dataset is to be kept on the volume
UNCATLG	System is to delete the catalog entry corresponding to this dataset and keep this dataset.

Abnormal-termination-status:
PASS is not allowed. Meaning of CATLG, UNCATLG, KEEP, DELETE are same as normal-termination status.

Absence of any parameter should be mentioned with ',' as they are positional parameter. Ex: DISP=(,,CATLG). If the dataset is a new dataset and DISP is not coded, then the default in effect would be *DISP=(NEW,DELETE,DELETE).*

DCB (Data Control block)
DCB specifies attributes of the record in the dataset.
Syntax DCB=(LRECL=NN,BLKSIZE=YY,RECFM=Z,DSORG=MM,BUFNO=nn)

RECFM
It specifies format of the dataset. It can be Fixed, Variable, Undefined, Fixed blocked, variable Blocked. (F, V, U, FB, VB). Other special record formats are VBS, FBS, VT, FT, FBA.

LRECL
It specifies logical record length. The length of the record is as in the program for fixed length records, length of the longest record with four more bytes for variable length record.

BLKSIZE
It contains physical record length. That is the length of the record in the storage medium. One block contains one or more logical records. It is suggested to code BLKSIZE as 0 so that the best size is chosen by the system, based device.
If you explicitly code it, then it should multiple of LRECL for FB datasets and should not be less than length of the longest record with eight more bytes for VB dataset. In the extra eight bytes, four bytes are used for length of the record and four bytes are used for length of the bloc.

DSORG.
PS (Physical Sequential) PO (Partitioned organization)

BUFNO.
The number of buffers to be allocated for the dataset is coded with BUFNO parameter. Maximum of 255 buffers can be coded. The performance of sequential processing will increase if more buffers are allocated. The default buffers are enough for most of the cases.

Source of DCB:

We don't always have to write the DCB parameter for a dataset. Writing DCB parameter is one of the three ways, the information can be supplied. The other two ways are:

1. Coded in the program. In COBOL, RECORD CONTAINS clause specifies the LRECL, BLOCK CONTAINS clause specifies the BLKSIZE, RECORDING MODE clause specifies RECFM and RESERVE clause specifies BUFNO. DSORG can be assumed from the name of the dataset and the directory space allocation of SPACE parameter.

2. Usually for an existing dataset, we don't have to code DCB parameters. It will be available in the dataset label. The dataset label is created in the VTOC (DASD) or along with dataset (TAPE) during the dataset creation.

LABEL

Syntax: LABEL = (Dataset-sequence-number
,label-type
,PASSWORD | NOPWREAD
,IN | OUT
,RETPD=nnn |EXPDT = (yyyddd|yyyy|ddd))

Dataset Sequence number – identifies the relative position of a dataset on a tape/cart volume. Should be 1 through 4 decimal digits. Omit this parameter if access is being made to the first dataset on the tape volume.

Label – indicates the label type of the tape/cart volume.
SL - indicates that a dataset has IBM standard labels. Default value.
NL - indicates that a tape dataset has no labels.
NSL - indicates that a tape dataset has nonstandard labels.
SUP - indicates that a tape dataset both IBM standard and user labels.
BLP - request that the system bypass label processing for a tape dataset.

PASSWORD - indicates that a dataset cannot be read, changed, deleted or written to unless the correct password is specified.
NOPWREAD - indicates that a dataset cannot be changed, deleted or written, unless the correct password is specified. No password is necessary for reading the dataset.

IN - indicates that a dataset opened for I/O is to be read only.
OUT - indicates that a dataset opened for I/O is to be written only.

RETPD / EXPDT - indicates the retention period and the expiration date for a dataset.

Ex: LABEL=EXPDT=04121 (Dataset expires in 121st day of 2004)
LABEL=RETPD=200 (Dataset is retained for 200 days)

SPACE

It is used to request space for the new dataset. It is mandatory for all the NEW datasets.

SPACE – ({TRK \ CYL | blklgth} (,Primary-qty , Second-qty, Directory)
[,RLSE] [,CONTIG] [,MXIG] [,ROUND])

TRK/CYL - Requests that space be allocated in tracks or cylinders.
Blklgth - Specifies the average block length, in bytes, of data, Specify a decimal number from1 through 65535. This takes precedence, when specified, together with the BLKSIZE field of DCB parameter.

Primary-qty - Specifies the amount of primary space required in terms of the space unit (tracks/cylinders/number of data blocks). One volume must have enough space for the primary quantity. If a particular volume is requested must have enough space for the primary quantity. If a particular volume is requested and it does not have enough space available for the request, the job step is terminated.
Second-qty - Specifies the number of additional tracks, cylinders, blocks to be allocated, if additional space is required.
Directory - Specifies the number of 256-byte records needed in the directory of a PDS. (In every block we can store 5-6 members)

RLSE - requests that space allocated to an output dataset, but not used, is to be released when the dataset is closed. Release occurs only if dataset is open for output and the last operation was a write.

CONTIG - requests that space allocated to the dataset must be contiguous. It affects only primary space allocation.

MIXIG - It is used to specify that space requested should be allocated to the largest contiguous area of space available on the volume. It affects only primary allocation.

ROUND - When the first parameter specifies the average block length, this parameter that allocated space must be equal to an integral number of cylinders. Else ignored.

Extents

Extent is contiguous memory location. Only 16 extents are possible for a physical sequential dataset in a volume. In loose terms, only 16 pointers can be stored for a PS dataset in one volume. For a VSAM dataset it can be 123. In addition to this, the primary (first) or secondary (consecutive) space request has to be met within 5 extents.

If any of the above is not met, then there will be space ABEND. So far a PS dataset, even though you request 1600tracks (using the space parameter (SPACE=TRKS,(100,100)), the system may not allocate you 1600 tracks always.

If all the contiguous available spaces are of size 20 tracks, then 5 extends are used for satisfying every primary or secondary. So 15 extents are used for providing just 300 tracks. If the system could find any 100 tracks for the 16^{th} extent, it would offer it and if not, there will be space ABEND. So in the best case you will 1600 tracks and in the worst case you will get 400 tracks for the space parameter mentioned.

Space ABENDS:

If you go through the JOB failure history of any system, you will find the frequency of space ABENDS are more than any other ABEND. The various space ABENDS are listed below.

SB37: End of volume. We have requested 1600 tracks (including primary and secondary). When the program tries to write more than 1600 tracks, the operation ended with SB37. It should be noted that though I have requested 1600 tracks, I may get this ABEND even after just 400 tracks. Because 1600 tracks is the best-case allocation and 400 tracks is the worst-case allocation.

Solution:

1.Increase the size of primary and secondary reasonably. If you increase the size blindly, then also you will get SB37 if the size of secondary or primary is not available within five extents.

2.If the job again and again comes down, then simple solution is make the dataset multi volume by coding VOL=(,,,3). Now the dataset will get space of 48 extents and it would be more than enough for successful run.

SD37: Secondary space is not given. We have coded only the primary space and that is already filled and then if the program tries to write more, you will get SD37 space ABEND.

Solution:

Include the secondary space in the SPACE parameter of the dataset that has given problem.

SE37: End of Volume. This is same as SB37. You will get this ABEND usually for a partitioned dataset.

Solution:

If the partitioned dataset is already existing one, then compress the dataset using 'Z' in ISPF/IEBGENER and then submit the job once again. If it again ABENDED, then create a new dataset with more primary, secondary and directory space and copy all the member of current PDS to the new PDS, delete the old, rename the new to old and resubmit the JOB.

<u>UNIT</u>

It is used to request the system to place the dataset on a specific device/a certain type of group of devices or the same device as another dataset.

```
UNIT=((device-number | device-type | group-name)
      (,unit-count \ P)
      (,DEFER))
        OR
UNIT=AFF=ddname
```

device-number - identifies a particular device by a 3-character hexadecimal number. It should not be used unless absolutely necessary.

device type - Requests a device by its IBM supplied generic name. (Eg. 3380)

group-name - Requests a group of devices by its symbolic name. Installation must have assigned the name to the device(s) during system initialization.
The group-name is 1 through 8 alphanumeric characters (Eg. TEMPDA)

unit-count - Specifies the number of devices for the dataset. (1-59)

P - asks the system to mount all volumes for a dataset in parallel.

DEFER - Asks the system to assign the dataset to the device but requests that the volume(s) not be mounted until the dataset is opened. DEFER is ignored for a new dataset on direct access.

AFF=ddname - Requests that system to allocate different datasets residing on different removable volumes to the same device during step execution. The ddname is that of an earlier DD statement in the same step. It reduces number of devices used in a job step.

Ex: UNIT=(TAPE,,DEFER) UNIT=AFF=DD1

VOLUME

A reel TAPE or a disk pack is called as one volume. VOLUME parameter is used to identify the volume(s) on which a dataset resides or will reside.

```
VOLUME = ((PRIVATE)
          (,RETAIN)
          (,volume-sequence-number)
          (,volume-count))
          (SER=serial-number1,
              Serial number2……)
```

PRIVATE - Requests a private volume, that is exclusive use of volume for the dataset specified. Only one job can access it at a time. TAPES are PRIVATE by default.

RETAIN - Requests that volume is not to be demounted or rewound after the dataset is closed or at the end of this step. It is used when a following step is to use the same volume.

Volume-sequence-number - identifies the volume of an existing multi volume dataset to be used to begin processing the dataset. (1-255)

Volume-count - Specifies the maximum number of volumes that an output dataset requires. (1-255)

SER-serial-number - Identifies by serial number the volume(s) on which the dataset resides or will reside. 1 through 6 alphanumeric or national (@,#,$) character.
You can code a maximum o 255 volume serials.
Ex : VOLUME=SER=DEV001 VOL=(,,,3,SER=(PAGE01,PAAGE02,PAGE03))

SYSOUT=class | *

It is used to identify this dataset as a system output dataset. The SYSOUT dataset is assigned to an output class. The attributes for each class are defined during JES initialization, including devices or devices for the output class. '*' refer-backs to MSGCLASS character of JOB CARD.

Positional and Keyword Parameter

All the parameters of JOB, DD and EXEC statements can be broadly classified into two types. They are POSITINAL and KEYWORD parameters.

Parameter that has its meaning defined by its position is positional parameter. Bypassing of any positional parameter has to be formed to system by ','. Ex: accounting information and programmer name of job card.

Keyword parameters follow positional parameter and they can be coded in any order. Ex: All the parameters suffixed by '=' are keyword parameters. PGM= and PROC= are exceptions for this rule. They are positional parameters.

In-stream data

The data passed in the JCL stream along with JCL statement is called in-stream data.

Syntax: //SYSIN DD &&&&

&&&&	**Meaning**
*	The data follows from the next line and ends when any // or /* appears at column 1& 2. So '//' and '/*' cannot be passed to the program. //EMPFILE DD * *2052MUTHU* *1099DEV* /*
DATA	The data follows from the next line and ends when any /* appears at column 1 & 2. So '/*' cannot be passed to the program. //SYSUT1 DD DATA *//STEP1 EXEC PGM=INV1040* *//INVLSTA DD SYSOUT=A* *//INVLSTB DD SYSOUT=A* /*
DATA, DLM=@@	The data follows from the next line and ends when the character coded in DLM appears at column 1 & 2. //SYSIN DD DATA,DLM=## *//EMPFILE DD * 2052MUTHU 1099DEV* /* ##

OTHER Statements

OUTLIM

It limits the number of prints lines. The limit ranges from 1 to 16777215. the job is terminated if the limit is reached.

//name DD SYSOUT=*,OUTLIM=3000

If the program tries to write 3001st line, JOB will ABEND with S722.

DEST

The DEST parameter is used in conjunction with the SYSOUT parameter where the output is to be sent. This might be used where a job is run on several MVS systems and the output is directed to a single system for the convenience of the end-user.
Syntax: //name DD SYSOUT=*,DEST=destination-ID

OUTPUT

OUTPUT statement is used to specify SYSOUT parameters for several DD statements with a single JCL statement. It allows printing the output from single DD statement several times, each with different SYSOUT parameter.
COPIES, OUTLIM, CLASS, DEST, FORMS, GROUPID, can be coded in OUTPUT.

DEFAULT=Y can be coded on JOB and STEP level. STEP level default overrides JOBLEVEL default.

```
//TEST#10  JOB    …
//STEP1    EXEC   PGM=ONE
//FORM2    OUTPUT DEFAULT=YES.COPIES=2,DEST=PPP
//SYSPRINT DD     SYSOUT=A                    == > Produces 2 copies at PPP
//STEP2    EXEC   PGM=TWO
//FORM3    OUTPUT COPIES=3,DEST=XYZ
//SYSPRINT DD     SYSOUT=Q,OUTPUT=(STEP2.FORM3,STEP1.FORM2)
                            == > Produces 3 copies at XYZ and 2 copies at PPP.
```

Maximum number of COPIES possible is 254.

FORMS

Specify the type of forms on which the SYSOUT datasets should be printed. It is 1-8 alphanumeric or national characters. SYSOUT DD FORMS parameter overrides OUTPUT PARMS parameter.
//name OUTPUT FORMS=form-name

FREE

The datasets are allocated just before the execution of step and de-allocated after the execution of step. FREE parameter de-allocates the file as soon as the file is closed. //DDNAME DD SYSOUT=X,FREE=CLOSE

INCLUDE

The purpose of INCLUDE statement is same as COPY statement of COBOL program. This is used to specify a PDS member that will be copied into the JCL at job submission time. It is used to specify a standard list of DDNAMES, which would otherwise be duplicated in many similar PROCS. This also has the advantage that amendments need only be made in one place. But it makes JCL unnecessarily fragmented or difficult to read/maintain in a live environment.

```
//          INCLUDE MEMBER1
```

MEMBER1 should exist in the procedure library. Procedure libraries are coded using JCLLIB statement. Include must not be used to execute a PROC. It is possible to nest up to 15 levels of INCLUDE statements.

REFERBACK

The backward reference or refer back permits you to obtain information from a previous JCL statement in the job stream. STAR (*) is the refer-back operator. It improves consistency and makes the coding easier.
DCB, DSN, VOL=SER, OUTPUT, PGM can be referred-back.

Refer back can be done using the following ways:
1.Another DD of the same step will be referred.
*,DDNAME
2.DD of another step can be referred.
*.STEPNAME.DDNAME (DDNAME of the STEPNAME)
3.DD of another proc step can be referred.
*.STEP-INVOKING-PROC.PROC-STEP-NAME.DDNAME

START in the SYSOUT parameter refer back to MSGCLASS of JOB card.

Refer-back example:

```
//STEP1    EXEC PGM=TRANS
//TRANFILE DD   DSNAME=AR.TRANS.FILE,DISP=(NEW,KEEP),
//          UNIT=SYSDA,VOL=SER=MPS800,
//          SPACE=(CYL,(5,1),
//          DCB=(DSORG=PS,RECFM=FB,LRECL=80)
//TRANERR   DD  DSNAME=AR.TRANS.ERR,DISP=(NEW,KEEP),
//          UNIT=SYSDA,VOL=SER=MPS801,
//          SPACE=(CYL,(2,1),
//          DCB=*.TRANFILE
//STEP2    EXEC PGM=TRANSEP
//TRANIN   DD   DSNAME=*.STEP1.TRANFILE,DISP=SHR
//TRANOUT  DD   DSNAME=AR.TRANS.A.FILE,DISP=(NEW,KEEP),
//          UNIT=SYSDA,VOL=REF=*.STEP1.TRANFILE,
//          SPACE=(CYL,(5,1),
//          DCB=*.STEP1.TRANFILE
.
//STEP5    EXEC PGM=*.STEP3.LOADMOD
```

Concatenation Rules

Concatenation allows naming of more than one dataset in a single input file without physically combining them:

```
//STEPLIB DD DSN=PROD.LIBRARY,DISP=SHR
//          DD DSN=TEST.LIBRARY,DISP=SHR
//          DD DSN=USR.LIBRARY,DISP=SHR
```

In this case Prod, Test & User libraries are concatenated.
1.16 PDS or 255 Sequential datasets can be concatenated.
2.LRECL and Record format should be same.
3.If the Block size is different, then largest block size dataset should be first.
4.Datasets may resides on different devices and device types.

Special DD names

STEPLIB

It follows EXEC statement. Load modules will be checked first in this library and then in the system libraries. If it is not found in both places, then the JOB would ABEND with S806 code.

JOBLIB

It follows the job statement. Load modules of any steps (EXEC) that don't have respective STEPLIB will be looked into this PDS. If not found, it will be checked against system libraries. It it is not found there also, then the JOB would ABEND with S806.

JCLLIB

It follows JOB statement. Catalogued procedures in the Job are searched in this PDS. If they are not found, they will be checked in system procedure libraries. If they are not there, then there will be JCLERROR with 'Proc not found' message.
Syntax: //PROCLIB JCLLIB ORDER(PDS1,PDS2)
INCLUDE members are also kept in procedure libraries. (JCLLIB)

ABEND DATASETS

In case of ABEND, one of the following three datasets will be useful. If more than one of the three datasets is coded, then the last coded DD will be effective.

SYSDUMP

Prints the program area, contents of registers, and gives a trace back of subroutines called. It will be in hexadecimal format.

SYABEND

Same as SYSUDUMP, but also prints the system nucleus. Don't use unless you need the nucleus. It will be in hexadecimal format.

SYSMDUMP

Same information as SYSABEND, but dump will be in machine language. Used to store dumps in a data set to be processed by an application program.

JOBCAT and STEPCAT

The datasets used in step are first checked in the STEPCAT (ICF or VSAM Catalog) before checking in system catalog. If no STEPCAT in the step and there is a JOBCAT, then the datasets are first searched in JOBCAT before checking in system catalog.

SYSIN

In-stream data can be coded in SYSIN DD *. Using ACCEPT statement, these records are read into the program. Every accept will read one line into working storage (80 column).

Procedures

Set of Job control statements that are frequently used are saved separately as procedures and invoked in the JOB. The use of procedures helps in minimizing duplication of code and probability of error. This is because a procedure should consist of pre-tested statements. Each time a procedure is invoked, there is no need to retest its functionality, since it is pre-tested.

If a procedure is placed in the same job stream, then it is called In-stream procedure. They start with PROC statement and end with PEND statement. They should be coded before first EXEC statement.

If a procedure exists in a PDS, it is called as catalogued procedure. PEND statement is not needed for catalogued procedures. One procedure can invoke another procedure and this is called nesting. Nesting can be done till 15 levels. But there is an indirect of limit of 255 steps. So we have to make sure than expansion of any proc in the JOB is not exceeding this limit on number of steps.

```
//procedure PROC        <==  names the cataloged/in-stream procedure.
//  JCL statements
//  PEND                 == > Only for instream procedure to point end of an
                             instream proc
```

You execute the JCL statements contained in the cataloged procedure by writing:

```
// EXEC procedure
```

Procedure Modification

Procedure should be generic so that it can easily be used by the multiple JOBS by simple overrides. During the invoking of procedures in the JOB, one can do the following.

1. Override: Change the dataset names or parameters that are already coded in the procedure
2. Addition: Add new datasets or parameters in the already existing steps of the procedure.
3. Nullify: Omit the datasets or parameters that are already coded in procedure.

When you override a cataloged procedure, the override applies just to that execution of the job. The cataloged procedure itself isn't changed.

Procedure Modification- EXEC statements

COND, TIME and PARM values of an EXEC statement in the procedure can be added/modified/nullified in the invoking JCL in the following way.

```
//STEP1 EXEC PROC-NAME, PARAMETER-NAME.STEPNAME-IN-PROC=NEW-VALUE
```

Ex: PROC COBCLG has a statement

```
//COB EXEC PGM=IGYCRCTL,REGION=400K
```

```
// EXEC COBCLG, REGION.COB=1M   => Overrides the value of 400K.
// EXEC COBCLG, TIME.COB=(0,10)  => Adds 10 seconds time limit for COB step.
// EXEC COBCLG, REGION.COB=      => Nullifies 400K region request. Default
                                    Region will be allocated now.
```

Other Rules:
1. Multiple overrides are allowed but they should follow the order. That is first you should override the parameters of step1, then step2 and then step3. Any overrides in the wrong order are IGNORED.
2. If the STEPNAME is not coded during override, then the system applies the override to first step alone.
//EXEC COBCLG,REGION=512K

Procedure Modification- DD Statement
DD statement in the procedure can be modified by
//STEPNAME-IN-PROC.DDNAME-OF-STEP DD parameter of dataset…

1. DD statement overrides should be done in the same order they appear in procedure. Within a DD statement, the order of parameter does not matter.

2. Any additions should follow modification. In a step, if you want to override the dataset attribute of one existing dataset and add another dataset, you should override the old one before adding the new one.

3. To omit a parameter fro ma DD statement in a procedure, just name it but don't pass any value for it.

Procedure Modification Using Symbolic Parameter

A symbolic is a PROC placeholder. The value for the symbolic is supplied when the PROC is invoked. (&symbol=value). If the value is not provided during invoke, then the default value coded in the PROC definition would be used for substitution.

Ex: If you want to override UNIT Parameter value of all the DD statements, define this as symbolic parameter in proc.

Catalog Procedure: PROC1

```
//PROC1 PROC,&UNIT=SYSDA              => SETS Default value of UNIT as SYSDA
//S1      EXEC PGM=TEST1
//DD1 DD  UNIT=&UNIT
//DD2 DD  UNIT=&UNIT
```

//STEP1 EXEC PROC,UNIT=TEMPDA will set &UNIT as TEMPDA for this run of procedure.

Statement Not Allowed in a Procedure

You can place most statements in a procedure, but there are a few exceptions.

1. The JOB statement and JES2/JES3 Control statements.
2. The JOBCAT and JOBLIB statements.
3. An in-stream procedure (an internal PROC/PEND pair)
4. SYSIN DD *, DATA statements

Procedure Example

SMSXL86.TEST.PROCLIB(EMPPROC)

```
//EMPPROC  PROC CLASS='*',SPACE='1,1'
//*default values are coded for the symbolic parameters CLASS and SPACE
//STEP1A   EXEC PGM=EMPPGM
//SYSOUT     DD    SYSOUT=&CLASS
//EMPMAST   DD    DSN=&HLQ..EMPLOYEE.EDS,DISP=SHR
//             DD    DSN=&HLQ..EMPLOYEE.MBT,DISP=SHR
//             DD    DSN=&HLQ..EMPLOYEE.IBM,DISP=SHR
//* &HLQ is a symbolic parameter that don't have default value. So it has to be
//* provided during the invoke of this proc.
//EMPOUT     DD    DSN=&&INVSEL,DISP=(NEW,PASS),
//                   UNIT=SYSDA,SPACE=(CYL,(&SPACE))
//INVSEL is a temporary dataset.
//EMPCNTL  DD   DUMMY
//* EMPCNTL is a control card and any in-stream data can be coded during the
//* invoke.
//*
//INV3020  EXEC PGM=EMPRPT
//SYSOUT   DD   SYSOUT=&CLASS
//INVMAST  DD   DSNNAME=&&INVSEL,DISP=(OLD,DELET)
//*temporary dataset INVSEL is deleted here.
//INVSLST   DD   SYSOUT=&CLASS
```

SMSXL86.TEST.JCLLIB(EMPJCL)

```
//EMPJCLA JOB (1000,200),CLASS=A,MSGCLASS=Q,NOTIFY=&SYSUID
//PROCLIB JCLLIB ORDER=(SMSXL86.TEST.PRCLIB)
//  SET SPACE='1,1'
//*SET statement is used for substituting the symbolic variables.
//STEP01 EXEC  EMPPROC,PARM.STEP1A='02/11/1979',HLQ=PROD
//* STEP1A PARM is added and value for symbolic parameter HLQ is supplied.
//STEP1A.EMPMAST  DD
//                      DD DSN=PROD.EMPLOYEE.CTS,DISP=SHR
//*Instaed of PROD.EMPLOYEE.MBT, PROD.EMPLOYEE.CTS dataset is used whereas
//*other two datasets PROD.EMPLOYEE.EDS and PROD.EMPLOYEE.IBM retains there
//*position in concatenation.
//STEP1A.EMPOUT DD UNIT=TEMPDA
//*UNIT parameter of EMPOUT file is modified
//STEP1A.EMPCNTL DD *
DESIG=SSE
/*
//*EMPCNTL control card value is passed.
//STEP1A.EMPOUT2 DD DSN=PROD.EMPLOYEE.CONCAT,
//                        DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
//                        SPACE=(CYL,(10,10))
//*EMPOUT2 file is added to the step STEP1A.
```

In the above example, CLASS retains the default value coded on the PROC definition Statement (CLASS='*').

IBM Utilities

IBM has provided utilities for the frequent file operations, as application programmer used to do. By familiar with these utilities, one can effectively develop an application by reducing the work in programming side. For example, if you want to select particular set of records from a master file for special processing, then you can easily complete this task if you are aware of SORT utility. Otherwise you would have gone for a new program coding, compilation, testing etc.

IKJEFT01

IKJEFT01 is known as terminal monitor program. All the TSO commands can be executed in batch using this program. DB2 program are run in batch using this program. IKJEFT1A and IKJEFT1B are two alternate entry point of this program.

Example: IKJEFT01 for TSO command execution

```
//STEP EXEC PGM= IKJEFT01
//SYSPRNT DD SYSOUT=*
//SYSTSIN DD *
  SE 'HELLO..HOWZ THE PREPARATION GOING ON:'  U(ABCD)
/*
```

The above step executes TSO SEND command in batch. It sends the message to user ABCD.

IEFBR14

It is a dummy Program. This is two line assemble program which Clears register 15 and branch to register 14. So whatever operation you do with IEFBR14, you will get return code of 0. In this sense, we cannot call this as utility program. But it is as useful as any other utility program.

If you return the job without deleting the dataset created by the job in the prior run, then your job will end with JCLERROR with message 'duplicate dataset name found'. (NOTCAT2). To avoid this, all the dataset are deleted in the first step using IEFBR14. Disposition of the dataset should be (MOD,DEL,DEL) so that if the dataset is not found, it will be allocated and deleted.

It can be used for allocating datasets.

It can be used for deleting the datasets in TAPE, which you cannot do in TSO.

Example:

```
//STEP1   EXEC PGM=IEFBR14
//DD1     DD   DSNAME=MM01.COPYLIB.COBOL,DISP=(NEW,CATLG),
//           UNIT=SYSDA,VOL-SER=MPS8BV,
//           SPCAE=(3200,(1000,250,5),,,ROUND),
//           DCB=(DSORG=PO,RECFM=FB,LRECL=80)
//DD2     DD    DSNAME=MM01.TEST.DATA,DISP=(OLD,DELETE)
```

Cause OS/390 to allocate and catalog the data set named MM01.COPYLIB.COBOL and delete the data set named MM01.TEST.DATA.

IEBCOPY

It is used to copy one or more member from an existing dataset to a new or existing PDS dataset. It can be also used for compressing PDS, Loading PDS to TAPE and unloading from TAPE to disk. This utility needs two work files SYSUT3 and SYSUT4 in addition to SYSIN and SYSORINT.

FIELD	**Meaning**
COPY	Function is COPY
SELECT	Specifies the member to be copied/replaced Syntax: (NAME-IN-OUTPUT,NAME-IN-OUTPUT,REPLACE-IF-EXISTS)
EXCLUDE	Specifies the member to be excluded from copy
LIST=YES	Displays the copied members in the SYSPRINT.
INDD	Points to input dataset
OUTDD	Points to output dataset. Should exist on the same line of COPY.

IEBCOPY- CONTROL CARD FOR MERGING TWO LIBRARIES

```
//SYSIN DD *
COPY OUTDD-OUTPUT INDD=(INPUT01,(INPUT02,R),LIST=NO)
/*
```

It says DD statements INPUT01 and INPUT02 are input files. OUTPUT is the output file. Note the 'R" in (INOUT02,R). IT instruct to IEBCOPY that like named members are to be replaced. LIST=NO indicates that the names of the members copied need to be listed in the SYSPRINT dataset.

IEBCOPY-CONTROL CARD FOR SELECTIVE COPY/REPLACE

```
COPY OUTDD=OUTPUT,INDD=INPUT01
SELECT MEMBER=((MEM1,NEWNAME,R),(MEM2,,R))
```

MEM1 is copied as NEWMEM in OUTPUT. If already NEWMEM exist, it will be replaced.

IEBCOPY-CONTROL CARD FOR OMITTING SELECTED MEMBERS

```
COPY OUTDD=OUTPUT,INDD=INPUT01
EXCLUDE MEMBER=(MEM1,MEM2)
```

All the member except MEM1 and MEM2 are copied into OUTPUT from INPUT01.

IEBCOPY-Complete step for Compressing PDS

```
//COMPRESS EXEC PGM=IEBCOPY
//SYSPRINT DD   SYSOUT=*
//COMPFILE DD   DSN=MM01.COPYLIB.COB,DISP=OLD
//SYSUT3    DD   UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSUT4    DD   UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSIN     DD   *
 COPY OUTDD=COMPFILE,INDD=COMPFILE
/*
```

IEBGENER

In addition to SYSIN and SYSPRINT datasets, it needs SYSUT1 and SYSUT2 datasets. SYSUT1 is coded with input dataset and SYSUt2 is coded with output dataset. If attributes were not given for SYSUT2, then the program would assume SYSUT1 attribute for SYSUT2.

It is primarily used as COPY utility. If you want to copy any TAPE file to DISK or DISK to TAPE, then no SYSIN is needed.

If you want to reformat your input file or if you want to create members out of your PS file, then you need control card (SYSIN) and the first statement should be GENERATE.

FIELD	**Meaning**
GENERATE	First statement which sets the value for MAXNAME,MAXGPS, MAXLITS, MAXFLDS
MAXNAME	Maximum MEMBER statements that can follow.(During member generation) Syntax: MAXNAME=3
MAXGPS	Maximum IDENT statement that can follow. (During member generation)
MAXFLD	Maximum FIELD statements that can follow.(During reformatting) Syntax: MAXFLDS=10
MAXLITS	Maximum size of literal during reformatting.
MEMBER	It identifies the name of the member to be created. Syntax: MEMBER NAME=MM!
RECORD IDENT	It usually follows MEMBER statement to identify the last record to be copied from the input dataset. RECORD IDENT= (Length,'Literal',Start-Column) Example: RECORD IDENT=(3,'MVS',1), then the last record to be copied into the member from the input dataset, has MVS in column 1-3.
RECORD FIELD	It is used for reformatting the record in the input file. RECORD FILED=(Length, 'literal' or input column, conversion, output column) Output column says where the field should be placed in the output file. Conversion can be ZP or PZ. PZ means the input packed decimal field is being converted into zoned format and ZP is the reverse

<u>IEBGENER- SYSIN CARD FOR CREATING THREE MEMBERS FROM INPUT PS FILE</u>

```
//SYSIN DD *
 GENERATE MAXNAME=3,MAXGPS=2
 MEMBER NAME= MEMB1
 RECORD IDENT=(8,'11111111',1)
 MEMBER NAME= MEMB2
 RECORD IDENT=(8,'22222222',1)
MEMBER NAME= MEMB3
//
```

IEBGENER creates three members. It reads input file writes into memb1 until it finds 11111111 in column 1. In the same way it reads and writes the record into memb2 until it finds 22222222 in column 1. The remaining records in the input datsets are copied into MEMB3.

IEBGENER- SYSIN CARD FOR REFORMATTING DURING COPY

```
//SYSIN     DD    *
 GENERATE MAXFLD=5,MAXLITS=4
 RECORD FIELD=(5,1,,1),FIELD=(20,21,,6),FIELD=(9,61,ZP,26),            X
            FIELD=(9,70,zp,31),field=(4,'TEST',,36)
/*
```

Input Column	Any Conversion	Output Column
Values in column 1-5		Copied into column 1-5
Values in column 21-40		Copied into column 6-25
Values in column 61-9	Convert the zoned into packed before copying.	Packed value is written in 26-30
Values in column 70-9	Convert the zoned into packed before copying.	Packed value is written in 31-35
		TEST literal is written in column 36-39

IEHLIST

It is used to list

1. The entries in the catalog. (SYSIN KEYWORD- LISTCTLG)
2. Directory(s) of 1-10 PDS (SYSIN KEYWORD- LISTPDS)
3. The entries in VTOC. (SYSIN KEYWORD-LISTVTOC)

Code SYSIN, SYSPRINT and one more DD that will mount the volume queried in SYSIN.

Example:

The following JOB lists the VTOC of INTB01 in a formatte way.

```
//MYJOB    JOB  CLASS=A,MSHCLASS=A,REGION=256KMSGLEVEL=(1,1)
//LISTVTOC EXEC PGM=IEHLIST
//SYSPRINT  DD  SYSOUT=*
//VOLDD     DD  UNIT=SYSDA,VOL=SER=INTB01,DISP=OLD
//SYSIN     DD  *
  LISTVTOC  FORMAT,VOL=3330=INTB01
/*
```

To list the contents of any PDS, replace the LISTVTOC command in the SYSIN card with this: LSTPDS DSNAME=(your.pds.name)

IEBCOMPR

It is used to compare two PS or PDS datasets. Two PS are same, if the number of records is same and all the records are identical. SYSIN is not needed for PS comparison. If they are not identical, then the following will be listed in the SYSPRINT.

DD statements that define the dataset, Record and Block numbers, the unequal records and maximum of 10 unequal records found.

Two PDS are same, if the corresponding members contain same number of records and all the records are identical. SYSIN should have COMPARE TYPE=PO for PDS.

```
//SYSUT1 INPUT DATASET 1
//SYSUT2 INPUT DATASET 2
//SYSPRINT
//SYSIN DD *
```

DFSORT

If you do a global search of your JCL inventory, you will find the program that is used very frequently is SORT. There are two famous SORT products are available in the market. One is DFSORT and the other is SYNCSORT. The basic commands in both the products are same.

ICETOOL provides a lot more than what SORT can offer and it comes with DFSORT product. SYNCTOOL comes with SYNCSORT product. PGM=SORT can point to DFSORT or SYNCSORT. It is actually an alias to SORT product in your installation.

DFSORT is IBM product and it needs the following datasets for its operation. SORTIN (Input dataset), SORTOUT (Output dataset), SYSIN (Control Card) and SYSOUT (Message dataset).

Message dataset can be altered using MSGDDN= parameter of SYSIN.

SORT Card to copy all the records from SORTIN to SORTOUT
SORT FIELDS=COPY.

SORT card to skip first 100 records and then copy 20 records
SORT FIELDS=COPY SKIPREC=100 STOPAFT=20

SORT cards to sort he records based on key fields
SORT FIELDS=(STARTPOS,LENGTH,TYPE,**A**SC|**D**ESC)
Type = CH (Character), BI (Binary), ZD (Zoned Decimal), PD (Packed Decimal),
FS (Signed numbers)

Ex: SORT FIELDS=(1,10,CH,A,15,2,CH,A)
SORTS all the SORTIN records with 1-10th column as major key and 15-16th column as minor key before writing to SORTOUT.

SORT card to select the records meeting the criteria
INCLUDE COND=(STARTPOS,LENGTH,TYPE,RO,VALUE)
RO-Relational operator can be EQ,NE,LT,GT,LE,GE.
Card to select the records with TRICHY in the column 4-9
INCLUDE COND= (4,6,CH,EQ,C'TRICHY')
Card to select the records which has same values in 2-3 and 5-6
INCLUDE COND= (2,2,CH,EQ,5,2,CH)

SORT card to reject the records meeting the criteria
OMIT CARD=(STARTPOS,LENGTH,TYPE,RO,VALUE)
Card to reject the records with TRICHY in the column 4-9
OMIT COND= (4,6,CH,EQ,C'TRICHY')
Card to reject the records which has same values in 2-3 and 5-6
OMIT COND= (2,2,CH,EQ,5,2,CH)

Sort card to change the PD to ZD
If input file has a PD field S9(5)V99 Comp-3 and to reformat as PIC S9(5).9(2) then use,
OUTREC FIELDS=(1,5,PD,EDIT(STTTTT.TT),SIGNS=(,-,,))

SORT card to remove the duplicates
SORT FIELDS= (1,5,CH,A),EQUALS
SUM FIELDS+NONE

SORTIN records are sorted on the key 1-5 and if more than one record is found to have same key, then only one record will be written to SORTOUT. If EQUALS is coded, then the record to be written is the FIRST record else it can be anything.

SORT card to sum the values for same-key records
SORT FIELDS= (1,5,CH,A),EQUALS
SUM FIELDS=(10,5,PD)

SORTIN records are sorted on key 1-5 and if more than one record is found to have same key, then the records are summed on column 10-14 and one record is written with total sum.

SORT cad to add sequence number to the output file

OUTREC=(1,20,SEQNUM,4,ZD) ➔ 4 digit zoned decimal sequence number is added with all he records of input file in column 21-24

This will be useful for reading a file from bottom to top. This will be useful for matching logic in JCL. Matching logic in JCL will be explained later.

SORT card to restructure the input file before feeding to sort
INREC FIELDS=(37,2,6,6,40,4,31,2)
The length of the output file is 14.

SORT card to create multiple files from single input file (Maximum 32 files)
OUTFIL FIELS=1 INCLUDE=(1,6,CH,EQ,C'MUMBAI')
OUTFIL FIELS=2 INCLUDE=(1,6,CH,EQ,C'TRICHY')
Code output files as SORTOF1 and SORTOF2.

SORT card to restructure the sorted file before writing

```
OUTREC FIELDS=(1:1,20,          => FIRST 20 CHAR FROM INPUT FILE
               21:C'MUTHU',     = > FOLLOWED BY STRING MUTHU
               26:10Z,          => FOLLOWED BY 10 BINARY ZEROS
               36:21,10)        => 21st to 10 CHARACTERS FROM INPUT FILE.
```

SORT card to change any 'XX' in the column 6-7 to 'YY'

```
OUTREC FIELDS=(1:2,5,
               6:1,2,CHANGE=(2,C'XX',C'YY'),NOMATCH=(6,2),
               8,42)
```

SORT card to merge
MERGE FIELDS=(STARTPOS,LENGTH,TYPE,**A**SC|**D**ESC,STARTPOS,…)
128 such Keys can be given. Datasets to be merged are coded in SORTIN00 to SORTIN99.

ICETOOL

DD statements in ICETOOL:

TOOLMSG	FOR	ICETOOL MESSAGES
DFSMSG	FOR	SORT MESSAGES
TOOLIN	FOR	ICETOOL-CONTROL-CARD
XXXXCNTL	FOR	SORT-CONTROL-CARD USED BY ICETOOL XXXX is coded in USING clause of TOOLIN.

TOOLIN card to copy
COPY FROM(INDD) TO(OUTDD) (Up-to 10 DD can be given).

TOOLIN card to copy only unique records
UNIQUE FROM(INDD) ON(STARTPOS,LENGTH,TYPE)
COPY FROM(INDD) TO(OUTDD)

TOOLIN card to copy unique/selected duplicate records
SELECT FROM(INDD) TO(OUTDD) ON (STARTPOS,LENGTH,TYPE)
NODUPS/ALLDUPS/LOWER(n)/HIGHER(n)/EQUAL(n)/FIRST/LAST

NODUPS	- COPY only the unique records.
ALLDUPS	- COPY only the duplicates.
HIGHER(n)	- COPY those duplicates that occurs more than n times (n = > 1-99)
LOWER(n)	- COPY those duplicates that occurs less than n times (n = > 1-99)
EQUAL(n)	- COPY those duplicates that occurs exactly n times (n = > 1-99)
FIRST	- Retains the first copy in case of duplicates
LAST	- Retains the last copy in case of duplicates

TOOLIN Card to get the statistics of a numeric field
STATS FROM(INDD) ON(START,LENGTH,TYPE)
Print the maximum, average and total for numeric fields. (10 ON possible)

TOOLIN Card to get the number of unique values in a field
UNIQUE FROM(INDD) ON(START,LENGTH,TYPE)
Print count of unique values.

TOOLIN Card to get all the values for a particular field
DISPLAY FROM(INDD) ON(STARTPOS,LENGTH,TYPE) LIST(LISTDD)
Prints values of a field in the input dataset to :ISTDD. (20 ON possible)

TOOLIN Card to get al the values for a particular field – With Occurrence constraint
OCCURS FROM(INDD) ON(STARTPOS,LENGTH,TYPE) LIST(LISTDD) OPTION
OPTION = > HIGHER(n) LOWER(n) EQUAL(n) ALLDUPS NODUPS
HIGHER(2) means only the values that are repeated more than 2 times is reported at LISTDD dataset.

TOOOLIN Card to get number of records fell into the range mentioned
RANGE FROM(INDD) ON(START,LENGTH,FORMAT) LIST(OUTDD) options

Options are OCCURS HIGHER(n)/LOWER(n)/HIGHER(n1) LOWER(n2)/
EQUAL(n) NOTEQUAL(n)
It prints count of records meeting value criteria and the FORMAT should be numeric.

TOOLIN card to invoke SORT

SORT FROM(INDD) TO(OUTDD) USING(xxxx)

SORT statement should be coded under the DDNAME xxxxCNTL

Matching Logic in JCL

I have two files file 1 and file2. I want t generate three reports out of these two files.

1.The first report should have records that exist in both files.

2.The second report should contain records that exist only in first file and not in second file.

3.The third report should contain records that exist only in the second file and not in the first file.

```
//STEP0100 EXEC PGM=ICETOOL
//*
//TOOLMSG   DD SYSOUT=*
//DFSMSG    DD SYSOUT=*
//IN1       DD *
1234567890
3456789012
5678901234
//IN2       DD *
3456789012
7890123456
8901234567
//T1        DD DSN=&T1,SPACE=(CYL,(5,5),RLSE),DISP=(,PASS)
//T2        DD DSN=&T2,SPACE=(CYL,(5,5),RLSE),DISP=(,PASS)
//INT       DD DSN=*.T2,DISP=(OLD,PASS),VOL=REF=*.T1
//          DD DSN=*.T2,DISP=(OLD,PASS),VOL=REF=*.T2
//FILEA     DD SYSOUT=*
//FILEB     DD SYSOUT=*
//OUT       DD SYSOUT=*
//TOOLIN    DD   *
 SORT FROM(IN1) USING(CTL1)
 SORT FROM(IN2) USING(CTL2)
 SORT FROM(INT) USING(CTL3)
//CT1CNTL DD *
 SORT FIELDS=(1,10,CH,A)
 OUTFIL FNAMES=T1,OUTREC=(1,80,C'1')
//CT2CNTL DD *
 SORT FIELDS=(1,10,CH,A)
 OUTFIL FNAMES=T2,OUTREC=(1,80,C'2')
//CT3CNTL DD *
 SORT FIELDS=(1,10,CH,A)
 SUM FIELDS=(81,1,ZD)
 OUTFIL FNAMES=OUT,INCLUDE=(81,1,ZD,EQ,3),OUTREC=(1,80)
 OUTFIL FNAMES=FILEA,INCLUDE=(81,1,CH,EQ,C'1'),OUTREC=(1,80)
 OUTFIL FNAMES=FILEB,INCLUDE=(81,1,CH,EQ,C'2'),OUTREC=(1,80)
/*
```

Explanation:
CTL1 – Add 1 to all the records of the first file at 80th column
CTL2 – Add 2 to all the records of the second file at 80th column
CTl3 – Concatenate both files and sort the file on key if duplicates found, sum on 81st column. So if any record exists in both the file, it will have 3 after summing.
So now extract records with '1', '2' and '3' into three files. While writing the records, remove the 81st byte added for our temporary purpose.

'1' – Records only in first file
'2' – Records only in second file.
'3' – Records exist in both the files.

IEHPROGM
It is used to
1.Catalogued a dataset (CATLG DSNNAME=A.B.C, VOL=SER=nnnn)
2.Uncatalog a dataset (UNCATLG DSNNAME=A.B.C)
3.Rename a dataset (RENAME DSNAME=A.B.C,VOL=SER=nnnn,NEWNAME=D>E>F)
4.Create an index for GDG (BLDG INDEX=gdg-name, LIMIT=n, [,EMPTY][,DELET])
5.Deleting the index for GDG (DTLX INDEX=index-name)
The SYSIN cards are given in bracket. The utility needs SYSUT1 and SYSUT2 work space and SYSPRINT for messages.

IDCAMS will be covered in detail, in VSAM section of this book.

Generation Data Group (GDG)

GDG is group of datasets that are related to each other chronologically or functionally. Each of these dataset is called a generation. The generation number distinguishes each generation from others.

If the GDG Base is MM01.PAYROLL.MASTER, then their generations are identified using the generic name "MM01.PAYROLL.MASTER.GnnnnVxx."
nnnn is generation number (01-9999) and xx is version number (00-99).

Referring Generation in JCL

The current generation is referred by GDGBASE(0). The previous generation by GDGBASE(-1) and the next generation by GDGBASE(+1).

GENERATIONS ARE UPDATED ONLY AT THE END OF THE JOB. It means, if the first step creates one generation, you code it as GDGBASE(+1) and if the second step creates another generation, then you SHOULD code this as GDGBASE(+2) as the (+1) version is not yet promoted as current version. Similarly if you want to refer the GDG created in the second step down the line, you have to refer it by GDGBASE(+2).

GDG datasets can be also referenced with their generation number like 'MM01.PAYROLL.MASTER.G001V00'

Advantage of GDG

1. GDG datasets are referred in the JCL using GDG base and relative number. So the same JCL can be used again and again without changing the dataset name and this is the biggest advantage of GDG.
2.GDG Base has pointers to all its generations. When you want to read all the transactions done till today, you can easily do it by reading the GDG base if it is available. Otherwise you have to concatenate all the transaction files before reading.

Creation of GDG

1.GDG Base is created using IDCAMS. The parameter given while creating the GDG are:

Parameter	**Purpose**
NAME	Base of the GDG is given here.
LIMIT	The maximum number of GDG version that can exist at any point of time. It is a number and should be less than 256.
EMPTY/NOEMPTY	When the LIMIT is exceeded, EMLTY keeps ONLY the most recent generation. NOEMPTY keeps the LIMIT number of newest generation.
SCRATCH/ NOSCRATCH	SCRATCH un-catalogue and deletes the versions that are not kept. NOSCRATCH just does un-cataloguing and it is not physically deleted from the volume
OWNER	Owner of the GDG.
FOR DAYS (n) / TO (DATE)	Expiry date. Can be coded either in the unit of days or till particular date.

2. Model dataset is defined after or along with the creation of base. Once model DCB is defined, then during the allocation of new versions, we no need to code DCB parameter. Model DCB parameter can be overridden by coding new parameter while creating the GDG version. It is to note that two GDG version can exist in two different formats.

A step that defines a GDG and allocates a model DSCB

```
//GDG        EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//MODEL    DD   DSNAME=MM01.PAYROLL.MASTER,DISP=(,KEEP),
//          UNIT=SYSDA,VOL=SER=MPS800,SPACE=(TRK,(0)),
//          DCB=(DSORG=PS,RECFM=FB,LRECL=400)
//SYSIN   DD   *
 DEFINE GDG ( NAME(MM01.PAYROLL.MASTER) –
          LIMT(5)                -
          MOEMPTY                  -
          SCRATCH )
/*
```

<u>GDG VERSIONS ARE UPDATED ONLY AT THE END OF THE JOB</u>

<u>Two step that updates GDG master base twice:</u>

```
//MM01PY1  JOB  (36512),'MUTHU',NOTIFY=&SYSUID
//UPDATE1  EXEC PGM=PAY3200
//OLDMAST  DD   DSNAME=MM01.PAYROLL.MASTER(0),DISP=OLD
//NEWMAST  DD   DSNAME=MM01.PAYROLL.MASTER(+1),DISP=(NEW,CATLG),
//          UNIT=SYSDA,VOL=SER=MPS8BV,
//          SPACE=(CYL,(10,1)
//*
//UPDATE2  EXEC PGM=PAY3210
//OLDMAST  DD   DSNAME=MM01.PAYROLL.MASTER(+1),DISP=OLD
//NEWMAST  DD   DSNAME=MM01.PAYROLL.MASTER(+2),DISP=(NEW,CATLG),
//          UNIT=SYSDA,VOL=SER=MPS8BV,
//          SPACE=(CYL,(10,1))
```

<u>Two on-step jobs that update the GDG master file</u>

```
//MM01PY2  JOB  (36512),'MUTHU',NOTIFY=&SYSUID
//UPDATE1  EXEC PGM=PAY3200
//OLDMAST  DD   DSNAME=MM01.PAYROLL.MASTER(0),DISP=OLD
//NEWMAST  DD   DSNAME=MM01.PAYROLL.MASTER(+1),DISP=(NEW,CATLG),
//          UNIT=SYSDA,VOL=SER=MPS8BV,
//          SPACE=(CYL,(10,1)),
//          DCB=(LRECL=80)
//*
//MM01PY3  JOB  (36512),'MUTHU',NOTIFY=&SYSUID
//UPDATE2  EXEC PGM=PAY3210
//OLDMAST  DD   DSNAME=MM01.PAYROLL.MASTER(0),DISP=OLD
//NEWMAST  DD   DSNAME=MM01.PAYROLL.MASTER(+1),DISP=(NEW,CATLG),
//          UNIT=SYSDA,VOL=SER=MPS8BV,
//          SPCE=(CYL,(10,1))
```

ABENDS

If your program ABENDED, first check whether the ABEND is SYSTEM ABEND or USER ABEND. (Snnn OR Unnn). If system ABEND, check the ABEND code in QW. If needed, identify the statement that caused the ABEND using offset. If the ABEND is user ABEND, then check the ABEND code in the program and study the reason from the comment section/functionality of the program.

SB37, SD37, SE37	Space ABEND. Refer Space parameter for solution.
S80A, S804, S822	Region problem. Refer REGION parameter for solution.
S122, S222	Job is cancelled. S122 – Operator cancelled your job as it requests some unavailable resource. S222-You cancelled your job.
S322	TIME OUT. Refer TIME parameter for solution.
S522	JOB exceeded maximum wait time.
S722	Output lines exceeded the limit set by OUTLIM OR LINES parameter.
S706	Load module found but it is not executable.
S806	Load module not found.
SOC1	Operation exception. Misspelled DD names
SOC4	Protection exception – trying to access a memory location for which you don't have access. Ex: Accessing 11th element of an array of size 10 in the program compiled with SSRANGE, Trying to read/write an unopened file.
SOC5	Addressing exceptions – Trying to access a memory location that is not available in memory.
SOC7	Data exception: Non-numeric operation on numeric field. It is usually due to un-initialized numeric item.
SOC8-SOC9	Fixed Point- Overflow and Divide exceptions respectively.
SOCA, SOCB	Decimal Point- Overflow and Divide exception respectively.
SOCC, SOCD	Floating point- Exponent Underflow and overflow exceptions respectively.
S013	Open Problem. Usually this ABEND occurs, when the program tries to read a member of PDS and the member is not found. Ex: During compilation, if you code a non-existing member as source (SYSIN).
S878	Memory issue. Usually this ABEND occurs in TSO, while executing your job using XPEDITOR. Close any other screens that are open and then try to re-execute the JOB.

How to Solve System ABENDS (SOC4,SOC7)?

1.Refer the SYSOUT of the job and get the next sequential instruction to be executed (Offset).
2.Compile the program with LIST option.
3.Check for the offset in the compilation list and Get the respective statement number.
4.Identfy the statement. This would be a numeric operation on non-numeric data.
5.Identify the source of the numeric data and correct it.

SYSOUT Message:
From compile unit TESTPGM at entry point TESTPGM at compile unit offset +0000043A *at entry offset +0000043A at address 00008116.*

In the compilation listing:
INVOCATION PARMAETERS:
NONUM,MAP,XREF,OBJ,OFF,BUF(28672),SSR,NOCMPR2,DATA(31),SSRANGE,*LIST*

LINE #	HEXLOC	VERB	LINE #	HEXLOC	VERB
000046	0003BE	MOVE	000047	0003C4	MOVE
000049	00042E	MOVE	000050	*00043A*	*COMPUTE*

43A is a part of statement number 50

Refer the statement number in the listing/Program:

5000 COMPUTE ACOMP = ACOMP / 3

So the problem is with the arithmetic operation of ACOMP.
Find the sources of the field after confirming the junk value in it.
Delete the corrupted record, the sources for this field, from the file and ran the job again.
(Production Support people- Need to check the importance of the data and correct or delete accordingly.)

Submission of Job from COBOL Program

Write the Job statement to a output file and route the file to INTRDR instead of cataloguing. INTRDR-Internal Reader

```
JOB: //STEP1 EXEC PGM=MAINPM
     //DD1    DD   DSN=MUTHU.TEST,DISP=SHR
     //JCLDD  DD   SYSOUT=(*,INTRDR)

PROGRAM:MAINPGM
     SELECT JCLFILE ASSIGN TO JCLDD….  (Environment Division)
     FD JCLFILE.
     01 JCL-REC PIC X(80).        (File Section)
     OPEN OUTPUT JCLFILE.        (Open in output and write JCL statements)
     MOVE '//TESTJOB JOB 1111'           TO JCLREC.
     MOVE '//STEP01 EXEC PGM=IEFBR14' TO JCLREC.
     CLOSE JCLFILE                (TESTJOB will be submitted automatically)
```

Submission of Job from CICS Program

Open the SPOOL using SPOOL OPEN command. Pass all the JCL statements to a COBOL variable (should be declared using OCCURS Clause) and then write the line one by one to the SPOOL using SPOOL WRITE command. While Closing the SPOOL using SPOOL CLOSE, the job will be submitted.

Storage Management Subsystem

It is optional feature of the operating system that manages user datasets. The SMS parameters apply only to the new datasets and the system ignores for existing datasets, DD* or DD DATA datasets, JOBCAT and JOBLIB DD statements, and SYSOUT datasets.

The following data classes establish various default values for catalogued datasets. An administrator assigns a name to each group of default values, and then you reference this name on your DD statement to use the values.

STORCLAS – Defines UNIT and VOL parameters
DATACLAS - Defines RECFM, LRECL, AVGREC, SPACE, RETPD/EXTPD, DSNTYPE, KEYLEN, RECORG, KEYOFF..etc
MGMTCLAS – Migration, backup, releasing unused space in a dataset.

If you want to override any one of the values of default, you can do that.

```
//PDS DD DSN=BPMAIN.MUTHU.SOURCE,DISP=(NEW,CATLG),
//          STORCLAS=DASDONE,SPACE=(,(,,50)),DATACLAS=COB2
```

➔ Overrides the directory space defined for COB2 data class.

JCL for SAR-IN-BATCH

The production Job logs are usually routed to one of the third party products. SAR is one such product. The other products are CA-VIEW and VIEW-DIRECT. The following JCL is used to query the SAR in batch.

```
//LOADLOG EXEC PGM=SARBCH
//STELIB     DD  DSN=SYSP.CAI.CAILIB,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//REPORT    DD  DSN=CORP.DBSY2KT.JOBRPT,DISP=MOD
//LOADDD   DD  DSN=CORP.DBSY2KT.LOADLOG,DISP=OLD
//SYSIN      DD  DSN=CORP.DBSY2KT.UTILS(CYCCARDW),DISP=SHR
```

To load the complete log (JOB PRODBKUP of generation 4941) into a dataset named as LOADDD, use the following card:

```
/LOAD DDNAME=LOADDD ID=PRODBKUP GEN=4941
```

To get run-date, run-time, return code and generation of all the prior runs of a job, use the following card. The result will be stored in the dataset named as REPORT.

```
/LIST ID=JOBNAME
```

JCL for CA7-IN-BATCH

CA7 is a scheduler product. We brief about the product in the next section. This JCL is used to query CA7 in batch.

```
//UCC77BTI  PROC  ID=0,POOL='1-8',DSNPFX=,OPT=,PG=SASSBSTR
//BTERM     EXEC  PGM=SAASBSTR,
//            PARM='0,POOL=(1-8)'
//STEPLIB      DD  DISP=SHR,DSN=CAI.CA7.LOADLIB
//UCC7CMDS  DD  DISP=SHR,DSN=CAI.CA7.COMMDS
//SYSPRINT    DD  SYSOUT=*,DCB=BLKSIZE=133
//ERRORS      DD  SYSOUT=*,DCB=BLKSIZE=133
//SYSUDUMP  DD  SYSOUT=*
//SYSIN DD*
 Code the CA7 commands
/*
```

JCL FOR XDC-IN-BATCH

There may be a need to read the run details of a job log inside the REXX. This can be done using the OUTPUT command of TSO. The following JCL capture the complete log information of the job CANNAEFD with ID JOB08079 into T.ISP.MUTHU(TEST1)

```
//RUNTSO EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD *
OUTPUT CENNAEFD(JOB08079) PRINT('T.ISP.MUTHU(TEST1)')
/*
//*
```

JCL FOR QW-IN-BATCH

QW is the ONLINE-help available in mainframe.
When the Information/Warning/Error/ABEND displayed in the JES message is not descriptive or syntax of any command, one can easily refer in QW>

QW searches the specified string in all the manuals available. If the string is found in more than one manual, it lists all the manuals. We have to choose the manual and proceed accordingly.

The following JCL can be used to invoke QW in batch to download the manuals available in WQ.

```
//SDRXG12A JOB (ACCT),'ICETOOL',CLASS=A
// MSGCLASS=X,MSGLEVEL=(1,1),REGION=0M,NOTIFY=&SYSUID
//QWBATCH EXEC PGM=QWIKREF1,PARM='V=* P=* R=V1R14,I=ICETOOL'
//STEPLIB    DD DSN=SYS5.QWF.LINKLIB,DISP=SHR
//QWPRINT  DD DSN=SDRXG12.QW.ICETOOL,DISP=(,CATLG,DELETE),
//                DCB=(RECFM=VBA,LRECL=133,BLKSIZE=6000),
//               SPACE=(CYL,(5,5),RLSE),UNI=SYSDA
//QWREFDD DD DSN=SYS50.QWF.DATABASE,DISP=SHR
```

This Will get the V1R14 ICETOOL document into QWPRINT. (PARM='V=* P=* R=V1R14 I=ICETOOL').

QREFDD is the Database of manuals and STEPLIB must have the load QWIKREF1. These datasets may differ from installation to installation. QINFO on QW panel lists this information for the installation.

Scheduler: CA-7

JES selects the job for processing. Once you submitted the job, you lose your control over the job. JES checks for the availability of initiators for the CLASS of the job submitted by you. If an initiator is available, it checks for any high priority (PRTY) JOB with same class is available. If not, based on FIFO policy it selects the job submitted by you.

So we cannot call JES as a scheduler. It just processes the job submitted by you. But a scheduler schedules the jobs at the time and the successful completion of the first job may trigger the second job automatically and the second can trigger third and this is how the whole business BATCH cycle is implemented in production system. Preciously saying, the trigger need to be always job but can be also dataset(s) or time. Other constraints like mutual exclusiveness, dependencies can be also added to a job during the definition in the scheduler. Well-defined jobs needs less human intervention as the jobs are automatically submitted at the right time.

Obliviously, if any job is ABENDED, human intervention is needed and that is a crucial part of any production support project. If the person in shift could not solve the problem and restart the job within the time, then the full cycle may get delayed and the ONLINE may not be up at the time agreed in SLA (Service Level Agreement).

There are lot of schedulers are available in the market. Control-M, OPC and CA-7 are few of them. Discussion of scheduler products is outside the scope of the book but we would like to introduce few commands that are frequently used in CA-7 environment and they are given in the table.

Difference between CA-7 and CA11

CA-7 is scheduler where as CA-11 is automatic re-run facility. It directs CA7, which step to start in case of ABEND. So no restart PARM is needed if you don't want to revert the updated done by prior steps. In that case, you may need to modify this parameter. I CA11 is there, you never get NOT CAT2 error. It performs dataset and GDG maintenance error. Using the file JEHF produced by CA11, we can get statistics of ABEND and RESTART details.

CA-11 inserts a step at the beginning of the job called CA07RMS. With this step, CA-11 scans the JCL being run and looks for any data sets being created in the job. If the first instance of a file in the job has a disposition beginning with "NEW", CA-11 then checks the system catalog to see if there is already an existing data set by that name. If there is, CA-11 then deletes it to prevent a NOT CAT2 error. THIS HAPPENED WHETHER IT IS A PRODUCTION RUN OF THE JOB, OR A RESTART.

Also, if necessary, GDG adjustments are made by CA-11 in the step. Any changes to bias number, (+1) to (0) on a restart, for example, are done by CA-11.

Difference between FORCE COMPLETE and CANCEL

If you CANCEL the job, it will be deleted from CA7 queue and if there is any job depends on this completion of this job, then it may not get triggered at all.
In such situations, you should go for FORCE-COMPLETION than CANCEL. If there is no dependency, then you can do anyone.

TSO CA7CLIST will take you CA7 panel. If you have access, then you can issue the following commands in that panel.

Command	**Purpose**
LJOB,JOB=xxxxx,LIST=ALL	Display all the details of the job xxxxx. LIST=TRIG gives only triggering and triggered jobs. LIST-RQMT gives only the requirement that should be satisfied for the job to run.
LSIT	List all the jobs that need restart, You frequently give this command to check for any ABENDS
LPRRN,JOB=xxxxx	Lists last execution details.
LJCL,JOB=xxxxx	List JCL for the JOB xxxxxxx
LDSN,DSN=xxx.yyyy,LIST=USERS	All jobs using xxx.yyyy dataset.
DEMANDH,JOB=xxxxx	Demand job into request queue. If you code DEMAND instead of DEMANDH then it will get into active queue directly. SET=NTR is used to avoid any triggering that may be done by this job.
LRLOG	Details of previous runs since midnight. Add date=* which gives the last five days worth. Can also find out which jobs ran late or were cancelled by using ST=LATE/CANC
FSTRUC,SCHDID=n,JOB=xxxxx	All jobs triggered by JOB xxxxx and all subsequent jobs triggered by those will be displayed in tree structure. Please note that there can be more than one schedule ID for each job and for each day.
SUBTM,JOB=Y*****	Allow you to make your job run at a particular time once already on the XQJ queue. Followed by TIME=hhmm
HELP	Given help on commands, syntax etc.
LSCHD,LSIT=CALS,JOB=xxxxx	It gives the calendar listing of what dates the job will run in the current year.

XQM:
Type XQM. You will get all the jobs, which are in request-ready-active and restart status. If you want to do any operation on the listed job, do it from this panel. It is vary user friendly. You no need to remember the commands.
Ex: CANCLE, Post-requirements etc.

Debugger- XPEDITOR

XPEDITOR/TSO is a testing, debugging and analysis tool that allows the programmer to view the source code as it executes in order to identify incorrect logic, test hard-to-validate routines and analyze both program structure and data flow. It is a CA product.

The program to be debugged in XPEDITOR environment has to be compiled with the JCL that has XPEDITOR steps. Check the installation configuration controller or the installation own utility for compiling a program with XPED option. The load library of the XPED compiled source must be added to the XPED load lib panel.

DDIO file should be allocated before the execution of the program. It is used to capture the source listing output from your XPEDITOR/TSO session. Now the program can be executed in XPEDITOR in one of the ways – TSO (Online) or BATCH (Interactive debugging, Un-attended debugging).

Once the program source listing is displayed, break points can be set up in any line by the line command **B**. **GO** command executes all the statements from the current statement of the program to the break point. After the break point, if you want, you can go step-by-step execution by **GO 1** command or alternatively again you can type GO command that takes you to next break point or END/ABEND of the program. **GT** line command is used for unconditional transfer of the execution to that line. **S** line command is used to skip the execution of particular statement. **GT** and **S** line commands have equivalent command line commands 'GT statement-no' and 'S statement-number'.

If you want to come out of execution of the program in between, use the command '**EXIT**'. '**RETEST**' command starts the test once again from the beginning. '**DELETE**' is used to delete all the XPED command (Ex: Break points).

K VAR-NAME in the command line displays the value of the variable. If you want to change the value of the variable before it is passed to next statement, it can be done. PEEK VAR-NAME can be also used for this purpose.

If you want to track the program flow when the variable got specific value in it OR when the value in the variable changes, then use the command '**WHEN VAR-NAME= 'value''** OR '**WHEN VAR-NAME CHANGES**'. '**WHEN DELETE**' command will reset the WNEM command issued before.

REVERSE command is used to review the execution path that led to current break point. To use REVERSE, MONITOR command should have been issued before. **MONITOR** command records the program execution in a buffer. After REVERSE command, GO 1 statement executes 1 statement in the reverse direction.

Command '**INTERCEPT** SUB-PROGRAM-NAME' just before the call statement, takes you to the first statement of sub-program (provided the sub-program is compiled with XPED option). Otherwise GO 1 command of main program (just before the execution of sub program) will take you to the next statement in the main program.

INTERCEPT, REVERCE, BREAK POINT, and DDIO files are imported from interview point of below.

Other Debuggers: COBTEST, INTERTEST.

JCL Interview:

1. Difference between JES2 and JES3
2. How can you specify temporary dataset in JCL?
3. Ways of passing data from JCL to program.
4. Space parameter and space ABENDS B37, D37 and E37.
5. How to solve system ABEND like SOC7?
6. GDG Defining parameter (LIMIT), Model dataset?
7. DISP parameters, What PASS will do?
8. Difference between in-stream and catalogued procedure?
9. Familiarly with SORT control cards.
10. Ways of overriding procedure.
11. What is symbolic parameter?
12. \STEPLIB JOBLIB JCLLIB purpose.
13. How to restart a job from a proc step.
14. Questions on condition codes.
15. COND, TIME and REGION can be coded in EXEC and JOB. Which overrides what?
16. Number of steps in JCL, nesting level of procedures, number of in-stream procedure in a job, number of DD statements in a step.

Some interesting questions, we used to ask in interviews:

<u>If I submit a job without any step, what return code will I get?</u>

You will get JCL error stating 'JOB HAS NO STEPS'.

<u>Will the following JOB step will give any error?</u>

```
//STEP1 EXEC PGM=TEST
MVSQUEST
//DD1 DD DSN=DATA.TEST,DISP=SHR
```

No. MVSQUEST will go as in-stream data to the program. If the program issues an ACCEPT, it can get MVSQUEST.

<u>I have 5 jobs in a PDS member. What will happen if I submit it? Which job will run first?</u>

5 jobs will be submitted. If all are of same job-name, then they will go one after other in FIFO basis. If their names are different, then depending on availability of initiator, multiple jobs can run concurrently.

<u>I have three jobs in my PDS. The successful execution of first job, should trigger the second one and the successful execution of the second job, should trigger third. How will I implement this scenario?</u>

In the last step of this job, include an IKJEFT01 step with SYSTSIN card SUBMIT PDS(JOB2). This step should have condition code in it, so that it will submit the next job only if all the prior steps properly ended. In the second job, include similar step that submit third job. Alternatively, you can use IEBGENER that use job2-pds-member as SYSUT1 and SYSUT2 as SYSOUT=(*,INTRDR).

<u>What is PSW?</u>

PSW is program status word. It is double word control register that is managed by both MVS and the hardware. Second word contains the next sequential instruction to be executed.

Can I write a dataset, which has a disposition of SHR?

Yes. But it is not advisable. Write should take exclusive access over the dataset before writing for maintaining integrity.

What are the ways, records can be appended to a file?

1. DISP as MOD and OPEN OUTPUT
2. DISP as OLD/SHR and OPEN EXTEND

How do you check an empty file in a batch stream?

1. IDCAMS – PRINT command with COUNT(1) will result return code 4, if the file is empty
2. IEBCOMPR comparison of two empty files would give non-zero return code. So check your file with a known empty file.

How do you check whether file is exist or not, in batch stream?

1. Do a LISTCAT on the file. If it is found, you will get return code = 16.

If the space parameter is SPACE=(TRKS,(20,20,10)), What you can say about this dataset?

This dataset need 20 tracks for primary allocation and 20 * 15 = 300 tracks secondary allocation (step-by-step). So it can get 320 tracks in the best case. In the worst case, it can get as low as 80 tracks. (Refer the extents section of the JCL). The dataset is a PDS and 10 directory blocks are allocated for this PDS. We can store 50-60 members in this dataset.

Do I need to mention SPACE for tape dataset?

No.

What the PERFORM parameter of the EXEC does?

It is used to specify the performance group of the job step. The performance group determines the rate at which the job steps in a program have access to system resources and it is helpful in optimizing system performance.

Can I recover a rolled out GDG version?

If it is not scratched.

When I am trying to edit a dataset, I get 'dataset-in-use' error message. My job is waiting for a dataset for a long time. How do I know who is using this dataset.

Way 1: TSO WHOHAS 'dataset-name', lists you the users/jobs using the dataset.
Way 2: ISPF 3.4, try to edit the dataset. You will get dataset-in-use message. As you can ask more detail in ISPF for any error message by pressing F1, press F1 again, ISPF lists you the users/jobs using the dataset.

What is the difference between PDS and PDSE?

No compression is needed for PDSE. When you delete a member, the system makes it space available for use –No directory space needs to be set aside for member names. Any PDSE can contain as many as 524286 members, assuming there is enough space to store them- PDSE is faster compared with PDS.

When PDS compressing gives me more space, why the same is not happening with PS?

In PS, data is overwritten every time. In PDS, every change creates a new version on SAVE command. The compression of PDS releases the space occupied by all prior versions and so you are getting more space. That is the reason why frequent save command are avoided in PDS.

Does it mean that I can retrieve the previous version of a member in PDS?

If you didn't compress your PDS, you can retrieve all the prior versions before the previous compression. But ISPF didn't provide options for this. You have to go for some third party product like STARTOOL (Serena).

I want to schedule my job tonight at 12:00 PM for some specific requirement (in test region). I don't want to stay late, just to submit a long running job. Can I schedule this job so that it can automatically submitted at 12:00 PM?

If you are using JES3, then you can try with DEADLINE command. But most of the installations have JES2. In JES2, distributed environment, such commands are not provided. But you can do it if you know the SLEEP function of OS. You can submit a job before you leave for the day, which invokes a REXX program in batch.

The REXX program can use SLEEP function and it can check the time for every 1 hour till 11 reaches and then for every 10 minutes till 11:50 reaches and afterwards for every 1 minute. And as soon as 12:00 PM is reached, using TSO SUBMIT command, it can submit the job. REXX can stay late instead of you!! (Check our REXX section for SLEEP command syntax).

VSAM (**V**irtual **S**torage **A**ccess **M**ethod)

History

Access Method is an interface between the application program and physical operation of storage device. It is a component of the operating system. VSAM is the first access method that efficiently uses the virtual storage of MVS. It can manipulate only the data that resides on a DASD. (Direct access storage device)

IBM introduced VSAM in 1973 as a replacement for several existing access methods designed by it earlier.

KSDS (Key Sequenced Data Set) replaced ISAM (Indexed Sequential Access Method)
RRDS (Relative Record Data Set) replaced BDAM (Basic Direct Access Method)
ESDS (Entry Sequence Data Set) provided same function as normal sequential QSAM (Queued Sequential Access Method).

Initially VSAM had only ESDS ad KSDS. RRDS and Alternate Index to KSDS are introduced in 1979. DF/EF (Data Facility Extended Function) VSAM was introduced in 1979 with Integrated Catalog Facility (ICF) to replace old VSAM catalog of the previous versions.

The latest version of DFP/VSAM released in 1991 called DFP/VSAM 3.3 contains enhancements like variable record length support for RRDS and added DFSMS facilities.

Advantages of VSAM over other access methods

1.Data retrieval will be faster because of an efficiently organized index. The index is small because it uses a key compression algorithm.
2.Insertion of records is easy due to embedded free space in the cluster.
3.Records can be physically deleted and the space used by them can be used for storing other records without reorganization.
4.VSAM datasets can be shared across the region and systems.
5.Datasets can be physically distributed over various volumes based on key ranges.
6.VSAM is independent of storage device type.
7.Information about VSAM datasets is certainly stored in VSAM catalog.
So reference of any VSAM dataset need not be detailed in JCL.

Disadvantages of VSAM

1.To allow easy manipulation of records, free space should be left in the dataset and this increases the space required.
2.Integrity of the dataset across the region and system need to be controlled by user.

CLUSTER

A cluster can be thought of as a logical dataset consisting of two separate physical dataset:

1. The data component (contains the actual data).
2. The index component (contains the actual index).

All types of VSAM datasets are called clusters even though KSDS is the only type that fulfills cluster concept. ESDS and RRDS don't have Index component.

Data Component

Control Interval and Control Areas

VSAM stores record in the data component of the Cluster in units called control intervals.

The control interval is the VSAM equivalent for a block and it is the unit of data that is actually transmitted when records are read or written. Thus many logical records from a control interval and many control intervals form a control area.

The Control Area (CA) is a fixed length unit of contiguous DASD storage containing a number of CI's. The control area is VSAM internal unit for allocating space within a cluster. The primary and secondary allocations consist of some number of CA. Minimum size of control area is 1 track and maximum size is 1 cylinder.

Format of Control Interval

There are four separate areas within a control interval.

1. Logical Record Area (LRA) that contains the records.
2. Free Space (FSPC). This area can be used for adding new records.
3. Unused Space. As FSPC may not be a multiple of record length, there will be always some unused space in every CI. This can be minimized for fixed length records by selecting proper control interval size.
4. Control Fields.
    - CIDF – Control Interval Definition Field – 4 bytes field containing information on free space availability in the control interval. One per control intervals.
    - RDF – Record Definition Field – 3 Bytes field. For fixed length record, there will be 2 RDF, first contains the number of records in the control interval and the second contains record length. For variable length records, the number of RDF can vary depending on how many adjacent records have the same length in the CI. If no two adjacent records are of the same length, then one RDF is needed to describe each record.

Index Component (Sequence Set and Index Set)

Besides the data component, VSAM creates an index component. Index component consists of index set and sequence set. The sequence set is the lowest level of the index is called the sequence set and it contains primary keys and pointers to the control intervals of the data component.

There is one sequence set for one control area. The highest record key of every control interval is stored in sequence set. The highest record key of the sequence set is stored in first level of index set. Based on the size of control interval of index component, there will be 1-3 levels of index sets in the index component of the dataset.

Control Interval and Control Area Split

When a VSAM dataset is created, the free space parameter defined can specify a percentage of control interval t be left free during the load of VSAM file. Later, when you add a record to the VSAM file, according to the key sequence, it is placed in a specific control interval.

But if the specific control interval is already full, then this cannot be placed in the sequence. The result is Control Interval split. It moves half of the records in the filled control interval to any other free control interval available in the control area and makes room for the new record in the right place.

If there is no free control interval exist in the control area, then half the control intervals are moved to new control area and this is called control area split.

Example

1. In the example below, there are four control areas and every control area contains two control intervals. Control fields are not shown in the diagram. There should be one sequence for every control area. So there are four sequence sets. There are two Levels of index set. The second level of index set contains pointers to sequence set.

2. Control Interval Split: When a record with a key 22 is added in the program, it should physically be stored between the existing records 21 and 23. 21 and 23 are in the first control interval of control area-3. There is no more space available to store this record. So control split will occurs. Records 20 and 21 continue to exist in the current control interval. Records 22 and 23 will be moved to any of the free control intervals. In our case control interval 2 is free. So they are moved there and index set is accordingly updated.

3. Control Area Split: When a record with key 3 in the program, it should be placed between 2 and 4. These records are first control interval of first control area and there is no free space. So control interval split is expected. But there is no free control interval in the control area-1. So control area split occur. New control area is allocated and half the records of control area-1 will be moved there and indexes are properly updated.

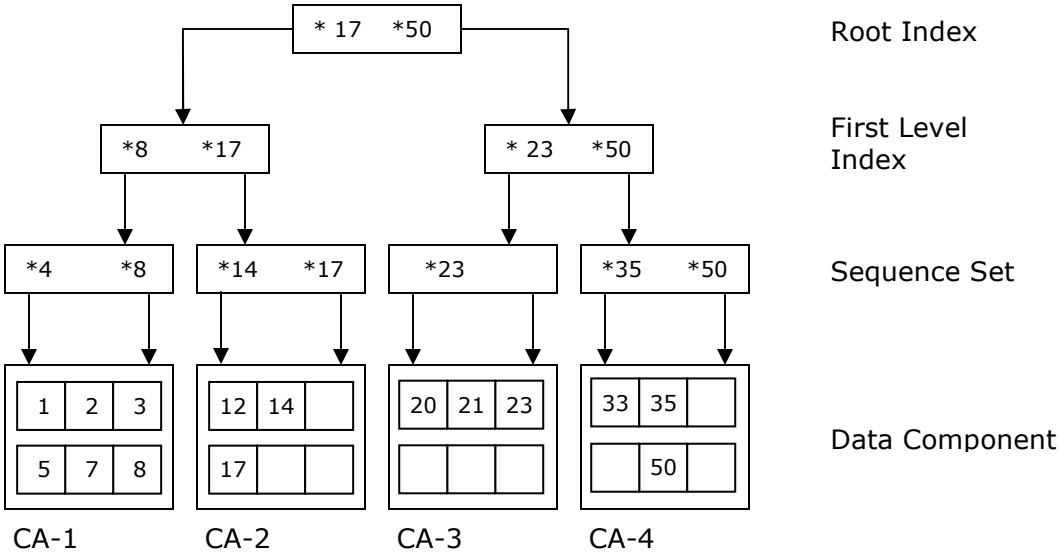
How index could make the access faster?

I faced an interesting question is an interview. The question follows:

My sequential file has 0 records. To get the 50^{th} record, I have to read 49 data record and bypass. Indexes store primary keys wit h actual location. So to read 50^{th} record using extend organization, I have to get the location of 50^{th} record from index. So I have to read and bypass the location of 49 records. The only difference is in the second case is I am doing sequential read in index set instead of dataset. How do you say my index access would be faster?

To answer this question, you should know how indexes are organized and read. We have already seen how they are organized. In the example below, to read the 50^{th} record, first root index is read and identified that the record should be in the right hand side. In the second I-O, I will get the sequence and in third I-O, I will get the location of the record. I will get my record in the fourth I-O instead 51 I-O. (50 I-O in index and 1 I-O for getting the data)

If I am accessing the first record, then sequential read needs only one I-O but obviously my random read needs more. So we prefer indexed organization only when the number of records is significant.



Properties of VSAM datasets

Properties of ESDS, KSDS and RRDS are listed a tabular format in the next page. Though the table says RRDS records should be of fixed length, MVS/DFP VSAM Version 3.3 allows variable length RRDS. VSAM internally implements it using KSDS>

VSAM LDS Properties: Linear datasets have no records. They are just long strings of bytes. Since it is long string of bytes an LDS has no FSPC, US, CIDF and RDF. The CISZ is always 4k bytes. The main use of LDS is to implement other data access organizations especially a relational database system like DB2.

As the data in an LDS is manipulated the operating system automatically pages in and out the portions of the dataset being worked on. The dataset is addressed by the RBA as if it were in memory and the system pages the needed pages in and out. Thus the process is very simple and fast.

ESDS differs from QSAM file in the following ways:

1. QSAM can be created over TAPE whereas ESDS cannot reside on TAPE.
2. Alternate Index can be created over ESDS. The alternate index has alternate key mapped with RBA. So Indexed and Random access are possible using the alternate Index in ESDS whereas it is not possible with QSAM.

Comparison of ESDS, KSDS and RRDS

Characteristics	ESDS	KSDS	RRDS
Entry Sequence	Based on entry sequence.	Based on collating sequence by key field.	Base on relative record number order.
Access	Only sequential access is possible.	Sequential and random access is possible. Random access is thru primary/alternate key.	Can be accessed directly using relative record number, which serves as address.
Alternate INDEX	May have one or more alternate indexes. But cannot be used in BATCH COBOL. Can be used in CICS COBOL.	May have one or more alternate indexes.	Not applicable.
Location of the record	A record RBA cannot be changed.	A record RBA can be changed.	A relative record number can be changed.
Free Space	Exist at the end of the dataset to add records.	Distributed free space for inserting records in between or change the length of existing record.	Free slots are available for adding records at their location.
Deletion	Cannot be deleted. REWRITE of same length is possible.	DELETE is possible.	DELETE is possible.
Record Size	Fixed or Variable.	Fixed or Variable.	Fixed.
SPANNED records	Possible.	Possible.	Not possible.
Specialty	Occupies less space.	Easy RANDOM access and most popular method.	Fastest access method.
Preference	Application that require sequential access only. Ex: PAYROLL PROCESSING.	Applications that require each record to have a key field and require both direct and sequential access. Ex: Banking Application.	Applications that require only direct access. There should be field in the record that can be easily mapped to RRN.

Integrated **D**ata **C**luster **A**ccess **M**ethod **S**ervices (IDCAMS)

AMS is used to perform various functions on VSAM datasets and catalogs. AMS has got a utility program called IDCAMS. The functions of AMS are performed using the different functional commands of IDCAMS. It can be invoked in the following three ways:

1.Batch mode with JCL statements.
2.Interactivelty with TSO commands.
3.Calls from an application program.

Important functional commands of IDCAMS are:

1. DEFINE - To create objects – KSDS/ESDS/RRDS/GDG/VSAMSPACE etc.
2. ALTER - To alter the parameter of the object already exists.
3. PRINT - To print and view the selected records of dataset.
4. DELETE - Delete the object.
5. LISTCAT - View the completer information about any object.
6. REPRO - Copy/Restore/Merge Utility for VSAM and NON-VSAM files.
7. EXPORT and IMPORT
   - Copy and restore datasets across the system.
8. VERIFY - To properly close the unclosed/ABENDED VSAM dataset.

Sample IDCAMS JCL

```
//JS10    EXEC PGM=IDCAMS,REGION=1024K,PARM=parameters
//STEPCAT   DD DSN=..,DISP=SHR  Optional STEPCAT
//SYSPRINT  DD SYSOUT=*        IDCAMS Messages
//SYSIN     DD *
  Control statements
/*
//
```

Guidelines for coding commands

1.At least one blank space must be there between the commands and the object and between sub parameter values.
2.A space is optional between a parameter and the parenthesis enclosing its values.
3.Sub-parameter values can be separated using a space or a comma.
4.Multiple parameters can be coded on a line but it is better to place each on a line by itself for better readability.
5A parameter and a sub-parameter cannot be separated with a hyphen. A plus sign should be used for this purpose.

Meaning of FILE and DATASET in AMS commands

1.Keyword FILE should point to DDNAME and it is logical. There should be mapping of the DDNAME with actual dataset. JES allocates the datasets associated with the file just before the execution of step.

2.Keyword 'DATASET' should directly point to the physical dataset and IDCAMS allocates this file dynamically for operation.

DEFINE CLUSTER
This command is used t create and name a VSAM Cluster.

Basic Parameters for Define Cluster

This parameter specifies name to the VSAM cluster. The cluster name becomes the dataset name in any JCL that invokes the cluster

// INPUT DD DSN=SMSXL86.TEST.VSAM,DISP=SHR
The name for a VSAM dataset can include up to 44 alphanumeric characters.

When the data and index parameter are coded to create the data and index components, the name parameter is coded for them as well. If the name parameter is omitted for the data and index VSAM tries to append part of .DATA or .INDEX as appropriate as the low level qualifier depending on how many characters the dataset name contains already and still staying within the 44 character limit.

If data and index components are named, parameter values can be applied separately. This gives performance advantages for large datasets.

DEFINE CLUSTER-DATA COMPONENT

The data parameter instructs IDCAMS that a separate data component is to be created. Data is optional but if coded must follow all parameter that apply to the cluster as a whole. There are several options for the data parameter but name is the most common.

DEFINE CLUSTER-INDEX COMPONENT

This index parameter creates a separate index component. Index is optional but if coded must follow all of the parameters that apply only to the data component. ESDS and RRDS must not have INDEX part.

DEFINE CLUSTER-SPACE Allocation

A VSAM dataset can have 123 extents in a VOLUME. Primary space is allocated initially when the dataset is created and based on request secondary space with be allocated. In the best case, you will get (primary + 122 * secondary). Refer the JCL section-SPACE parameter section for more detail on EXTENTS.

VSAM calculates the control area size internally. Control area can of one cylinder, the largest permitted by VSAM, usually yields the best performance. So it is always better to allocate space in cylinders because this ensures a CA size of one cylinder.

The RECORDS parameter is used to allocate space in units of records for small datasets. When this is done the RECORDSIZE parameter must be specified.
If allocation is specified in units of KILOBYTES or MEGABYTES VSAM reserves space on the minimum number of tracks it needs to satisfy the request.

Syntax: UNIT(primry secondary)
UNIT can be CYL/CYLINDERS TRK/TRACKS REC/RECORDSD
KB/KILOBYTES MB/MEGABYTES

DEFINE CLUSTER-VOLUMES Parameters

The volumes parameter assigns one or more storage volumes to the dataset. Multiple volumes can be specified and if so, they must be of the same device type. The data and index components can be stored in separate volumes to provide a performance advantage for large datasets.

VOLUMES(volser) OR VOLUMES(volser1 volser2)

DEFINE CLUSTER-RECORDSIZE Parameters (RECSZ)

The record size parameters specify VSAM what records to expect. The AVG and MAX are the average and maximum values for variable length records. If records are of fixed length AVG and MAX can be the same. Record size can be assigned at the cluster or data level.

Syntax: RECORDSIZE(AVG MAX)

DEFINE CLUSTER- KEYS Parameters

The keys parameters define the length and offset of the primary key in a KSDS record. The offset is the primary key's displacement in bytes from the beginning of the record. This parameter is defined for a KSDS only. Default is KEYS(64 0).

Syntax: KEYS(length offset)

DEFINE CLUSTER- Dataset Type Parameters

The dataset type parameter specifies whether the dataset is INDEXED(KSDS), NONINDEXED(ESDS), NUMBERED(RRDS) or LINEAR(LDS).

INDEXED(IXD) specifies a KSDS and it is the DEFAULT. When this parameter is specified VSAM automatically creates and catalogs an index (if other parameters like keys are valid). Indexed is also used for a variable length RRDS.

NONINDEXED(NIXD) when specified denotes an ESDS. No index is created and records are accessed sequentially or by their relative byte address (RBA).

NUMBERED (NUMD) specifies an RRDS and LINEAR specifies an LDS.

Performance Parameters For DEFINE CLUSTER

The performance parameters are coded in the DEFINE CLUSTER commands to

1.Reduce the amount of physical I-O.

2.Make necessary I-O faster.

3.Optimize disk utilization.

DEFINE CLUSTER-CONTROL INTERVAL SIZE Parameters

The size of the CI is specified by the CISZ parameter. The size of the CI is specified in bytes and it should be a multiple of 512 or 2048 depending on the type of catalog (ICF or VSAM) and the length of the record.

For datasets catalogued in ICF catalogs the control interval should be a multiple of 512 bytes with a range of 512 to 32768 bytes.

For datasets cataloged in VSAM catalogs, the data CISZ must be a multiple of 512 if records are of 8192 bytes or less, and a multiple of 2048 if records are of more than 8192 bytes. If a CISZ, which is not a multiple of the above two is assigned VSAM rounds the CISZ value up to the next highest multiple if necessary.

Best control interval size selection

For sequential processing of a KSDS, a relatively large CISZ will reduce physical I/O by keeping more records accessible in the buffers. On the other hand, for random processing of a KSDS a smaller CISZ would require lesser data transfer time and fewer buffers, thus making I/O faster.

For an ESDS (since it is processed sequentially) the CISZ should be relatively large depending on the size of the record.

Since an RRDS is processed randomly the CISZ should be relatively small.

The FREESPACE Parameter (FSPC)

The FREESPACE parameter is applicable to the KSDS and the variable-length RRDS only. FREESPACE cannot be assigned to an ESDS because all new records are added to the end of the dataset.

This parameter allocates some percentage of both the CI and CA for planned free space, which can be used for adding new records or expanding existing variable records. The parameter is coded as follows

FREESPACE(ci% ca%)
FREESPACE(ci%) control interval only
FREESPACE(0 ca%) control area only
FREESPACE(0 0) is the default
FREESPACE(100 100) means only one record will be loaded in every control interval and only one control interval will be loaded in every control area.

In order to effectively allocate FREESPACE the following factors have to be taken into consideration.

1.The expected rate of growth: If even growth is expected apply FREESPACE to both CI and CA. If uneven growth is expected apply the FREESPACE only to the CA.
2.The expected number of records to be deleted.
3.How often the dataset will be recognized with REPRO.
4.The performance requirements.

The CI FREESPACE allocation should be enough to cover the length of one record plus any additional RDFs that may result from a variable record length.

DEFINE CLUSTER-SPEED and RECOVERY Parameters

RECOVERY pre formats a VSAM dataset at the time of initial load or resume load. It takes longer time to load the dataset through RECOVERY than through SPEED which does not pre format the dataset. However if the load operation is aborted in the middle, RECOVERY enables resumption immediately after the last record loaded.

If SPEED is specified load will have to be restarted from the beginning. SPEED is highly recommended because it will speed up the initial data load.

The BUFFERSPACE Parameter (BUFSP)

By default, VSAM allocates two data buffers for all types of datasets. One data buffer for processing and reserves the second for potential split activity. In addition to this, it would allocate one index buffer for KSDS.

This parameter is used to override the default values. The BUFFERSPACE parameter represents the amount of storage in bytes required to process the contents of a minimum of one CI worth of data.

Syntax: BUFFERSPACE(bytes)

For sequential processing more number of data buffers need to be allocated. For random processing more index buffers may be required, at least one of each required in proportion to the radio of sequential and random processing planned.

The BUFSP value provided gets translated into data buffers for an ESDS or RRDS. But for a KSDS, VSAM decide on the number of data and index buffer based on the access method specified in the application program.

If BUFSP is specified in the DEFINE CLUSTER command, all application that use the dataset can use only this allocation unless they override it in the BUFSP sub parameter of the Access Method Parameter of JCL.

DEFINE CLUSTER-SPANNED Parameter (SPND)

The SPANNED parameter allows large records to span more than one CI. If maximum record length is more than the CI size specified, allocation will fail unless the SPANNED parameter is specified. However records cannot span across control areas. The resulting free space in the spanned CI is unusable by other records even if they fit logically in the unused bytes. NONSPANNED is the default.
It can be specified for ESDS and KSDS only.

DEFINE CLSUTER-The KEYRANGES Parameters (KYRNG)

The KEYRANGES parameter divides a large dataset among several volumes. The volumes in which the records are to be placed are specified using the VOLUMES parameter. Unless the KEYRANGES parameter is specified with the ORDERED parameter, the records are assigned randomly to the volumes. These parameters are illustrated by the following example

```
DEFINE CLUSTER
        (
        NAME(NTCI.V.UE4.W20000.T30.AV.DW200006)          -
        CYL(5 1)                                         -
        KEYS(8 0)                                        -
        RECSZ(80 80)                                     -
        KEYRANGES((00000001 29999999)                    -
                  (30000000 47000000)                    -
                  (47000001 99999999))                   -
        VOLUMES   (NTTSOB                                -
                   NTTSOJ                                -
                   NTTSO5)                               -
        ORDERED                                          -
        NOREUSE                                          -
        INDEXED                                          -
        _ _ _ more parameters
        )
```

When the ORDERED parameter is coded the number of volumes and KEYRANGES must be the same.

DEFINE CLUSTER-REUSE Parameter (RUS)

This specifies that a cluster can be opened again and again as a reusable cluster. It means, whenever you open the dataset in OUTPUT mode, all the records that are already exist in the dataset are logically deleted. NOREUSE (UNRUS) is the default and specifies the cluster as non-reusable.

A cluster cannot be reused if

1. KEYRANGES parameter is coded for it
2. An alternate index is built for it
3. The cluster has its own data space in both the VSAM and ICF catalog environments.

DEFINE CLUSTER-IMBED and REPLICATE Parameters (IMBD/REPL)

These parameters are applicable to a KSDS only. The IMBED parameter implies that the sequence set (lowermost level) of the index component of a KSDS will be placed on the first track of a data component CA and will be duplicated as many times as possible on that track.

What IMBED does for a sequence set REPLICATE does for an index set. The REPLICATE parameter forces each CI of the index set of the index component to be written on a separate track and replicated as many times as possible. This parameter reduces rotational delay when VSAM is accessing high-level index CI.

The IMBED option reduces the seek time it takes for the read-write head to move from the index to the data component and the replication factor reduces the rotational delay of the revolving disk.

NOIMBED (NIMBD) and NOREPLICATE (NREPL) are the defaults.

DEFINE CLUSTER-WRITECHECK Parameter (WCK)

This parameter instructs VSAM to invoke a DASD verify facility whenever a record is written. NOWRITECHECK (NWCK) is the default and provides no DASD verification. Since contemporary DASD devices are very reliable and because of the high performance overhead associated with this parameter it is better to accept the NOWRITECHECK option.

DEFINE CLUSTER-SHAREOPTION Parameter

This parameter specifies how a VSAM dataset can be shared across the regions and across the system. Default value is (1 3)

Syntax: SHAREOPTIONS(cross-region cross-system)

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 AAND 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.

DEFINE CLUSTER-ERASE Parameter

The ERASE parameter instructs VSAM to overwrite sensitive data with binary zeroes when the cluster is deleted. NOERASE is the default and it means that the deleted cluster is not to be overwritten with binary zeroes. ERASE can be coded at the cluster or data level.

When applied to the cluster level it refer only to the data component since the index component contains only keys and not data. ERASE can be specified with the ALTER and DELETE command. ERASE provided with DELETE overrides both DEFINE and ALTER specification.

Password Protection

VSAM dataset can be password protected at four different levels. Each level gives a different access capability to the dataset. The levels are

1. READPW- provides read only capability.
2. UPDATEPW- records may be may be read, updated, added or deleted at this level.
3. CONTROLPW- provides the programmer with the access capabilities of READPW and UPDATEPW.
4. MASTERPW- all the above operations plus the authority to delete the dataset is provided.

Passwords provided at the cluster level protect only if access requires using the cluster's name as dataset name. Therefore it is advisable to protect the data and index components using passwords because someone could otherwise access them by name. Another feature of MVS called Resources Access Control Facility (RACF) ignores VSAM passwords and imposes its own security and most VSAM datasets RACF security is sufficient.

The ATTEMPTS parameter coded with the password parameters specifies the number of attempts permitted for the operator to enter the password before abending the step.

The CODE parameter allows for the specification of a code to display to the operator in place of entry name prompt.

The AUTHORIZATION parameter provides for additional security by naming an assembler User Security Verification Routine (USVR). The sub parameter for this enclosed in parenthesis is the entry point of the routine.

TO and FOR Parameters

When a dataset is allocated by Access Method Services the TO and FOR parameters are two mutually exclusive parameters given to specify the retention period of the cluster being defined.

TO(YYDDD) or FOR(nnnn)

YYDDD is Julian data & nnnn can be 1-9999

About CATALOG parameter

Most of the AMS commands provide CATALOG option. If the command is DEFINE, then the dataset being defined will be placed in the catalog, mentioned in the CATALOG parameter. Similarly while accessing the dataset,

1. The dataset is first searched in the catalog mentioned in CATALOG parameter.
2. If CATALOG is not coded, then the dataset is searched into the catalog coded in STEPCAT or JOBCAT catalog.
3. If there is no STEPCAT and JOBCAT, then the dataset is searched in the user catalog corresponding to the high level qualifier of the dataset.
4. If there is no user catalog found, then the dataset is looked into system catalog.
5. If the dataset is not listed in system catalog also, then you will get the error message of dataset not found.

There will be one master catalog and n number of user catalogs in the system. Every user catalog should have an entry in master catalog.

KSDS-ESDS-RRDS-LDS Sample Definitions

KSDS Definition

```
//KSDSMAKE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
  DEFINE CLUSTER                    -
  (NAME(EMPLOYEE.KSDS.CLUSTER)      -
   VOLUMES(VSAM02)                  -
  CYLINDERS(2,1)                    -
  CONTROL INTERVAL SIZE(4096)       -
  FREESPACE(10,20)                  -
  KEYS(9,0)                         -
  RECORDSIZE(50,50))                -
  DATA                              -
  (NAME(EMPLOYEE.KSDS.DATA))        -
  INDEX                             -
  (NAME(EMPLOYEE.KSDS.INDEX))       -
  CONTROL INTERVAL SIZE(4096)       -
  CATALOG(VSAM.USERCAT.TWO)
/*
```

ESDS Definition

```
//ESDSMAKE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
  DEFINE CLUSTER                    -
  (NAME(EMPLOYEE.ESDS.CLUSTER)      -
   VOLUMES(VSAM02)                  -
  CYLINDERS(2,1)                    -
  CONTROL INTERVAL SIZE(4096)       -
  RECORDSIZE(50,50)                 -
  NONINDEXED)                       -
  DATA                              -
  (NAME(EMPLOYEE.ESDS.DATA))        -
  CATALOG(VSAM.USERCAT.TWO)
/*
```

RRDS Definition

```
//RRDSMAKE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
  DEFINE CLUSTER                    -
  (NAME(EMPLOYEE.RRDS.CLUSTER)      -
   VOLUMES(VSAM02)                  -
  CYLINDERS(2,1)                    -
  CONTROL INTERVAL SIZE(4096)       -
  RECORDSIZE(50,50)                 -
  NUMBERED)                         -
  DATA                              -
  (NAME(EMPLOYEE.RRDS.DATA))        -
  CATALOG(VSAM.USERCAT.TWO)
/*
```

LDS Definition

```
//LDSMAKE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
  DEFINE CLUSTER                    -
  (NAME(EMPLOYEE.LDS.CLUSTER)       -
   VOLUMES(VSAM02)                  -
  CYLINDERS(2,1)                    -
  LINEAR)                           -
  DATA                              -
  (NAME(EMPLOYEE.KSDS.DATA)         -
  DATA                              -
  (NAME(EMPLOYEE.LDS.DATA))         -
  CATALOG(VSAM.USERCAT.TWO)
/*
```

IDCAMS – REPRO Command

It is the general-purpose command that can operate on both VSAM and non-VSAM datasets. It performs three basic functions:

1. It loads an empty VSAM cluster with records. The data and index components for KSDS are built automatically.
2. It creates a backup of a VSAM dataset on a physical sequential dataset and later it can be used for restore/rebuild the VSAM dataset.
3. It merges data from two VSAM datasets.

Command Syntax

```
REPRO                                                    -
      INFILE(DDNAME) | INDATASET(DATASET-NAME)           -
      OUTFILE(DDNAME) | OUTDATASET(DATASET-NAME)         -
      Optional parameters
```

INFILE/INDATASET and OUTFILE/OUTDATASET point to input and output datasets.

When loading a KSDS using REPRO, the input data should be first sorted in ascending sequence by the field that will became the primary key in the output dataset. When loading an ESDS the sort step can be eliminated since the records are loaded in entry sequence. For an RRDS, the records are in relative record sequence starting with 1. The dataset should be sorted on the field that correlates to the relative record number.

REPRO Record Selection

REPRO can be used for selective copying.

Where to start REPRO	Where to stop REPRO	Valid For
FROMKEY(REC-KEY)	TOKEY(REC-KEY)	KSDS, ISAM
FROMADDRESS(RBA)	TOADDRESS(RBA)	KSDS, ESDS
FROMNUMBER(RRN)	TONUMBER(RRN)	RRDS
SKIP(number)	COUNT(number)	KSDS, ESDS, RRDS

SKIP refers to the number of input record to skip before beginning to copy and COUNT specifies the number of output records to copy.

The REUSE Parameter With REPRO

The REUSE option of REPRO will logically delete the existing records of a target KSDS, ESDS or RRDS and add new records from the source dataset as if the target dataset is empty. In other words REUSE sets the value of the ending RBA of the Highest Used control area in the data component (HURBA) to zero.

In order to use REUSE with REPRO it is a prerequisite that the target dataset must have been defined with the REUSE option in the DEFINE CLUSTER command.

Merging DATASETS Using REPRO and REPLACE Option

The REPRO command can be used for merging two datasets into one. The target dataset can be non-empty KSDS, ESDS or RRDS.

The input datasets can be any organization if the target dataset is a KSDS or ESDS. However if the target dataset is an RRDS the source dataset has to be an RRDS. KSDS to KSDS is the most common application of this merge technique.

If the REPLACE option is specified as part of the REPRO command then records with duplicate primary keys (for a KSDS) and duplicate relative record numbers (in the case of an RRDS) will be replaced.

IDCAMS – PRINT Command

It is used to print the contents of dataset. The output can be made available in various formats. This section deals with the PRINT command and its various mutually exclusive options.

```
PRINT INDATASET(data set-name) | INFILE(dd-name)
options
```

PRINT – CHAR/HEX/DUMP

This specifies the format in which to print.

CHARACTER (CHAR) prints as EBCDIC, 120 character per line. This format is mostly used for alphanumeric data. Any combination of bits that does not correspond to a printable character will be printed as period. If length of the record is greater than 120 characters, it is printed in blocks of 120 characters per line.

HEX format prints each character in the dataset as two hexadecimal digits. A maximum of 120 hexadecimal digits are printed on each line, an equivalent of 60 characters.

DUMP format is a combination of the character and hex formats. Each character is printed both as character and as the corresponding hex representation. Each print line consists of 64 hex digits and the 32 related characters. If the record length is greater than 32 characters the rest of the record is printed in blocks of 64 hex digits and 32 corresponding character per line.

PRINT – SKIP, COUNT, FROM and TO

The records to be printed can be selected in the same way records are selected in REPRO to COPY.

Where to start Printing	Where to stop Printing	Where Used
SKIP(number)	COUNT(number)	KSDS, ESDS, RRDS, non-VSAM
FROMKEY(key-value)	TOKEY(key-value)	KSDS, ALTERNATE INDEX
FROMADDRESS(RBA)	TOADDRESS(RBA)	KSDS, ESDS
FROMNUMBER(RRN)	TONUMBER(RRN)	RRDS

A command that prints records 29, 30 and 31 in character format

```
PRINT INDATASET(MM01.CUSTOMER.MASTER)   -
      CHARACTER                         -
      SKIP(28)                          -
      COUNT(3)
```

Empty file-check

If the file is empty, PRINT COUNT(1) ends with return-code as 4.

```
PRINT INDATASET(OUTPUT.DSN) CHARACTER COUNT(1)
```

IDCAMS – DELETE Command

The DELETE command can be used to delete both VSAM and non-VSAM object.

Syntax: DELETE ENTRY NAME OBJECT optional-parameters

OBJECT- CLUSTER GDG PATH AIX ALIAS CATALOG NONVSAM SPACE USERCATALOG

Options – ERASE/NOERASE PURGE/NOPURGE SCRATCH/NOSCRATCH FILE FORCE/NOFORCE

DELETE - ENTRY NAME

The name of the entry to delete is coded. GENERIC NAMES can be also given.

DELETE – ERASE (ERAS) / NOERASE (NERAS)

When the DELETE command is executed it does not physically delete a dataset. Only its ICF or VSAM catalog entry is removed making this space available for future use. The data component remains on the disk until another dataset is allocated on the same spot.

If the DELETE command is executed with the ERASE option, not only will the entry be removed from the catalog but also the data component will immediately be overwritten with binary zeroes. This option is coded for sensitive data files.

DELETE – PURGE (PRG) / NOPURGE (NPRG)

NOPURGE specifies that the entry not to be deleted if the retention period has not expired. (Default). So DELETE command doesn't delete the dataset before expiry date. PURGE parameter must be coded to delete the dataset before retention period.

DELETE – SCRATCH (SCR) / NOSCRATH (NSCR)

When a dataset is created, it will be listed in the catalog as well as the VTOC of the DASD. SCRATCH specifies that the dataset to be removed from VTOC as well as catalog. NOSCRATCH specifies that the dataset to be removed from the catalog alone.

DELETE – FORCE (FRC) / NOFORCE (NFRC)

It specifies whether objects that are not empty should be deleted.

FORCE allows you to delete data spaces, generation data group, and user catalog without first ensuring that these objects are empty.

NOFORCE causes the DELETE command to terminate when you request the deletion of a data space, generation data group, or catalog that is not empty.

DELETE – FILE(DDNAME)

It specifies the name of the DD statement that identifies.

1.The volume that contains a unique data set to be deleted.
2.The partitioned dataset from which a member (or members) is to be deleted.
3.The dataset to be deleted if ERASE is specified.
4.the volume that contains the data space to be deleted.
5.The catalog recovery volume(s) when the entry being deleted is in a recoverable catalog. If the volumes are of a different device type, concatenated DD statements must be used. The catalog recovery volume is the volume whose recovery space contains a copy of the entry being deleted.

IDCAMS – LISTCAT Command

The output of this command gives an insight into the inner functioning of VSAM. LISTCAT is used to view dataset attributes, password and security information, usage statistics, pace allocation information, creation and expiration dates and much more.

LISTCAT stands for LISTing a CATalog entry. It is useful for listing attributes and characteristics of all VSAM and non-VSAM objects catalogued is a VSAM or ICF catalog. Such objects can be catalogued itself, its aliases, the volumes it owns, clusters, alternate indexes, paths, GDG's, non-VSAM files etc. The listing also provides statistics about a VSAM object from the time of its allocation, namely the number of CI and CA splits, the number of I/O on index and data components, the number of records added, deleted and retrieved besides other useful information.

Syntax:
LISTCAT ENTRIES(OBJECT-NAME) ALL|ALLOCATION|VOLUME|HISTORY|NAME

Parameter	**Meaning**
NAME	List the name and type of the entry.
HISTORY	Lists reference information for the object including name, type of entry, creation and expiration date and the release of VSAM under which it was created.
VOLUME	Lists the device type and one or more volume serial number of the storage volumes where the dataset resides. HISTORY information is also listed.
ALLOCATION	Lists information that has been specified for space allocation including the unit (cylinders, tracks etc.), number of allocated units of primary and secondary space and actual extents. This is displayed only for data and index component entries. If ALLOCATION is specified VOLUME and HISTORY are included.
ALL	All the above details are listed.

LEVEL Parameter in LISTCAT

LISTCAT ENTRIES(SMSXL86.*) ALL

This will return complete information about all objects having two level of qualification and having the first qualifier SMSXL86. But if we require information about all objects having the high level qualifier SMSXL86, we have to use the LEVEL parameter. LISTCAT LEVEL(SMSXL86.*) ALL

Example

```
//SMSXL681 JOB (36512),'MUTHU',NOTIFY=&SYSUID
//LISTCAT    EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//SYSIN       DD *
 LISTCAT ENTRIES(SMSXL86.PAYROLL.MASTER)   -
          GDG                              -
          ALL
/*
```

IDCAMS – EXPORT an IMPORT commands

The EXPORT / IMPORT commands can be used for

1. Backup and Recovery
2. Exporting a dataset, an alternate index or a catalog to a different (but compatible) system.

Advantage over REPRO

1. Catalog information is exported along with the data.
2. Cluster deletion and redefinition are not necessary during the import step because input dataset already contains catalog information.
3. Also since the dataset contains catalog information it can be easily ported to other systems. An exported dataset has cross-system portability.

But EXPORT/IMPORT can be used with VSAM datasets only and the dataset created by EXPORT can have only a sequential organization. The dataset created by the EXPORT step cannot be processed until it has gone through a corresponding IMPORT step. Thus performing an EXPORT/IMPORT takes up more time.

Syntax: EXPORT entry-name OUTFILE(ddname) / OUTDATASET(dsname) –
Optional parameters

Entry-name is the name of the object that needs to be exported. OUTFILE mention exported into what name.

EXPORT – INHIBITSOURCE|NOINHIBITSOURCE

It specifies whether the original data records (the data records of the source cluster or alternate index) can be accessed for any operation other than retrieval after a copy is exported. This expectation can later be altered through the ALTER command.

INHIBITSOURCE (INHS) – cannot be accessed for any operation other than retrieval.
NOINHIBITSOURCE – original data records in the original system can be accessed for any kind of operation.

EXPORT – INHIBITTARGET (INHT) / NOINHIBITTARGET (NINHT)

It specifies how the data records copied into the target alternate index or cluster can be accessed after they have been imported to another system. The ALTER command is used to alter this parameter.
INHIBITTARGET – cannot be accessed for any operation other than retrieval.
NOINHIBITTARGET – Target object can be accessed for any type of operation after it has been imported into another system.

EXPORT – TEMPORARY|PERMANENT

It specifies whether the cluster or alternate index to be exported is to be deleted from the original system.

TEMPORARY specifies that the cluster or alternate index is not to be deleted from the original system. The object in the original system is marked as temporary to indicate that another copy exists and that the original copy can be replaced.

PERMENANT specifies that the cluster or alternate index is to be deleted from the original system. Its storage space is freed. If its retention period has not yet expired, you must also code PURGE. PERMENANT is the default.

EXPORT – ERASE|NOERASE

This specifies whether the data component of the cluster or alternate index to be exported is to be erased or not (overwritten with binary zeros).

With ERASE specification, the data component is overwritten with binary zeros when the cluster or alternate index is deleted.

With NOERASE specification, the data component is not overwritten with binary zeros when the cluster or alternate index is deleted.

Example:

```
//EXPORT EXEC PGM=IDCAMS
//DD2     DD DSN=SMSXL86.LIB.KSDS.BACKUP(+1),
//              DISP=(NEW,CATLG,DELETE),UNIT=TAPE,
//              VOL=SER=121212,LABEL=(1,SL),
//              DCB=(RECFM=FB,LRECL=80)
//SYSIN    DD *
  EXPORT A2000.LIB.KSDS.CLUSTER      -
  OUTFILE(DD2)
/*
```

IDCAMS – IMPORT Command

The IMPORT command restores the dataset from a backup copy created by the EXPORT command. The INFILE parameter specifies the DDNAME that refers to the backup copy created by the EXPORT command.

```
IMPORT INFILE(ddname)/INDATASET(dsname)          -
          OUTFILE(ddname) / OUTDATASET(dsname)    -
           Optional parameters
```

IMPORT – INTOEMPTY (IEMPTY)

When this parameter is specified, a previously exported dataset can be imported into an empty previously defined cluster. If this parameter is not specified an attempt to import into an empty dataset will fail.

IMPORT – OBJECTS

When the OBJECTS parameter is coded attributes of the new target dataset can be changed. These attributes include VOLUMES and KEYRANGES.

IDCAMS – VERIFY Command

If a job terminates abnormally and a VSAM dataset is not closed, the catalog entry for the dataset is flagged to indicate that the dataset may be corrupt. Before the dataset can be opened again, the VERIFY command n=must be used to correctly identify the end of the dataset and reset the catalog entry.

Model syntax for the VERIFY command:

```
VERIFY {FILE(ddname[/password]) | DATASET(entryname[/password])}
```

Base cluster and alternate index can be verified. The verification of base cluster does not verify its alternate indexes so each one of them must be treated separately.

IDCAMS – ALTER Command

The ALTER command is used to change many of the attribute of an existing VSAM dataset. It is easier doing it this way than to delete the old dataset and to redefine it with the new attributes. If the latter method is used the data will have to be backed up before deletion to preserve the records in it and later restored after reallocation.

It is important to note that all the attributes cannot be changed with ALTER. For example, you cannot alter the record length or keys of a VSAM dataset if it already contains some records. The only way in this scenario is Delete and Define.

Syntax: ALTER entry-name parameters

The attributes that can be altered are below:

NAME, ADDVOLUMES, REMOVEVOLUMES, BUFFERSPACE,
READPW, CONTROLPW, UPDATEPW, MASTERPW, ATTEMPTS
ERASE|NOERASE, SCRATCH|NOSCRATCH, EXCEPTIONEXIT, FREESPACE
OWNER, SHAREOPTIONS, TO(date)|FOR(days), INHIBIT|NOINHIBIT, NULLIFY,
UNIQUEKEY|NOUNIQUEKEY, UPDATE|NOUPDATE, UPGRADE|NOUPGRADE,
AUTHORIZATION, CODE, WRITECHECK|NOWRITECHECK, LIMIT

We frequently use this utility for renaming (NEWNAME), altering free space (FREESPACE), temporarily disabling updates over dataset (INHIBIT) and increase the generation limit of GDG (LIMIT).

Example

JCL for an AMS job with components that runs an ALTER command

```
//SMSXL861 JOB (36512),'MUTHU',NOTIFY=&SYSUID
//NEWNAME EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//SYSIN      DD *
  ALTER MM01.CUSTOMER.MASTER  -
      NEWNAME(MM01.CUSTMAST)    /* CHANGE NAME */
      FREESPACE(10 10)              /* CHANGE FREE SPACE ALLOCATION */
/*
```

SYSIN for adding candidate Volumes

```
        ALTER                           -
        EMPLOYEE.KSDS.DATA              -
        ADDVOLUMES(VSAM03)              -
```

IDCAMS – MODAL COMMANDS

Functional commands are used to perform functions. Modal commands are used to test and SET the condition codes returned by the functional commands.

LASTCC contains the return-code of the last executed functional command and if more than one functional command is executed in single IDCAMS step, then MAXCC contains the maximum value of the condition codes from the previously executed functional commands. MAX CC is returned as step return code in JCL.

SET command is used to reset the MAXCC or LASTCC within AMS.

IF-THEN-ELSE statements are used to control the command execution sequence.

In the below example, REPRO command loads data from a sequential dataset on to a KSDS. Only if the condition code of the REPRO step is zero, the next LISTCAT step will be executed. Otherwise the KSDS will be deleted. MAXCC is set to zero at the end to avoid non-zero return code.

```
//SYSIN       DD *
REPRO                                    -
       INDATASET(SMSXL86.DATA.TEST)      -
       OUTDATASET(SMSXL86.DATA.KSDS)     -
IF LASTCC=0                              -
THEN                                     -
LISTCAT                                  -
ENTRIES(SMSXL86.DATA.KSDS) ALL           -
ELSE                                     -
DELETE SMSXL86.DATA.KSDS                 -
END-IF                                   -
SET MAXCC=0
/*
```

IDCAMS Step Return Codes and Meaning

Error Code	**Severity Of Error**	**Meaning**
0	No Error	Functional command completed its processing successfully.
4	Minor Error	Processing is able to continue, but a minor error occurred, causing a warning message to be issue.
8	Major Error	Processing is able to continue, but a more severe error occurred, causing major command specifications to be bypassed.
12	Logical Error	Generally, inconsistent parameters are specified, causing the entire command to be bypassed.
16	Severe Error	An error of such severity occurred that not only can the command causing the error not be completed; the entire AMS command stream is flushed.

ALTERNATE INDEX

Need for AIX

1. KSDS dataset is currently accessed using a primary key. We want to access the dataset RANDOMLY with another key.
2. We want to access ESDS file RANDOMLY based on key.

Steps Involved

Step 1: DEFINE AIX

DEFINE ALTERNATEINDEX command defines an alternate index. Important parameters of these commands are:

Parameters	**Meaning**
RELATE	Relates AIX with base cluster
NONUNIQUE/ UNIQUE	Duplicates are allowed / not allowed in alternate key.
KEYS	Defines the length and offset of alternate key in base cluster.
UPGRADE	Adds the AIX cluster to the upgrade set of base cluster. Whenever base is modified, its upgrade set is also modified. UPGRADE is default. NOUPGRADE didn't add the AIX to base cluster upgrade set.
RECORDSIZE	Specifies the record size of alternate index record. It is calculated using the formula in the next table.

AIX Record Size Calculation:

	UNIQUE Alternate Key	**NONUNIQUE Alternate Key with maximum 'n' duplicates**
KSDS	5 + Alternate Key Length + Primary Key length	5 + Alternate Key Length + (n * Primary Key length)
ESDS	5 + Alternate Key Length + RBA size (RBA size = 4 bytes)	5 + Alternate Key Length + (n * RBA size) (RBA size = 4 bytes)

Five Byte Control Information of Alternate Index Record:

Byte-1	Type of Cluster; X'00' indicates ESDS and X'01' indicates KSDS.
Byte-2	Length of base cluster pointers in alternate index; Primary key length for KSDS and X'04' for ESDS
Byte-3 Byte-4	Half word binary indicates number of occurrences of primary key pointers in alternate index record. X'0001' for unique alternate key.
Byte-5	Length of alternate key.

Step 2: BLDINDEX

Alternate index should have all the alternate keys with their corresponding primary key pointers. After the AIX is defined, this information should be loaded from base cluster. Then only we can access the records using the AIX. BLDINDEX do this LOAD operation.

Important parameters of this command are:

1. INFILE and OUTFILE points to Base Cluster and Alternate Index Cluster.

2. INTERNALSORT, EXTERNALSORT, WORKFILES:

Loaded AIX should have sorted alternate record key with base cluster keys. So an intermediate SORT is required in between reading the base cluster and loading the AIX. We should allocate work datasets IDCUT1 and IDCUT2 for this SORT.

If INTERNALSORT parameter is coded, then AMS tried to do internal sort if enough memory is available and it would be fast. If enough space is not available, then if you have coded IDCUT1 and IDCUT2, it uses that space and does an external sort.

If you code EXTERNALSORT parameter, then AMS won't try for internal sort.

If you want to give your own DDNAME files for sorting instead of IDCUT1 and IDCUT2, inform it to ASM by WORKFILE (DDNAME1, DDNAME2) parameter. Here, DDNAME1 and DDNAME2 are allocated instead of IDCUT1 and IDCUT2.

Step 3: DEFINE PATH

The DEFINE PATH command establishes a bridge between the Base Cluster and an Alternate Index. It does not occupy any space. It is just a catalog entry, which establishes a link between an alternate index and base cluster. Important parameters of this command are:

1. PATHENTRY command relates PATH with AIX.
2. When you open a PATH, respective base cluster will be automatically opened. UPDATE opens the UPGRADE set of the base cluster also and it is default. NOUPDATE opens only the base cluster and not it's UPGRADE set.

In BATCH, if you want to access the base cluster with alternate key, you should allocate the PATH in the JCL. The name of the DDNAME for the PATH in JCL is DDNAME of the base cluster suffixed with 1 for first alternate key and n for nth alternate key. Refer the COBOL material for more information.

In CICS, if you want to access base cluster with alternate key , then you should register the PATH as FCT entry.

Example:

1. Command that Defines an Alternate Index

```
DEFINE AIX (NAME(MMA2.EMPMAST.SSN.AIX)                  -
            RELATE(MMA2..EMPMAST)                       -
            KEYS(9 12)                                  -
            UNIQUEKEY                                   -
            UPGRADE                                     -
            REUSE                                       -
            VOLUMES(MPS800))                            -
   DATA     (NAME(MMA2.EMPMAST.SSN.AIX.DATA)            -
            CYLINDERS(1 1))                             -
   INDEX    (NAME(MMA2.EMPMAST.SSN.AIX.INDEX))
```

2. Command that Defines PATH

```
DEFINE PATH (NAME((MMA2.EMPMAST.SSN.PATH)               -
            PATHENTRY((MMA2.EMPMAST.SSN.AIX)            -
            UPDATE)
```

3. A Job that Builds an Alternate Index

```
//MM01LC3 JOB  (36512),'MUTHU',NOTIFY=&SYSUID
//BLDINDX  EXEC PGM-IDCAMS
//SYSPRINT DD  SYSOUT=*
//IDCUT1   DD  UNIT=SYSDA,VOL=SER=MPS800,DISP=OLD
//IDCUT2   DD  UNIT=SYSDA,VOL=SER=MPS800,DISP=OLD
//SYSIN    DD   *
  BLDINDEX   INDATASET(MMA2.EMPMAST)              -
             OUTDATASET(MMA2.EMPMAST.SSN.AIX)     -
             CATALOG(MMA2)
/*
```

The order of build index and definition of PATH does not matter.

AMP Parameter in JCL

The Access Method Parameter (AMP) completes information in an Access method Control Block (ACB) for a VSAM dataset.

The ACB can be coded for a Key-Sequenced (KSDS), Entry-Sequenced (ESDS) or Relative Record (RRDS) VSAM dataset. AMP is only supported for VSAM datasets. AMP is most often used to allocate I/O buffers for the index and data components for optimizing performance.

Syntax: AMP=(sub-parameter, sub parameter,)
Or
AMP=('sub parameter, sub parameter,')
Single apostrophes are required if the sub parameter(s) contains special characters.

AMORG

This parameter, which stands for Access Method ORGanization, indicates that the particular DD statement refers to a VSAM dataset.

BUFND

This parameter gives the number of I/O buffers needs to for the data component of the cluster. The size of each buffer is the size of the data CI.
The default value is two data buffers one of which is used only during CI/CA splits. Therefore the number of data buffers left for normal processing is one.
If more data buffers are allocated, then performance of sequential processing will improve significantly.

BUFNI

This parameter gives the number of I/O buffers needed for the index component for the cluster. Each buffer is the size of the index. The sub-parameter may be coded only for a KSDS because ESDS and RRDS do not have index components. The default value is one index buffer.

If more index buffers are allocated, then performance of random processing will improve significantly.

BUFSP

This parameter indicates the number of bytes for data and index component buffers. If this value is more than the value given in the BUFFERSPACE parameter of the DEFINE CLUSTER, it overrides the BUFFERSPACE. Otherwise BUFFERSPACE takes precedence. The value of BUFSP is calculated as

BUFSP = DATA CISIZE * BUFND + INDEX CISIZE * BUFNI

However it is recommended not to code this parameter ad let VSAM perform the calculation from the BUFND and BUFNI values instead.

Other parameters are TRACE, STRNO, RECFM, OPTCD, CROPS.

INTERVIEW Questions

1. Control Interview and Control Area Splits. *****
2. Need for Alternate Index. Steps involved in creating Alternate Index. *****
3. Explain the need of PATH. ****
4. Importance of UPGRADE and UPDATE. ***
5. File Status – 21 22 23 24 02 90 92 *****
6. How do we read KSDS using AIX in batch COBOL and CICS? ****
7. Alternate record size calculation. ***
8. Difference between DELETE PURGE ERASE FORCE. **
9. Understanding of KEYS parameter. If the key is from 10^{th} byte to 13^{th} byte, how will you code KEYS parameter? ***
10. IMBED and REPLICATE options DEFINE CLUSTER? ***
11. SPEED/RECOVERY of DEFINE CLUSTER? **
12. REUSE option of DEFINE CLUSTER? **
13. REPLACE option of REPRO? **
14. Selective copy options of REPRO **
15. How to rename a set of files in one shot? **
16. How do you choose control interval and control area size of your application? *****
17. How to access ESDS randomly in batch COBOL? ***
18. What is skip-sequential READ and how batch COBOL supports it? **
19. SHAREOPTION of DEFINE CLUSTER **
20. Familiarity with FILEAID for VSAM operations. **
21. FSPC and its importance. FSPC(0 0), FSPC(100 100) meaning. **
22. How do you check how many records are there in the VSAM cluster and the FSPC availability for the future insertion? **
23. What is VSAM reorganization and how frequently we need to do that? **
24. After the VSAM is created, first operation we should do is, initializing it with dummy record. True/False. If I am not initializing, what operation would give me problem? **
25. Advantages over VSAM over other Access Methods. **
26. VSAM to DB2 conversion strategy **
27. How do you choose BUFNI, BUFND parameter for better performance? **
28. Can I un-catalog or delete a dataset using DISP parameter of JCL? **
29. What is the LISTCAT level parameter will do? **
30. VSAM dataset is allocated with DISP=SHR option in a JCL step. Does it mean that the program access the file in READ only mode. ***
31. What are the ways you can get the attributes of VSAM file when you don't have VSAM DEFINE card with you? **
32. What are different types of VSAM and their properties? There may be lot of questions in the properties like can I delete ESDS record, Can I have AIX for RRDS etc. So read the properties table and understand thoroughly.

History

IBM's first database is IMS. This is a hierarchical database. IBM introduces its seconds base on relational concepts in 1980's & it in called as Database 2 (DB2).

Advantage of DBMS over File Management System

1. Increased Independency

   If a new field is added to a file, the layout of the file should be changed in all the programs with new field, though the programs do not use this field. But if a new column is added to a table, it does not need any program change, as long as the programs do not need this field. Thus programs using Databases are developed independent of physical structure.

2. Less Redundancy

   In file system, the same data is stored in multiple files to meet the requirements of different applications. In DB2, we can store the common information in one table & make all the application to share it. By means of centralized control & storing the fact at all right place & only once, data redundancy is reduced.

3. Increased Concurrency & Integrity

   Concurrency allows more than one task to share a resource concurrently. Integrity is defined as validity or correctness of data. In file system, if a file is allocated with OLD disposition, then no other program can use it. DB2 allows more than one task to use the same table in update mode. At the same time, integrity of data is ensured by its sophisticated locking strategy.

4. Improved Security

   In file system, all the records in the file are subjected to a specific access right. The access cannot be controlled at field level or set level. But in DB2, we can restrict access on column level & set level.

Types of Database

Hierarchical: Relation between entities it's established using parent-child relationship- Inverted Tree Structure. This is first logical model of DBMS. Data stored in the form of SEGMENTS. Example: IMS.

Network: Any entity can be associated with any other entity. So distinguishing between parent & child is not possible. This is linked structure model. Data stored in RECORD & different record types are linked by SETS. Example: IDMS.

Relational: In mathematical terms, relation is TABLE. Data is stored in the form of tables consists of Tuples (rows) & attributes (columns). Example: DB2

DB2 Objects

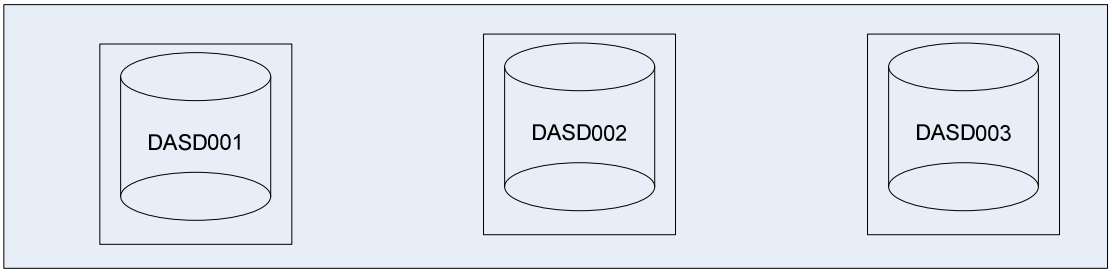
There are two types of elements or objects in DB2.

System Objects: Objects that are controlled & used by DB2.
Example: DB2 Directory, DB2 catalog, Active & Archive logs, Boot Strap Dataset (BSDS), Buffer Pools, Catalog visibility Database & Locks.

Data Objects: Objects that are created & used by the users.
Example: Tables, Indexes, Views, Table spaces, Databases, Storage Groups.

DB2 Object-Storage Group

Storage group is a collection of direct access volumes, all of the same type. DB2 objects are allocated in Storage group. A storage group can have maximum 133 volumes.

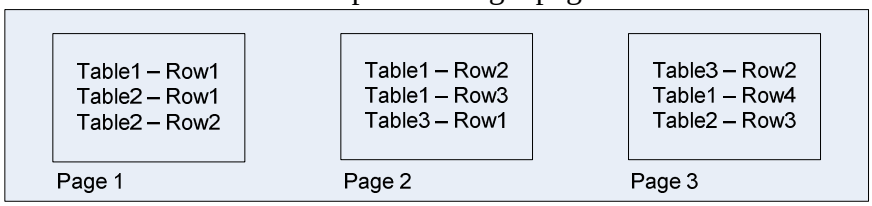


DB2 Object-Tables Space

Table spaces are the physical space where the tables are stored. There are three types of table spaces. There are three types of table spaces.

1. Simple Tables Space:

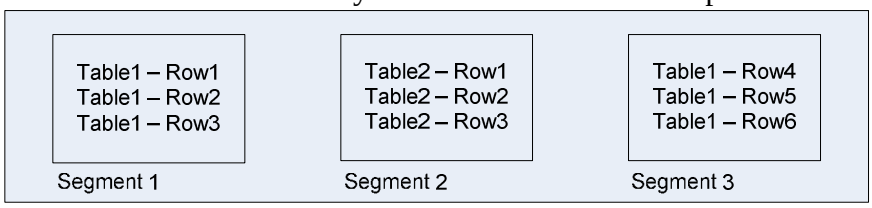
More than one table is stored in the table space & single page can contain rows from all the tables.



2. Segmented Table Space:

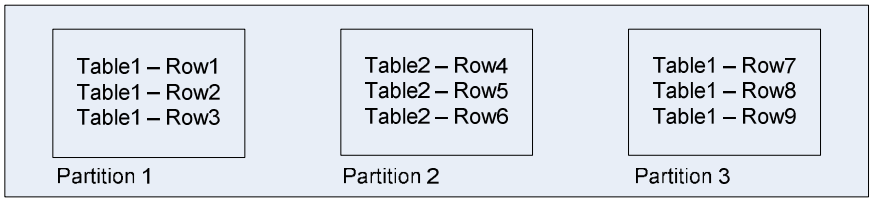
Segment is logically contiguous set of n pages where n is defined by SEGSIZE parameter of TABLESPACE definition. Tables are stored in one or more segments. Mass delete & sequential access are faster in Segmented type table space.

Mass delete & sequential access area faster in segmented type table space. Reorganization will restore the table in its clustered order. Lock table command locks only the table & not the table space.



3. Partitioned Table Space:

Only one table can be stored. 1-64 partitions are possible in table space. NUMPART of TABLESPACE definition decides the partitions. It is partitions with value ranges for single or combination of columns & there columns cannot be updated. Individual partitions can be independently recovered & reorganized. Different partitions can be stored in different groups.



DB2 Object Database

It is not a physical object. It is logical grouping consist of table spaces, Index spaces, Views, Tables etc. whenever you create an object you have to explicitly say which database it belongs to or DB2 implicitly assigns the object to the right database.

In the creation of table, we should say in which database & in which table space we are housing the table (Explicitly). In creating an index, we mention only the table name. DB2 decides as a unit of start & stop. There can be a maximum of 65,278 databases in DB2 subsystem.

DB2 Object – Table

Table is a structure consists of number of columns & rows. It is created using the DDL CREATE TABLE.

Physical Hierarchy.

1. ROWID is a pointer to data record in the page.
2. It is a 4 byte field. (3 bytes for page & 1 byte for line in the page)
3. Page consists of 1 to 127 data records. It is the unit of I-O.
4. Table space consists of pages & it is stored in STOGROUP.
5. Database consists on one or more table space & other related objects.

DB2 Object - Index

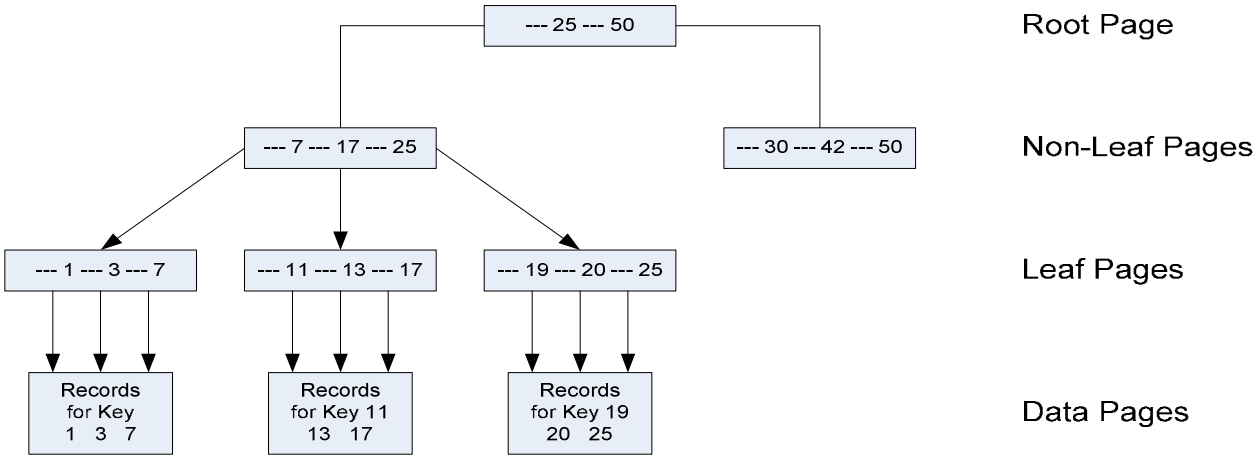
Generally, Index is used to access a record quickly. There are two types of Indexes: Clustering & Non-Clustering. If there is a Clustered Index is available for a table, then the data rows are stored on the same page where the other records with similar index Keys are already stored on. There can be only one cluster index for a table. Refer the diagram in the next page for better understanding of clustered & non-clustered indexes.

Understanding: Root Page, Leaf Page & Non-Leaf Page

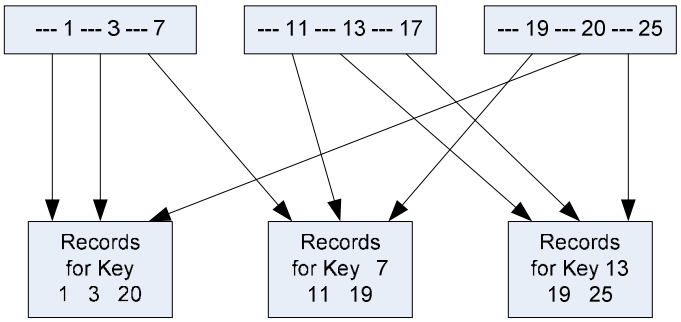
Referring to VSAM basics, sequence set has direct pointers to control interval & last level indexes have pointers to sequence set & higher-level indexes have pointers to lower level indexes & there is one root index where the B-tree starts.

Similarly Leaf-pages contain full keys with row-ids that points to data page. (Row Id has page number & line number). Non leaf pages have pointers to leaf pages or lower level non-leaf pages & the root pages has pointer to first level non-leaf pages.

View of Clustered Index – Diagram



View of Non-Clustered Index – Diagram(non-leaf pages & data pages are shown)



DB2 Object – Index Space

Unlike table spaces, Index spaces are automatically created when indexes are created. Indexes are stored in index spaces.

DB2 Object – Alias & Synonym

Alias & Synonym are alternate name for a table. The main differences between them are listed below:

SYNONYM	ALIAS
Private object. Only the user who created it, can access it.	Global object. Accessible by anyone.
Used in local environment to hide the high level qualifier.	Used in distributed environment to hide the location qualifier.
When the base table is dropped, associated synonyms are automatically dropped.	When base table is dropped, aliases are not dropped.
SYSADM authority is not needed	To create an alias, we need SYSADM authority or CREATELIAS privilege.

DB2 Object- View

Views provide an alternative way of looking at the data in one or more times. A view is a named specification of a result table. For retrieval, all views can be used like base tables. Maximum 15 base tables used in a view.

Advantages of View

1. Data Security: Views allows to set-up different security levels for the same base table. Sensitive columns can be secured from the unauthorized Ids.
2. It can be used to present additional information like derived columns.
3. It can be used to hide complex queries. Developer can create a view that results from a complex join on multiple tables. But the user can simply query on this view as if it is a separate base table, without knowing the complexity behind the building.
4. It can be used for domain support. Domain identifies a valid range of values that a column can contain. This is achieved in VIEW using WITH CHECK OPTION.

Read Only Views.

Views on which INSERT, UPDATE & DELETE operations cannot be carried out are called non-updateable views or read only views.

Characteristics of Updateable views

1. It Should be derived from one base table & should not contain derived columns,
2. Views should not contain GROUP BY, HAVING & DISTINCT clauses at the outermost level.
3. Views should not be defined over read-only views.
4. View should include all the NOT NULL columns of the base table.

DB2 Object – Catalog

It is a data dictionary for DB2, supporting & maintaining data about the DB2 environment. It consists of 54 tables in the system database DSNDB06. The data in DB2, about the data in the tables are updated when RUNSTATS utility runs. The information on the DB2 catalog table can be accessed using SQL.

DB2 Object – Directory.

This is second data dictionary of DB2 & used for storing detailed, technical information about the aspects of DB2's operation. It is for DB2 internal use only. It cannot be accessed using SQL statements. It is stored in the system database DSNDB01.

DB2 Object – Active & Archive Logs.

DB2 records all the data changes & significant events in active logs as & when they occur. In case of failure, it uses this data to recover the lost information. When active log is full, DB2 copies the content of active log to a DASD or magnetic tape data set called archive log.

DB2 Object – Boot Strap Dataset

It stores the inventory of all active & all archive logs. During installation of DB2, two BSDS are created & kept in different volumes. BSDS datasets are VSAM KSDS.

DB2 Object – Buffer Pools

Buffer pools are virtual storage area, which DB2 uses for its temporary purpose. There are fifty 4K buffer pools & ten 32 buffer pools in DB2. DB2 default buffers are BP0, BP1, BP2 & BP32.

Program Preparation

SQL (Structured Query Language) is the language that is used to Retrieve/update/delete DB2 data. SQL statements are embedded into COBOL program within the scope of 'EXEC SQL' & END-EXEC'.

DB2 program is first feed to DB2 Pre-compiler that extracts the DB2 statements into DBRM & replaces the source program DB2 statements with COBOL CALL statements. This modified source is passed to COBOL compiler & then link editor to generate load module. During pre compilation, time stamp token is place on modified source & DBRM. This m numbers of package are then feed to BIND PLAN step that produces PLAN. Package is not executable but plan is executable. To run a DB2 program, plan is mandatory.

Plan & Package

Binding process can happen in two stages, BIND PACKAGE & BIND PLAN. One DBRM is created for one program. If the main program calls n number of Sub-programs, then there will be n DBRMS in additions to main program DBRM.

These n+1 DBRM can be directly feed to BIND PLAN to produce a single PLAN or create m number of intermediate packages each formed by one or more DBRM. This m numbers of packages are then feed to BLIND PLAN step that produces PLAN. Package is not executable but Plan is executable. To run a DB2 program, Plan is mandatory.

Advantages of Package:

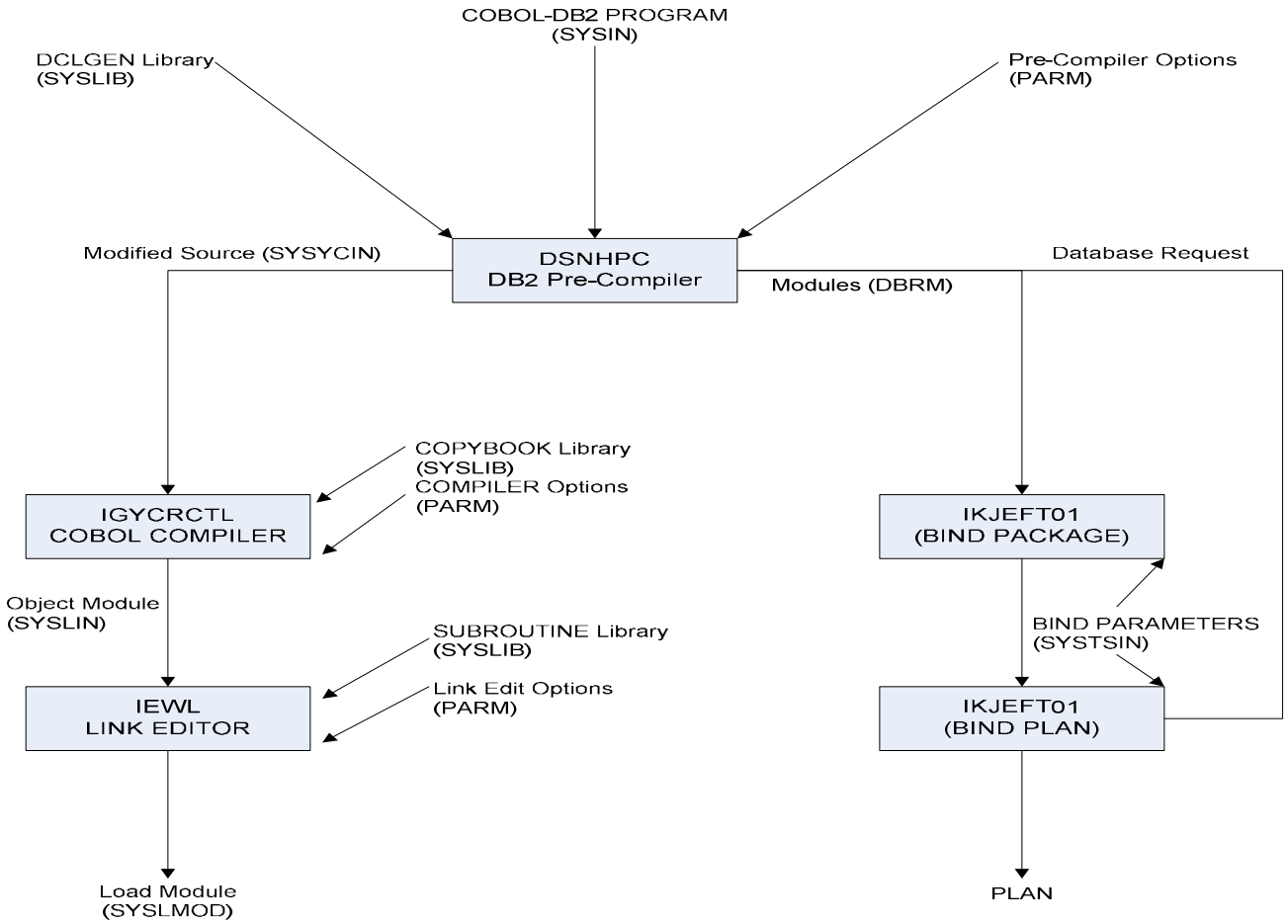
1. When there is a change in sub program, it is enough to recompile the subprogram & create the PACAGE for that sub program. There is no need for BIND PLAN.
2. Lck options & various bind options can be controlled at sub-program level.
3. It avoids the cost of large bind.
4. It reduces Rebound time.
5. Versioning: same package can be bounded with multiple programs with different collection IDs.

Execution of DB2 Program

DB2 Program is executed using terminal monitor program IKJEFT01. it can be executed using IKJEFT1A or IKJEFT1B also. They are alternate entry points of IKJEFT01. Db2 region name, program to run, Db2 plan & PARM(if needed) are provided in the SYSTSIN card of IJEFT01.

```
//STEP EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT = *
//SYSTSIN DD *
  DSN SYSTEM(DST2)                          ➔ In DB2 test region DST2.
  RUN PROGRAM(pgmname) PLAN(planname)       ➔ Execute the program
  LIN('loadlibrary')                        ➔ Whose load module is here
  END
/*
```

Program Preparation in Details



Time Stamp token is passed from pre-compilation stage to Load module & Plan,

When first time the plan is accessed, the time stamp of plan is verified against the time stamp of load module. If they match, then table will be accessed using the access path n PLAN & proceed further. If they don't match, then the program abnormally ends with SQLCODE of -818. It means the load module & plan are out of sync. It usually occurs if we modify the program & pre-compile, compile & link edit but did not bind the plan with latest DBRM produced in pre-compilation stage.

There is a modification in COBOL part. There is no change in any of the DB2 statements in the program. Is binding necessary?

YES. It is necessary. For COBOL changes, we have to compile & link edit. For compilation we need pre-compiled source. So new times stamp is transferred to your load module from the pre-compilation stage. If we don't BIND it, timestamp mismatch is obvious & the program will abnormally end.

Can this time stamp check be avoided?

If the program pre-compiled with LEVEL option, then there won't be time stamp verification but this option is not generally recommended.

Bind Card

BIND PACKAGE sub command is used to bind a DBRM to a package & BIND PLAN is used to bind a DBRM to a plan or group of package to a PLAN.

Sample Bind Card:

```
//BIND EXEC PGM=IJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD*
  DSN
  BIND PLAN(EMPPLAN)
  MEMBER(EMPDBRM)
  VALIDATE(BIND)
  ISOLATION(CS)
  RELEASE(COMMIT)
  OWNER(SMSXL86)
  LIB('SMSXL86.DB2,DBRMLIB')
```

Bind Parameter – Member & LIBRARY

DBRM to be bound is given in MEMBER & the partitioned dataset containing the DBRM is given in LIB. During BIND package, if an existing package is used to create the new package then, COPY should be mentioned instead of MEMBER & LIBRARY.

COPY (Collection-id.package-id)
COPYVER (version-id) – version of the copy to be used. Default is empty if

Bind Parameter – PLIST

PLIST is a BIND parameter of BIND PLAN. Package to be connected with PLAN are named here. PLIST (COLL1.*) bound the entire package with collection ID COLL1 to the PLAN.

Package Naming – Collection-ID & Versioning

Packages have three qualifiers: Location ID is used in distributed environment. Collection ID is used for grouping of packages.
Naming Syntax: Location-Id. Collection-Id. Package-name

VERSION with packages, the pre-compiler option VERSION can be used to bind multiple versions of the programs into package.

PROG → Pre-compile → BIND Package → COLL1.PROG.TEST
With VERSION (TEST)
→ Pre-compile → BIND Package → COLL1.PROG.PROD
With VERSION (PROD)

Bind Parameter – ACTION

Package or Plan can be an addition or replacement. Default is replacement. REPLACE will replace an existing package with the same name, location & collection. REPLVER (version-id) will replace a specific version of an existing package.
ADD – Add the new package to SYSIBM.SYSPACKAGE.
Syntax: ACTION (ADD|REPLACE)

Bind Parameter – Isolation Level

The isolation level parameter of the BIND command describes to what extent a program bound to this package can be isolated from the effects of other programs running. This is determines the duration of the page lock.

CURSOR STABILITY (CS) – As the cursor moves from the record in one page to the record in next page, the loc over the first page is released (provided the record is not updated). It avoids concurrent updates or deletion of row that is currently processing. It provides WRITE integrity.

REPEATABLE READ (RR) – All the locks acquired are retained until commit point. Prefer this option when your application has to retrieve the same rows several times & cannot tolerate different data each time. It provides READ & WRITE integrity.

UNCOMMITTED READ (UR) – It is also known as DIRTY READ. It can be applied only for retrieval SQL. There are no locks during READ & so it may read the data that is not committed. Highly dangerous & use it when concurrency is your only requirement. It finds its great use in statistical calculation of the large table & data-warehousing environment.

Bind Parameter – ACQUARE & RELEASE

ACQUIRE (USE) & RELEASE (COMMIT) – DB2 imposes table or table space lock when it executes SAL statements that references a table in the table space & it release the acquired lock on COMMIT or ROLLBACK. This is the default option and provides greater concurrency.

ACQUIRE (USE) & RELEASE (DEALLOCATE) – Locks table & table spaces on use & releases when the plan terminates.

ACQUIRE (ALLOCATE) & RELEASE (DEALLOCATE) – When DB2 allocates the program thread, it imposes lock over all the tables & table spaces used by the program. This option avoids deadlocks by locking your source at the beginning but it reduces concurrency.

ACQUIRE (ALLOCATE) & RELEASE (COMMIT) is NOT ALLOWED. This increases the frequency of deadlocks.

Bind Parameter – SQLERROR

If says whether to create the package in case of SQL errors.
SQLERROR (NOPACKAGE) – Default – If there is any SQL error, package will not be created.
SQLERROR (CONTINUE) – Create the Package even if SQL error are encountered.

BIND Parameter - VALIDATE

It controls the handling of 'object not found' & 'not authorized' errors. Default is VALIDATE (RUN) – If all objects are found & all privileges are held, then don't check once again at execution time. If there is any 'privilege issue' or 'object not found' error, then produce WARNING messages, bind the package & check authorization & existence at execution time.

VALIDATE (BIND) – If there is any privilege or object-not-found issue, then produce ERROR message & create the package only if SQLERROR (CONTIUNE) is coded in the bind card. The statement is error will not be executable.

BIND Parameter - Explain

EXPLAIN (YES) loads the access path selected by the optimizer in PLAN_TABLE. EXPLAIN (NO) is default & it wont load any such details in PLAN_TABLE.

BIND Parameter – QUALIFIER

Qualifiers are added at the bind time to the objects used in the program. It is suggested NOT to use any qualifiers in the program & use this parameter during BIND so that the program can be easily promoted from one region to next region without any change in program.

REBINDING

When SQL statements are not changed but a new index is added or RUNSTATS is run, then it is advisable to do a REBIND for the best possible access path. REBIND PLAN is cheaper than BIND with ACTION (REPLACE).

SYSTSIN CARD should be

```
REBIND PLAN(PLAN-NAME)
VALIDATE(BIND)
…
```

HOST Variables

DB2 is separate subsystem. As DB2 data set stored in external address space, you cannot directly modify or read from your program. During read, the DB2 values are retrieved into your working storage variables. During update, the DB2 columns are updated using your working storage variables. These working storage variables used for retrieving or updating DB2 data should be same size of Db2 columns. They are called as host variables as they are defined in the host language (COBOL). Host variables are prefixed with colon (:) in the embedded SQL.

DCLGEN (Declaration Generator)

DCLGEN is a DB2 utility. If we provide table details, it will generate DB2 structure & host variable structure of the table. It can also generate NULL indicators for the NULL columns. This output is stored in a PDS

member. These members are included in the program using the INCLUDE <pds-member>. INCLUDE is a pre-compiler statement 7 so it should be coded within the scope of EXEC SQL & END-EXEC

DCL generated host variable names are same as DB2 column names. As underscore is not a valid character for variable names in COBOL, it is replaced by hyphen in host variable generation of DCLGEN. Prefix & Suffix can be added to all the variables while creating the DCLGEN copybook.

Host variables can be declared in working storage section. But they cannot be stored in copybooks as other file layouts.

INCLUDE is expanded during pre-compilation time, but COPY is expanded during compilation & so declaration of host variable in copybook will produce errors during pre-compilation.

DECLARE TABLE that is table structure of DCLGEN is NOT really needed for pre-compilation. But it if is used, then any misspelled table name, column names are trapped in pre-compilation stage itself.

DB2 need not be up during pre-compilation. As pre-compiler just extracts the DB2 statements & produce DBRM, it does not need DB2 connection. But DB2 region should be up for binding as the optimizer builds access path in this stage from catalog information.

DCLGEN Panel (Option 2 of DB2I)

```
|
| DSNED                              DCLGEN                          SSID:  DSN
| = = = =>
|
| Enter Table Name For Which Declarations Are Required:
|  1    SOURCE TABLE NAME    = = => DSN6410.EMPTABLE     (Unqualified table name)
|  2    TABLE OWNER          = = =>                      (Optional)
|  3    AT LOCATION          = = =>                      (Optional)
|
| Enter destination data set:                  (Can be sequential or portioned)
|  4    DATA SET NAME        = = =>
|  5    DATA SET PASSWORD    = = =>          (If password protected)
|
| Enter options as desired:
|  6    ACTION               = = => ADD      (ADD new or REPLACE old declaration)
|  7    COLUMN LABEL         = = => NO       (Enter YES for column lable)
|  8    STRUCTURE NAME       = = =>                          (Optional)
|  9    FIELD NAME PREFIX    = = =>                          (Optional)
|  10   DELIMIT DBCS         = = => YES      (Enter YES to delimit DBCS identifiers)
|  11   COLUMN SUFFIS        = = => NO       (Enter YES to append column name)
|  10   INDICATOR VARS       = = => NO       (Enter YES to indicator variables)
|
|  PRESS: ENTER to process          END to exit          HELP for more information
|
```

DSNE9051 EXECUTION COMPLETE, MEMBER EMPTABLE ADDED

DB2 Data Types & equivalent host variables

DB2 Column	Bytes	COBOL PIC Clause	Bytes
SMALLINT	2	PIC S9(04) COMP	2
INTEGER	4	PIC S9(09) COMP	4
DECIMAL (P,Q) (P should be less than 32)	Int (P/2)	PIC S9(P-Q)V9(Q)	Integer ((P+Q)/2+1)
DATE	4	PIC X(10)	8
TIME	3	PIC X(08)	6
TIMESTAMP	10	PIC X(26)	26
CHAR(n) (n=1 To 254)	N	PIC X(n)	N
VARCHAR(n) (n=0-4046)	N	01 WS-ITEM 49 WS-LENGTH PIC S9(04) COMP 49 WS-COLUMN PIC X(n)	N+2

Limit of Date: Jan 1st 1 AD – Dec 31st 9999 AD
Limit of Time: 000000 – 240000
Limit of Time Stamp – 0001010100000000000000 – 00001231240000000000

SQLCA

SQLCA is SQL communication area. It is a DB2 copybook that should be included in all the DB2 programs. The fields declared in the copybook are updated for every DB2 query. The length of SQLCA is 136. The most important field in the copybook is SQLCODE. The success or failure of DB2 query can be checked in this field

Value of 0 in SQLCODE indicates the successful operation.
Positive value in SQLCODE indicates the completed operation with some exception Negative value in SQLCODE indicates the unsuccessful operation.

So it is common programming practice that the SQLCODE of every query must be verified for valid values after the execution of the query. If the SQLCODE value is no acceptable, then the control is transfers to ABEND routine that would end the program with proper error messages.

SQLERRD (3) contains the number of rows affected by a DB2 INSERT/DELETE operation. If the operation is DELETE, this contains number of rows deleted after the execution of the query.

WHENEVER statement

WHENEVER is an error-trapping statement that directs the processing to continue or branch to an error handling routine based on the SQLCODE returned for the statement. When it processed, it applies to all the subsequent SQL statements issued by the application program.

EXEC SQL
WHENEVER condition action.
END-EXEC

Conditions can be NOT FOUND, SQLWARNING or SQLERROR.
Action can be either CONTIUNE or GO TO

NOT FOUND is TRUE if SQLCODE is 100.
SQLWARNING is TRUE if SQLCODE is greater than zero but not equal to 100
SQLERROR is TRUE if SQLCODE is less than zero.

It is almost always safe to code specific SQLCODE checks after each SQL statements & process accordingly. Avoid using WHENEVER.

DSNTIAR – SQLCA Formatter

DSNTIAR is an IBM supplied program, which would display the error message in formatted way. Most of the installations call this program in their error routine.

```
CALL'DSNTIAR' USING SQLCA, ERROR-MESSAGE, ERROR-TEXT-LEN.

01 ERROR-MESSAGE.
    05 ERROR-LEN PIC S9(04) COMP VALUE +1320.
    05 ERROR-TEXT PIC X(132) OCCURS 10 TIMES.

77 ERROR-TEST-LEN PIC S9(09)  COMP VALUE  +72.
```

Components of DB2

DB2 internal structure is very complex & contains large number of components. But from a high-level point of view, DB2 can be considered to have four major components.

1. Systems Services Component: Supports System operation, operator communication logging & similar function.
2. Locking Services Component: Provides the necessary control for managing concurrent access to data.
3. Database Services Component: Supports the definition, retrieval & update of user & system data.
4. Distributed Data Facility Component: Provides DB2's distributed database support.

Before get into SQL, let us briefly see the sub-components of Database services component.

Database Services Component

It has five sub-components in it. We have already discussed the functions of two components – pre-compiler & Optimizer (Bind).

Runtime Supervisor:

It is resident in the main memory when the application program is executing. Its job is to oversee that execution. When the program requests some database operation to be performed, control first goes to the runtime supervisor, which uses the control information in the application plan to request the appropriate on the part of the data manager.

Data Manager (DM):

It performs all the normal access method functions – search, retrieval, update etc. it invokes other system components in order to perform details functions such as locking, logging etc.

Buffer Manager (BM):

It is responsible for transferring the data between external storage and virtual memory. It uses sophisticated techniques such as 'read-head-buffering' and 'lock-aside-buffering' to get the best performance out of the buffer pools under its control to minimize the amount of physical i/o actually performed.

Catalog & Directory are also part of database services component only.

SQL

SQL is fourth generation language. The definition of fourth language is 'Tell the need. System will get if done for you'. There is no need to instruct the method to get it.

For example, I want list of employee working currently in Malaysia. If the data is in file, then I have to declare, define, & Open the file. Then I should read all the records into my working storage one by one and if their location is Malaysia, I display them. Finally I close the file.

If the data is in DB2, I would simply write a query (SELECT * FROM table where location = 'Malaysia') & DB2 optimizer do everything for me in identifying the best access path. No worries of DECLARE, DEFINE, OPEN, READ & OMIT stuffs. That's the power of fourth generation language & that makes the programmer life easy!!

SQL has three kinds of statements.

DDL – Data Definition Language Statements	CREATE ALTER DROP
DML- Data Manipulation Language Statements	SELECT INSERT UPDATE DELETE
DCL-Data Control Language Statements	GRANT REVOKE

DDL CREATE

This statement is used to create DB2 objects.

General Syntax	CREATE object object-name parameters
Table creation sample	CREATE TABLE table-name. (Column definition, Primary Key definition, Foreign Key definition, Alternate Key definition, LIKE table-name/View-name, IN DATABASE-NAME.TABLESPACE-NAME)

DDL-Column Definition

Define all the columns using the syntax:

Column-name Data-type Length (NULL/NOT NULL/NOT NULL WITH DEFAULT)

Data-types are already explained in DCLGEN section.

→ Column-name can be of maximum length 30. It should start with an alphabet & it can contain numbers & underscore.

→ NULL means UNKNOWN; if the value is not supplied during an insertion of a row, then NULL will be inserted into this column.

→ NOT NULL means, value for this column should be mentioned when inserting a row in the table. NOT NULL with DEFAULT means, if the value for this column is not supplied during an insertion, then based on type of the column default values will be moved into the table. Default are listed in the below table.

Data-Type	Default Value
CHAR	Spaces
VARCHAR	Empty String (String of Length 0)
DATE, TIME, TIMESTAMP	Current (DATE/TIME/TIMESTAMP)
INTEGER SMALLINT DECIMAL	Zero

DDL – Primary Key Definition

Candidate key is defines as column(s) that uniquely identifies each row in the table. One of the candidate keys is defines as primary during the table creation. The main entity, about which the information is stored, is selected as primary key.

If the PRIMARY KEY is single column(s), then it can be coded along with column definition. If the primary key is composite key, more than one column, then it can de defined after the column definition. Primary key definition can be added or dropped later using ALTER table.

The definition of primary key is complete only after unique index is created on it, until that time, the table is unavailable for use. If the primary key is added using ALTER command, then unique index on the key column(s) must already be available.

Syntax: PRIMARY KEY (column1, column2)

DDL – Foreign Key definition

Defining a foreign key establishes a referential constraint between the foreign key (Child Table) & the primary key (parent table). The parent table of a referential constraint must have a primary key & a primary index.

There must be a value in parent table (primary key) for every non-null value in child table (foreign key). Foreign key can be defined along with column definition or separately after the definition of all the columns.
Syntax 1: (Along with column definition)
COLUMN1 CHAR(4) REFERENCES TABLE2 ON DELETE CASCADE/DELETE/SET NULL

Syntax 2: (After column definition)
FOREIGN KEY FKEY(COLUMN1,COLUMN2) REFERENCES TABLE2 ON
DELETE/CASCADE/SET NULL

Delete Rule.

Delete rule defines what action needs to be taken on child table when there is a deletion in primary table. The three possibilities are below:

CASCADE: Child table rows referencing parent table will also be deleted.
RESTRICT: Cannot delete any parent row when there is reference in child table.
SET NULL: SET the foreign key value as NULL when the respective primary row is deleted.

Inset & Update rules cannot be defines. But they can be implemented using TRIGGERS.

DDL – Alternate Key Definition

Candidate keys that are not defined as primary key are called alternate keys. This can be defined along with column definition or separately after the definition of the entire column using the keyword UNIQUE. Alternate key definitions CANNOT be added later using ALTER command.

Syntax 1: COLUMN1 CHAR(10) UNIQUE NOT NULL
Syntax 2: UNIQUE(Column1)

DDL – Adding Constraints

A check constraint allows you to specify the valid value range of a column in the table. For example, you can use a check constraint to make sure that no salary in the table is below Rs 10000, instead of writing a routine in your application to constrain the data.

CHECK clause & the optional clause CONTSTRAINT (for named check constraints) are used to define a check constraint on one or more columns of a table. A check constraint can have a single predicate, or multiple predicates joined by AND or OR. The first operand of each predicate must be a column name, the second operand can be a column name or constant, and the two operands must have compatible data types.

Syntax 1: ADD CHECK (WORKDEPT BETWEEN 1 & 100)
Syntax 2: ADD CONSTRAINT BONUSCHK CHECK (BONUS < = SALARY);

Although CHECK IS NOT NULL is functionally equivalent to NOT NULL, it wastes space & is not useful as the only content of a check constraint. However, later if you want to remove the restriction that the data be Non-null, you must define the restriction using the CHECK IS NOT NULL clause.

DDL – Index creation

Table spaces should be created before creating the table. But index spaces are automatically created with creation of index. So create index command provides optional automatically created with creation of index. So create index command provides optional parameter of USING STOGROUP, PRIQTY, SECQTY, FREEPAGE & PCTFREE etc. for the allocation of index space. But usually we don't mention these parameters and allow the system to allocate it in the right place with respect to table space.

CREATE [UNIQUE] INDEX index-name on Table-name (column-name [asc/desc],..)
CREATE UNIQUE INDEX EMPINDEX ON EMPTABLE (EMPNO ASC)

Creating unique index on a NULL column is possible if there is only one NULL is available in that column.

DDL – Sample Storage group Creation

CREATE STOGROUP DSN8G41
VOLUMES (DBPK01, DBPK02)
VCAT DB2CATZ;

Using the ALTER command, volumes can be added or removed at later stage using the command ADD VOLUMES & REMOVE VOLUMES.

DDL – Sample Database Creation

CREATE DATABASE DSN8D41A

```
    STOGROUP DSN8G410
    BUFFERPOOL BPO;
```

DDL – Sample Table Space Creation

```
CREATE TABLESPACE DSN8S41D
    IN DSN8D41A
    USING STOGROUP DSN8G410
        PRIQTY 20
        SECDTY 20
        ERASE NO
    LOCKSIZE PAGE LOCKMAX SYSTEM
    BUFFERPOOL BPO
    CLOSE NO
    FREEPAGE 0 PICTURE 5
    DSETPASS DSN8;
```

PRIQTY - Primary space to be allocated during table space creation. (KB)
SECQTY – Secondary space taken as amount of data in table space grows. (KB)
ERASE – Whether DB2 defined datasets are to be erased when table space is dropped.
LOCKSIZE – Indicate size of lock. Can be TABLESPACE, TABLE, PAGE, ROW or ANY. ANY means DB@ will decide the lock.
LOCMAX – Indicated maximum number of page/row locks above which lock escalation occurs.
BUFFERPOOL – Buffer pool associated with the table space.
CLOSE – Indicates whether datasets associated with table space should be closed when there are no current users of table space.

FREEPAGE – Number of pages after which a empty page is available.
PCTFREE – Percentage of page to be left free for future inserts.

DDL – Sample Table Creation

```
CREATE TABLE DSN8410.EMP
    (EMPNO      CHAR(6)             NOT NULL,
     NAME       VARCHAR(50)         NOT NULL,
     WORKDEPT   CHAR(3),
     PHONENO    CHAR(4)             CONSTRAINT NUMBER CHECK
                (PHONENO >= '0000' AND
                 PHONENO <= '9999'),
     BIRTHDATE DATE,
     SALARY DECIMAL(9,2),
     PRIMARY KEY (EMPNO),
     FOREIGN KEY RED (WORKDEPT) REFERENCES DSN8410.DEPT
     ON DELETE SET NULL)
IN DSN8D41A.DSN8S41D;
```

DDL – Sample View Creation

The general syntax of view definition is
CREATE VIEW view-name (column1,..) AS sub-query [WITH CHECK OPTION]

WITH CHECK OPTION ensures that all the inserts & updates made to the base table using this view satisfy the limit imposed by the where condition of the sub-query. (Domain Integrity)

```
CREATE VIEW DB2GROUP (EMPID EMPNAME SKILLSET)
AS SELECT EMPID, EMPNAME, SKILLSET
    FROM DSN8410.EMP
    WHERE SKILLSET = 'DB2' WITH CHECK OPTION;
```

This view shows name & id of all the employees who knows DB2. 'WITH CHECK OPTION' ensures that all the inserts made to base table using the view DB2GROUP, must have DB2 as SKILLSET. Any update-attempt to change the value of SKILLSET from DB2 to anything else using the view DB2GROUP is restricted.

DDL – ALTER TABLE

When you create a table using model table, the columns are defines in the new table as in the model table. But the PRIMARY, FOREIGN KEY definitions are not using ALTER TABLE, there should be unique index..

```
CREATE TABLE YDEPT
    LIKE DSN8410.DEPT

CREATE UNIQUE INDEX YDEPTX
    ON YDEPT (DEPTNO);

ALTER TABLE YDEPT
    PRIMARY KEY(DEPTNO);
```

Using ALTER statement, alternate keys cannot be defined. Other than that we can add columns, constraints, checks & DROP primary key.

When a new column is added with NOT NULL WITH DEFAULT, then the value for the column for all the rows already exist in the table is filled based on the type of column:

1. Numeric - Zero
2. Char, Varchar - Spaces
3. Date - 01/01/0001
4. Time - 00:00 AM

DDL-DROP

DROP deletes the table definition in addition to all the rows in it. You also lose all synonyms, views, indexes, referential and check constraints associated with that table. Plans & Package are deleted using the command FREE.

DROP Object Object-Name
DROP TABLE table-name
FREE PACKAGE(COLL1,-,) Deletes all the packages belongs to collections COLL1.

Data Manipulation Language Statements

DML-SELECT

SELECT statement is the heart of all queries & it is specifically used for retrieving information. As a primary statement to retrieve the data, it is the most complex & most complicated DML statement used. Six different clauses can be used with a SELECT statement & each of these clauses has its own set of predicates. They are FROM-WHERE-GROUPBY-HAVING-UNION and ORDER BY.

Syntax:
```
SELECT column FROM table
WHERE conditions
GROUP BY columns
HAVING conditions
ORDERBY columns
```

Up to 750 columns can be selected. 15 sub-queries can be coded in addition to one main query.

DML-FROM clause of SELECT

FROM clause is used, in conjunction with one or more references, to specify DB2 data manager where to retrieve data.

WHERE Clause

It is always followed by a search condition that contains one or more predicates that defines how the DB2 DM is to choose the information that will be contained in the result dataset produced.
Any relational operator can be coded. (<>,>,<,=,<=,>=)

DML-SELECT-WHERE-NOT condition

NOT keyword can be used to select all the rows except the row identified with the search condition.
Syntax: WHERE NOT <condition>.
'NOT' cannot be used directly with comparison operators. NOT = is invalid.

DML-SELECT-WHERE-LIKE condition

LIKE is used for pattern search. % is a wild card for ant number of zero or more character and '-' is used to indicate single unknown character.

If you want to look for % in the string, then you should define ESCAPE character. Ex: WHERE col1 LIKE 'MUTHU_+%SARA%' ESCAPE '+'

This will look for a string that starts with 'MUTHU_%SARA' where _ can be any character.

DML-SELECT-WHERE-BETWEEN and IN.

Use BETWEEN to check the value of column is within the range. The lower & upper boundaries are INCLUSIVE. Use IN to check the value of column against a values list.

DML-SELECT-WHERE-ANY/SOME/ALL.

SELECT … WHERE COLUMN1 > ANY|ALL|SOME (Sub-Query)

ALL – If the value of COLUMN1 is greater than all the values return by sub-query, then only the outer table row will be selected. If no rows are returned by sub-query then all the rows of outer table will be selected.

ANY & SOME – If COLUMN1 value is greater than one of the value return by sub-query, then the outer table row will be selected.

DML-SELECT-WHERE-EXIST.

This is useful when you want to just check whether one or more rows exist.

Syntax: WHERE EXISTS (Sub-Query)

Your main query will return result only if at least one row exists for your sub query. NOT EXIST check for unavailability of any rows for the search condition in the sub query.

DML-SELECT-GROUP BY.

Use GROUP BY for grouping rows by values of one or more columns. You can then apply column function for each group. Except for the columns named in the GROUP BY clause, the SELECT statement must specify any other selected columns as an operand of one of the column functions.

If a column you specify in the GROUP BY clause contains null values, DB2 considers those null values to be equal. Thus all nulls form a single group. You can also group the rows by the values of more than one column.

DML-SELECT-UNION & UNION ALL.

Using the UNION keyword, you can combine two or more SELECT statements to form a single result table. When DB2 encounters the UNION keyword, it processes each SELECT statement. If you use UNION to combine two columns with the same name, the result table inherits that name.

You can use UNION to eliminate duplicates when merging lists of values obtained from several tables whereas UNION ALL retains duplicates.

DML-SELECT-ORDER BY

The ORDER BY clause is used to short & order the rows of data in a result dataset by the values contained in the column(s) specified. Default is ascending order (ASC). If the keyword DESC follows the column name, then descending order is used. Integer can also be used in place of column name in the ORDER BY clause.

```
SELECT COL1,COL2,COL3 FROM TAB1 ORDER BY COL1 ASC COL3 DESC
SELECT COL1,COL2,COL3 FROM TAB1 ORDER BY 1 ASC 3 DESC
```

DML-SELECT-CONCAT

CONCAT is used for concatenating two columns with the string you want in between. To concatenate the last name & first name with comma in between,

```
SELECT LASTNAME CONCAT ',' CONCAT FIRSTNME
    FROM DSN8410.EMP;
```

COLUMN FUNCTION & SCALAR FUNCTION

Column function receives a set of values from group of rows & produces a single value. SCALAR function receives one value & produces on value.

COLUMN FUNCTION:

Function	Returns	Example
MAX	Maximum value in a column	MAX(SALARY)
MIN	Minimum value in a column	MIN(SALARY)
AVG	Average value of a column	AVG(SALARY)
SUM	Sum value of a column	SUM(SALARY)
COUNT	Number of selected rows	COUNT(*)

1. DISTINCT can be used with the SUM, AVG & COUNT functions. The selected function operates on only the unique value in a column.
2. SUM & AVG cannot be used for non-numeric data types where as MIN, MAX & COUNT can.

SCALAR FUNCTION

Function	Returns	Example
CHAR	String representation of its first argument.	CHAR(HIREDATE)
DATE	Date derived from its argument.	DATE('1977-12-16')
DAY	Day part of its argument	DAY(DATE1-DATE2)
DAYS	Integer representation of its argument.	DAYS('1977-12-16') – DAYS('1977-12-16') + 1
MONTH	Month part of its argument.	MONTH(BIRTHDATE) = 12
YEAR	Year part of its argument.	YEAR(BIRTHDATE) = 1977
DECIMAL	Decimal representation of its first argument.	DECIMAL(AVG(SALARY)8,2)
FLOAT	Floating-point representation of its first argument.	FLOAT(SALARY)/COMM
HEX	Hexadecimal representation of its first argument.	HEX(BCHARCOL)
INTEGER	Integer representation of its argument.	INTEGER(AVG(SALARY)+0.5)
HOUR	Hour part of its argument.	HOUR(TIMECOL) > 12
MINUTE	Minute part of its argument.	MINUTE(TIMECOL) > 59
SECOND	Second part of its argument	SECOND(TIMECOL) > 59
MICROSECOND	Microsecond part of its argument	MICROSECOND(TIMECOL) > 12
TIME	Time derived from its argument.	TIME(TSTMPCOL) < '12:00:00'
TIMESTAMP	Timestamp derived from its argument or arguments.	TIMESTAMP(DATECOL,TIMECOL)
LENGTH	Length of its argument.	LENGTH(ADDRESS)
SUBSTR	Sub-String of a string	SIBSTR(FSTNAME,1,3)
VALUE	The first argument that is not null.	VALUE(SMALLINT1,100) + SMALLINT2 > 1000

CURSOR

CURSOR is useful when more than one row of a table to be processed. It can be loosely compared with sequential read of file. Usage of cursor involves four steps.

1. DECLARE Statement.
   This statement DECLARES the cursor. This is not an executable statement.
   Syntax: DECLARE cursor-name CURSOR [WITH HOLD] FOR your-query
   [FOR UPDATE OF column1 column2 | FOR FETCH ONLY]

2. OPEN statement.
   This statement just readies the cursor for retrieval. If the query has ORDER BY or GROUP BY clause, then result table is built. However it does not assign values to the host variables.
   Syntax: OPEN Cursor-name.

3. FETCH statement.
   It returns data from the results table one row at a time & assigns the value to specific host variables.
   Syntax: FETCH cursor-name INTO
   : WS-Column1

: WS-Column2

The number of INTO variables should be equal to SELECT columns of your DECLARE cursor statement. If FOR UPDATE OF clause is coded, then while fetching the row, the exclusive lock is obtained over the page of the row. FETCH cursor is usually places in the perform loop that will be executed until the SQLCODE is 100. SQLCODE will be sent to 100 when end of cursor is reached.

If FOR UPDATE OF clause is given, then the entire columns that are going to be updated using the cursor should be mentioned in this clause. If you don't mention FOR UPDATE OF clause, you can update any number of columns using WHERE CURRENT OF cursor-name but you will be getting the exclusive lock only during the UPDATE statement & there are chances for the row to be locked exclusively by another task in between your READ & UPDATE.

So it is suggested to use FOR UPDATE OF clause when you are using the cursor for update purpose.

4. CLOSE statement.

   It releases all resources used by the cursor. If you don't close the cursor, COMMIT statement will close the cursor. To retain the cursor in OPEN stage during COMMIT, Use WITH HOLD option in the DECLARE statement of the CURSOR. This option will be effective only in batch environment. ROLLBACK close all the cursors including the cursors defined with 'WITH HOLD' option

   Syntax: CLOSE cursor-name.

Read Only Cursor

The cursor that cannot be used for any update purpose is called as READ ONLY CURSOR. For example, cursor defined with UNION, JOIN or GROUP BY cannot be used for update purpose.

DML-INSERT

INSERT statement is used to insert rows to the table. NOT NULL column values should be supplied during INSERT. Otherwise INSERT would fail.

1. Specify values for columns of single row to insert.

```
INSERT INTO YDEPT (DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION)
  VALUES ('E31', 'DOCUMENTATION','000010','E01','');
INSERT INTO YEMP
  VALUE('000400','RUTHERFORD','B','HAYES','E31','5678','1983-01-01','MANAGER',
16,'M','1943-07-10'.24000,500,1900);
```

   You can name all the columns or omit. When you list the column names, you must specify their corresponding values in the same order as in the list of column names.

2. Mass Insert. Another table or view contains the data for the new row(s).

```
INSERT INTO TELE
  SELECT LASTNAME, FIRSTNAME, PHONENO FROM DSN8410.EMP
  WHERE WORKDEPT = 'D21';
```

   If the INSERT statements are successful, SQLERRD (3) is set to the number of rows inserted.

Dependant table insertion: (Table with foreign Key)

Each non-null value you insert into a foreign key column must be equal to some value in the primary key (the primary key is in the parent table). If any field in the foreign key is null, the entire foreign key is considered null. If you drop the index enforcing the primary key of the parent table, an INSERT into either the parent table or dependent table fails.

DML-UPDATE

UPDATE statement is used to modify the data in a table. The SET clause names the columns you want to update & provide the values you want them changed to. The condition in the WHERE clause locates the row(s) to be updated. If you omit the WHERE clause, DB2 updates every row in the table or view with the values you supply.

If DB2 finds an error while executing your UPDATE statement (for instance, an update value that is too large for the column), it stops updating and returns error codes in the SQLCODE & SQLSTATE host variables & related fields in the SQLCA. No rows in the table change (rows already changed, if any, are restored to their previous values). If the UPDATE statement is successful, SQLERR (3) is set to the number of rows updated.

Syntax: UPDATE table-name SET column1=value1, column2=value2 [WHERE condition];

DML-DELETE

DELETE statement is used to remove entire rows from a table. The DELETE statements removes zero or more rows of a table, depending on how many rows satisfy the search condition you specified in the WHERE clause. If you omit a WHERE clause from a DELETE statement, DB2 removes all the rows from the table or view. You have named. The DELETE statement does not remove specific column from the row. SQLERRD (3) in the SQLCA contains the number of deleted rows.
Syntax: DELETE FROM table-name [WHERE CONDITION]

DML-NULLS

One of the 12 CODD rules for relation database system is ‘NULL values are supported for representing missing information in a systematic way irrespective of the data type’. DB2 supports NULL values.

NULL IN SELECT STATEMENT

NULL is defined as Unknown value in RDBMS terminology.

Two unknown values cannot be same. As EQUAL can be used only for known values, COLUMN1=NULL is the right way of coding. If GROUPBY is done on a NULL column, then all the columns whose value is unknown (NULL) forms one GROUP.

Example, if QUALIFICATION is a column defined with NULL attribute in EMPTABLE, then SQL for retrieving all the employees whose QUALIFICATION is not yet known is:

```
SELECT EMPNAME, QUALIFICATION FROM EMPTABLE
 WHERE QUALIFICATION IS NULL
```

High-level languages don’t have any NULL concept. So if the column is NULL, then the host variable corresponds to that column should be set to zero (numeric) or spaces (non-numeric). This can be done in the programming. But for doing this we should now whether the column is NULL or NOT. NULL indicators are used for this purpose. NULL indicators is a 2 byte field (S9(4) COMP). Negative values (-1) in NULL indicator field indicate that the column is NULL.

```
EXEC SQL
SELECT QUALIFICATION
```

```
        INTO :WS-QUALIFICATION :WS-QUALIFICATION-NULL
        FROM EMPTABLE
        WHERE EMPID=2052
END-EXEC
IF SQLCODE=0
        PERFORM 1000-CHECK-FOR-NULLS
…..
1000-CHECK-FOR-NULLS
        IF WS-QUALIFICATION-NULL < 0 THEN
                MOVE SPACES TO WS-QUALIFICATION
        END-IF.
```

If QUALIFICATION of EMPID = 2052 is NULL & if you didn't code null indicator, then the query will fail with SQLCODE -305 & the error message corresponding to this code is 'THE NULL VALUE CANNOT BE ASSIGNED TO OUTPUT HOST VARIABLE NUMBER posit-num BECAUSE NO INDICATOR VARIABLE IS SPECIFIED.

This kind of NULL check (1000-CHECK-FOR-NULLS) is highly recommended for numeric fields. Otherwise the program may abnormally end at some place down the line, when the field is used for some arithmetic operation.

Instead of NULL indicator, VALUE function can also be used.

```
SELECT VALUE(QUALIFICATION.' ')
        INTO :WS-QUALIFICATION
        FROM EMPTABLE
        WHERE EMPID=2052
```

If QUALIFICATION is NULL, then ' ' will be moved to WS-QUALIFICATION.

DML-NULLS

One of the 12 CODD rules for relation database system is NULL values are supported for representing missing information is a systematic way irrespective of the data type. DB2 supports NULL values.

NULL IN SELECT STATEMENT

NULL is defined as Unknown value in RDBMS terminology.

Two unknown values cannot be same. As EQUAL can be used only for known values, COLUMN1=NULL is meaningless. In the where predicate if NULL needs to be checked, WHERE COLUMN1 IS NULL is the right way of coding. If GROUPBY is done on a NULL column, then the entire column whose value is unknown (NULL) forms one GROUP.

Example, if QUALIFICATION is a column defined with NULL attribute in EMPTABLE, then all the columns whose value is unknown (NULL) forms one yet known is:

```
SELECT EMPNAME, QUALIFICATION FROM EMPTABLE
  WHERE QUALIFICATION IS NULL
```

High-level languages don't have any NULL concept. So if the column is NULL, then the host variable corresponds to that column should be set to zero(numeric) or spaces(non-numeric) or spaces(non-numeric). This can be done in the programming. But for doing this we should know whether the column is NULL or NOT. NULL indicators are used for this purpose. NULL indicator is a 2 bytes field (S9(04) COMP). Negative values (-1) in NULL indicator field indicate that the column is NULL.

```
EXEC SQL
SELECT QUALIFICATION
        INTO : WS-QUALIFICATION:WS-QUALIFICATION-NULL
        FROM EMPLOYEE
        WHERE EMPID = 2052
END-EXEC
IF SQLCODE=0
        PERFORM 1000-CHECK-FOR-NULLS
….

1000-CHECK-FOR-NULLS
        IF WS-QUALIFICATION-NULL < 0 THEN
                MOVE SPACES TO WS-QUALIFICATION
        END-IF.
```

If QUALIFICATION of EMPID = 2052 is NULL & if you didn't code null indicator, then the query will fail with SQLCODE -305 & the error message corresponding to this code is 'THE NULL VALUE CANNOT BE ASSIGNED TO OUTPUT HOST VARIABLE NUMER posit-num BECAUSE NO INDICATOR VARIABLE IS SPECIFIED'.

This kind of NULL check (100-CHECK-FOR-NULLS) is highly recommended for numeric fields. Otherwise the program may abnormally end at home place down the line, when the field is used for some arithmetic operation.

Instead of NULL indicator, VALUE function can also be used.

```
SELECT VALUE (QUALIFICATION,'')
        INTO: WS-QUALIFICATION
        FROM EMPTABLE
        WHERE EMPID=2052
```

If QUALIFICATION is NULL, then '' will be moved to WS-QUALIFICATION.

<u>NULL IN INSERT/UPDATE STATEMENT</u>

The concept is same. DB2 informs the NULL presence thru the NULL indicator the programming language. In similar way, the programming language should inform the NULL to DB2 by NULL indicator.

Just before INSERT or UPDATE move -1 to NULL indicator variable & use this variable along with host variable for the column to be inserted/updated. Once -1 is moved to null indicator, then independent of value presence in the host variable, NULL will be loaded. In the following query, through 'B.E' is moved to host variable, it will not be loaded into the table as null indicator has -1. It should have been set to 0 before loading.

```
MOVE -1 TO WS-QUALIFICATION-NULL
MOVE 'B.E' TO WS-QUALIFICATION.
EXEC SQL
        UPDATE EMPTABLE
        SET QUALIFICATION =: WS-QUALIFIACTION: WS-QUALIFICATION-NULL
        WHERE EMPID = 2052
END-EXEC.
```

It is common feeling that NULL provides more trouble than benefit. So it is always better to specify NOT NULL or NOT NULL WITH DEFAULT for all the columns.

Values in null indicator

A negative value in null indicator field implies that the column is NULL & a positive value or ZERO in null indicator implies that the column is NOT NULL. Value of -2 in null indictor indicates that the column has been set to null as a result of a data conversion error.

DCL-GRANT & REVOKE commands

Data Control Language consists of commands that control the user access to the database objects. The Database Administrator (DBA) has the power to give (GRANT) & take (REVOKE) privileges to a specific user, thus giving or denying access to the data.

Syntax:
GRANT access ON object TO USER-ID | PUBLIC [WITH GRANT OPTION]
REVOKE access ON object from user-id | PUBLIC.

Example:
GRANT SELECT, UPDATE ON EMPTABLE TO SMSXL86.
GRANT ALL ON EMPTABLE TO SMSXL86.
REVOKE CREATE TABLE, CREATE VIEW FROM SMSXL86.
REVOKE INSERT ON EMTABLE FROM SMSXL86.

COMMIT & ROLLBACK

A transaction or a unit of work is a recoverable sequence of one or more SQL operations that are grouped together as a single unit, usually within in an application process.

All the changes made to the database since the initiation of transaction, are made permanent by COMMIT. ROLLBAC brings the database to the last committed point.

Syntax:
EXEC SQL COMMIT END-EXEC
EXEC SQL ROLLBACK END-EXEC

DB2 Restart Logic:

Usually there is only one COMMIT or ROLLBACK just before the termination of the transaction. But it is not preferred always.

If the program is updating one million records in the table space & it abnormally ends after processing ninety thousand records, then the issued ‘ROLLBACK’ brought the database back to the point of transaction initiation. The restart of the program must have to process ninety thousand successful-but-not-committed updates once again. The repeated cost occurred is due to bad design of the application.

If the program is expected to do huge updates, then commit frequency has to be chosen properly. Let us say, after careful analysis, we have designed our application COMMIT frequency as thousand records. If the program abnormally ends while processing 1500^{th} record, then the restart should not start from list record but from 1001^{st} record. This is done using restart logic.

Create one temporary table called RESTARTS with a dummy record & inserts one record into the table for every commit with key & occurrence of commit. This insertion should happen, just BEFORE the issue of COMMIT.

First paragraph of the procedure should read the last record of the table & skipped the records that are already processed & committed (1000 in the previous case). After the processing of all the records (one million), delete the entries in the table & issues one final COMMIT.

JOINS

Most of the times, the complete information about an entity is retrieved from more than one table. Retrieving information from more than one table can be done in two days. One method is JOIN & the other method is UNION. It is used to get information about more entities. In other words, it returns rows. JOIN is used to get details information about entities. In other words, it returns more columns.

There are two kinds of JOIN. Inner join returns rows from tow or more tables based on matching values. Outer join returns rows from two or more tables independent of matching or non-matching.

JOIN: INNER

Two or more tables are joined together using the column having equal values among them.

EMPTABLE			SALTABLE	
EMP_ID	EMP_NAME	DESIG	DESIG	SALARY
100	MUTHU	TL	SSE	400000
101	DEVI	SSE	SE	300000

```
SELECT EMP-NAME SALARY FROM EMPTABLE, SALTABLE WHERE
    EMPTABLE.DESIGNATION = SALTABLE.DESIGNATION
```

Result

DEVI 400000

Since there is no TL designation in the SALTABLE, MUTHU will not appear in the output,

JOIN: FULL OUTER JOIN.

The cause LEFT OUTER JOIN includes rows from the table named before it where values in a row of the result table contain nulls.

```
SELECT EMP-NAME SALARY FROM EMPTABLE FULL OUTER JOIN SALTABLE ON
    EMPTABLE.DESIGNATION = SALTABLE.DESIGNATION
```

Result:

MUYHU

DEVI 400000

----- 300000

JOIN: LEFT OUTER JOIN

The clause LEFT OUTER JOIN includes rows from the table named before it where the values in the joined columns are not matched by values in the joined columns of the table named after it.

```
SELECT EMP-NAME SALARY FROM EMPTABLE LEFT OUTER JOIN SALTABLE ON
    EMPTABLE.DESIGNATION = SALTABLE.DESIGNATION
```

Result

MUTHU ----

DEVI 400000

JOIN: RIGHT OUTER JOIN

The clause RIGHT OUTER JOIN includes rows from the table named after it where the values in the joined columns are not matched by values in the joined columns of the table named before it.

```
SELECT EMP-NAME SALARY FROM EMPTABLE RIGHT OUTER JOIN SALTABLE ON
    EMPTABLE.DESIGNATION = SALTABLE.DESIGNATION
```

Result

```
DEVI        400000
-----       300000
```

Restriction on Outer Joins

1. VALUE or COALESCE function can be used only with FULL OUTER join.
2. Only '=' can be used as comparison operator in FULL OUTER join.
3. Multiple conditions can be coded with AND OR & NOT are not allowed.

Sub-Queries

Sub-queries can appear in the predicates of other sub-query. This is called nested sub-query & the nesting is possible up to 15 levels.

Usually sub-query executes only once & it is executed first & the value returned by sub-query will be used for main query. This kind of sub-query is called as un-correlated sub-query. When the sub-query refers to the table referred in this kind of query is called correlated sub-query. Correlated sub-queries cannot be used for update/inset/delete purpose.

Example:

The following query listed the employees who get more than the average salary of their department.

```
SELECT EMPNAME, DEPTNUM, SALARY FROM EMPTABLE
    WHERE SALARY > (SELECT AVG (SALARY)
                    FROM DEPTNUM
                    WHERE EMPTABLE.DEPTNUM = DEPT.DEPTNUM)
```

OPTIMIZER – ACCESS PATH SELECTION

Generally the fastest way to access DB2 data is with index. Indexes are structured in such a way to increase the efficiency of finding a particular piece of data. DB2 uses many different algorithms to traverse an index structure. These algorithms are designed to elicit optimum performance in a wide variety of data access scenarios.

To use an index:

1. One of the predicates for the SQL statement must be index-able.
2. One of the columns (in any index-able predicate) must exist as column in an available index.

There are various types of indexes are available. They are:

1. Direct Index Look-Up.
2. Index Scan – Matching Index Scan.
3. Index Scan – Non-matching Index Scan.

4. Index-Only access.
5. List pre-fetch.
6. Multi-index access (AND & OR).

Example: The primary key of the table TASKTAB is a composite key with columns CLIENT_ID, PROJECT_ID, MODULE_ID.

Direct Index Look-Up

Values are provided for each column in the index. In the above example, if the value for all the three columns are given in the where clause, then there will be a direct index look up.

SELECT MODULE_ID, MODULE_STATUS FROM TASKTAB
WHERE CLIENT_ID = 100 & PROJECT_ID = 10 AND MODULE_ID = 1.

If only CLIENT_ID & MODULE_ID is given then direct index look-up is not possible. Index scan would occur.

Matching Index Scan or Absolute Positioning.

It starts with root page & works down to a leaf page in much the same manner as direct index loop up does. However since the full key is unavailable, DB2 must scan the leaf pages using the value it does have, until all matching values have been retrieved. As it starts at root page, the value of first column(s) should be given.

SELECT MODULE_ID, MODULE_STATUS FROM TASKAB
WHERE CLIENT_ID = 100 AND PROJECT_ID = 10

The query use matching index scan & returns all the module id & module status for the given client & project.

Non-matching index scan or Relative positioning.

Non-matching index begins with the first leaf page in the index & sequentially scans subsequent leaf pages applying the available predicates. If any ORDER BY or GROUP BY is closed, then optimizer prefers non-matching index scan than table space scan.

SELECT CLIENT_ID, MODULE_ID, MODULE_STATUS FROM TASKAB
WHERE PROJECT_ID = 10
ORDER BY MODULE_ID.

Index-only access

If all the data required by the query is available in index, then DB2 can avoid reading the data page & perform non-matching index scan. This special type of non-matching index scan is called index-only access.
SELECT CLIENT_ID FROM TASKAB WHERE PROJECT_ID = 10 AND MODULE_ID = 1.

Sequential pre-fetch & List pre-fetch

Sequential pre-fetch is a technique used by optimizer during SEQUENTIAL table space SCAN. It can be though of as a read ahead mechanism invoked to pre-fill DB2 buffers so that data is already in memory before it is requested.

List pre-fetch is a technique used by optimizer during INDEXED access. If the cluster ratio of the index is poor (less than 80%) then RID's of matching indexes are sorted based on data on data page number & then the data-page is accessed.

This significantly reduces the I-O as same page will not be read second time in this method. Clustered index access doesn't use this technique as already the data rows are in the same order of index. (Cluster ratio is1).

Multi index access.

When more than indexes are used in where clause.

1. Using the first index, RID's matching the requirement are chosen & sorted based on data-page number.
2. Using the second index, RID's matching the requirement are chosen & sorted based on data-page number.
3. If the indexes are joined using OR, then the union of RID list of 1 & 2 is used to access data page.
4. If the indexes are joined using AND, then the intersection of RID list of 1 & 2 is used to access data page.

Table Space Scan

If there is no index-able column in the predicate or DB2 decides not to use available index, then the optimizer choose table space scanning.

Table space scanning is like sequential read of file. All the rows of all the pages are read sequentially & the rows satisfying the selection criteria in the where clause is chosen for further processing.

Query Parallelism.

This is another technique that can be applied by the optimizer is query parallelism. When parallelism is invoked, an access path can be broken up into parallel groups. Each parallel group represents a serious of concurrent operations with the same degree of parallelism. Degree of parallelism refers to the number of concurrent tasks used to satisfy the query.

Query I/O parallelism (as of DB2 V3):

Breaks the data access for the query into concurrent I/O streams executed a parallel should reduce the overall elapsed time for the query. With query I/O parallelism, DB2 is limited to operate on a single processor for each query.

Query I/O parallelism (as of DB2 V4):

The large query is decomposed into multiple smaller queries that can be executed concurrently with one another on multiple processors. CP parallelism always uses I/O parallelism. This should further reduce the elapsed time of the query.

Query Sysplex parallelism (as of DB2 V5):

If further enhances parallel operations by enabling s single query to be broken up & run across multiple DB2 subsystems within a data-sharing group. It takes the advantage of the processing power of multiple Db2 subsystems.

Filter factor.

Filter factor is a ratio that estimates I/O costs. The optimizer calculates the filter factor for a query's predicates based on the number of rows that will be filtered out by the predicates. That is, if the predicate is most restrictive, the filter factor would be low, the I-O cost would be less & the query will be more efficient. For an indexed column, FIRSTEYCARDF column of SYSIBM.SYSINDEXES consists of number of distinct values in the column.

If the predicate is column = value then filterfactor = 1/FIRSTKEYCARDF
If the predicate is column <> value then filterfactor = (1-1/FIRSTKEYCARDF
This case involves more I-O involved & inefficient.

EXPLAIN

We have already seen that DB2 optimizer chooses the access path for the query. If we want to know what access path DB2 chose during the plan preparation, then the request should be places to Db2. the request can be done two ways:

Method 1: Use EXPLAIN (YES) parameter in the BIND card.
SELECT * FROM ownerid PLAN_TABLE ORDER BY APPLNAME, COLLID, VERSION, PROGNAME, TIMESTAMP DESC, QUERYNO, QBLOCKNO, PLANNO.

Method 2: Use the following command directly in the program or in SPUFI or QMF.
Syntax: EXPALIN ALL SET QUERYNO = integer FOR SQL statement.

Before posting the request, there should be a PLAN_TABLE under the user_id, based on model SYSIBM.PLAN_TABLE. PLAN_TABLE is a standard table that must be defined with predetermined columns, data type & length. During bind process, DB2 optimizer briefs the access path chose by it in the PLAN_TABLE.

If you want to query the access path for single query then use the query below:
SELECT * FROM PLAN_TABLE WHERE QUERYNO=integer ORDERBY QBLOCKNO, PLANNO.

Now the access path taken by the optimizer is loaded into PLAN_TABLE. One should know the meaning of PLAN_TABLE columns & values to understand the access path. DB2V5 has 46 columns in PLAN_TABLE.

PLAN_TABLE Column name	Meaning
QUERYNO	Meaning value assigned by the user issuing the EXPALIN or DB2.
QBLOCKNO	Integer identifying level of sub-query or union in a SQL statement. The first sub-select is numbered as 1, the second as 2 & so on.
APPLNAME	Name of the PLAN or PACKAGE based on BIND stage of static SQL. It will be spaces for SPUFI request.
PROGNAME	Name of the program in which SQL statement is embedded. DSQIESQL for SPUFI EXPALIN
PLANNO	Integer value saying the step of the plan in which QBLOCKNO is processed. (The order in which plan steps are undertaken)
METHOD	Integer value identifying the access method used for the given step. 1 – Nested loop join 2 – Merge scan join 3 – Independent sort as a result of ORDERBY, GROUP BY, DISTINC or IN 4 – Hybrid Join.
ACCESSTYPE	Technique used for access. I – Indexed access I1 – One fetch index scan R – Table space scan N- Index access with in Predicate M – Multiple Index Scan MX – Specification of the index name for multiple index access. MI – Multiple index access by RID intersection. MU – Multiple index access by RID union.
MATCHCOLS	Number of index columns used in INDEXED access. Access type should be I, I1, M, MX.
INDEXONLY	‘Y’ indicates access to the index is sufficient to satisfy the query.

PREFETCH	Indicates which type of pre-fetch will be used. S-Sequential, L-List, Blank-No or determined at run time.
PARALLELISM MODE	Indicates type of parallelism used at bind time. I – I/O Parallelism, C-CPU parallelism, X-Sysplex Parallelism, Blank – No or determined at run time.
QBLOCK_TYPE	Type of SQL operation performed for the query block. CURSOR – Correlated sub query. NCOSUB – Non correlated sub query UPDCUR, DELCUR – Update & Delete using WHERE CURRENT OF option of cursor. SELECT, INSERT, UPDATE, DELETE – Type of DML. SELUPD – SELECT with FOR UPDATE OF.
JOIN_TYPE	Indicates type of join implemented. F – Full Outer Join L – Left Outer Join (or converted right outer join) Blank – Inner join or no join.
TIMESTAMP	Date & Time the EXPLAIN for this row was issued.

Other columns are:
CREATOR, TNAME, TABNO, ACCESSCREATOR, ACCESSNAME, SORTIN_UNIQ, SORTIN_JOIN, SORTIN_ORDERBY, SORTIM_GROUPBY, SORTC_UNIQ, SORTC_JOIN, SORTC_GROUPBY, SORTC_GROUP, TSLOCKMODE, REMARKS, COLUMN_FN_EVAL, MIXOPSEQ, VERSION, COLLID, ACCESS_DEGREE, ACCESS_PGROUP_ID, JOIN_DEGREE, JOIN_PGROUP_ID, SORTC_PGROUP_ID, SORTIN_PGROUP_ID, MERGE_JOIN_COLS, CORRELATION_NAME, PAGE_RANGE, GROUP_MEMBER, IBM_SERVICE_DATA, WHEN_OPTIMIZE, BLIND_TIME.

DYNAMIC SQL

Static SQL is embedded in application program & the only values that can change are the value of the host variables in the predicated. Dynamic SQL are characterized by the capability to change columns, table & predicates during a program's execution.

Advantage: Flexibility & best access path (as the bind happens at the run time using latest RUNSTATS information)

Disadvantage: Slow as the runtime is bind time + execution time.

Types of Dynamic SQL

1. EXECUTE IMMEDIATE

   Move the SQL statement to be executed into the host variable & issue execute immediate command.

```
01   WS-HOST
     49   WS-HOST-LENGTH      PIC S9(04) COMP.
     49   WS-HOST-VAR         PIC X(40).
```

MOVE +40 TO WS-HOST-LENGTH
MOVE "SEL EMP_NAME = 'MUTHU' WHERE EMP_NO=2052" TO WS-HOST-TEXT.
EXEC SQL EXECUTE IMMEDIATE: WS-HOST-VAR END-EXEC.

Disadvantages:

- It cannot be used for SELECT.

- Executable form of the SQL will be deleted once it is executed.

2. EXECUTE WITH PREPARE

   This is equivalent to first form but here the SQL is prepared before execution & so the second disadvantage is avoided in this method.

Form: 1

```
MOVE +40 TO WS-HOST-LENGTH
MOVE “SEL EMP_NAME = ‘MUTHU’ WHERE EMP_NO=2052” TO WS-HOST-TEXT.
EXEC SQL PREPRE RUNFORM FROM: WS-HOST-VAR END-EXEC.
EXEC SQL EXECUTE RUNFORM END-EXEC.
```

Form: 2

Parameter markers can be used in place of constant values. This acts like placeholder for the host variables in the SQL.

```
MOVE +40 TO WS-HOST-LENGTH
MOVE “SEL EMP_NAME = ‘MUTHU’ WHERE EMP_NO=?”  TO WS-HOST-TEXT.
EXEC SQL PREPRE RUNFORM FROM: WS-HOST-VAR END-EXEC.
MOVE 2052 TO WS-EMP-NUM
EXEC SQL EXECUTE RUNFORM USING: WS-EMP-NUM END-EXEC.
```

Disadvantage:
Select statement is not supported.

3. SELECT with STATIC columns (fixed-list SELECT)

   The following method is used for data retrieval but the column to be selected are know & unchanging.

   Let the query be ‘SELECT EMP_NAME FROM EMPTABLE WHERE DEPT_NO=?’

```
MOVE ‘select SQL’ TO WS-HOST.
EXEC SQL DECLARE CSR1 CURSOR FOR SELSQL END-EXEC
EXEC SQL PREPARE FROM :WS-HOST END-EXEC
MOVE 100 TO WS-DEPT-NO
EXEC SQL OPEN CSR1 USING: WS-EMP-NO END-EXEC
PERFORM UNTIL SQLCODE = 100
    EXEC SQL FETCH CSR1 INTO : WS-EMPNAME END-EXEC
END-PERFORM.
EXEC SQL CLOSE CSR1 END-EXEC.
```

Disadvantage:

1. Table name cannot be changed. Number of columns cannot be modified during run time.

The fourth type of dynamic SQL known as ‘varying-list select’ provide hundred percent flexibility where the number of columns & even the table we are accessing can be changed during run time. The used the copybook SQLDA for its purpose.

INTEGRITY

Integrity means the accuracy, correctness or validity of data contained in the database. Maintaining integrity is not an easy task in a multi user environment. So the task maintaining integrity is handled by the system than the user.

Domain Integrity.

It ensures that a value in a column is a member of column's domain or legal set of values. Simple example is alphabetic data on the integer column is restricted.

Entity Integrity.

It means every row in a table is unique. To ensure entity integrity, developer has to define a set of column(s) as primary key.

Referential Integrity.

It ensures the data integrity between the tables related by primary (parent) & foreign (child) keys.

Concurrency & its issues:

DB2 allows more than one application to access the same data at essentially the same time. This is known as concurrency. Concurrency results the following problems.

1. The LOST UPDATE problem.

Time	Transaction-A	Transaction-B
T0	Read row R1	
T1		Read & Update row R1.
T2	Rollback	

Transaction-B overwrites the change done by Transaction-A at T2. In other words the update of transaction-A is lost.

2. Uncommitted Dependency Problem.

   It arises when a transaction is allowed to retrieve (update) a row that has been update by another transaction & has not yet been committed by the other transaction.

Time	Transaction-A	Transaction-B
T0	Update row R1	
T1		Read & Update row R1.
T2	Rollback	

Transaction-B updates row R1 based on the uncommitted change done by Transaction-A. Transaction-B rolled back its changes.

3. Data inconsistency.

   Let us say there is 2 accounts A1, A2 & account A3 should have sum of amounts in A1 & A2. Say A1 = 1000 A2 = 2000 A3 = 3000.

Time	Transaction-A	Transaction-B

T0	Read row A1. (1000)	
T1		Read row A1 & Update to 2000.
T2	Update row A2. (3000)	
T3	Calculate A3 & update. (4000)	
T4	Commit	Commit

Now the value of A1 = 2000 A2 = 3000 but A3 = 4000 ➔ Data inconsistent as A3 is expected to be 5000 as per design.

Locking

Locking solves the concurrency issues described above. Locking prevent another transaction to access the data that is being changed by the current transaction. Three main attributes of lock are mode, size & duration.

Mode of the lock:

It specifies what access to the locked object is permitted to the lock owner & to any concurrent application process.

Exclusive (X) – the lock owner can read or change the locked page. Concurrent processed cannot acquire ANY lock on the page nor do they access the locked page.

Update (U) – The lock owner can read but cannot change the locked page. However the owner can promote the lock to ‘X’ & update. Processes concurrent with the U lock can acquire ‘S’ lock & read the page but another ‘U’ lock on the page.

Share(S) – The lock owner & any concurrent processes can read but not change the locked page. Concurrent processes can acquire S or U locks on the page.

The above three are with respect to pages. Three more kinds of locks are possible at table/table space level. They are: Intent Share (IS), Intent Exclusive (IX) & Share/Intent Exclusive (SIX).

SIZE of the lock

It can be ROW, PAGE, TABLE & TABLESIZE.

Duration of the lock

It is the length of time the lock is held. It varies according to when the lock is acquired & released. This is controlled by ACQUIRE, RELEASE & ISOLATION LEVEL parameter of BIND.

LOCK Issues

Locking is introduced to avoid concurrency issues. Locking resulted suspension, timeout & deadlock problems. But LOCK is needed for data integrity.

Suspension.

IRLM suspends an application process if it requests a lock on an object that is already owned by another application process & cannot be shared. The suspended process is said to be in ‘Wait Stage’ for the lock. The process resumes when the lock is available.

Timeout

When the object is in wait stage more than predefined time, then the process (program) is terminated by DB2 with SQL return code -911 or -913 meant or TIMEOUT. The IRLMRWT of DSNZPARM determines the duration of the wait time.

Deadlock

When two or more transactions are in simultaneous wait stage, each waiting for one of others to release a lock before it can proceed, DEADLOCK occurs.

If deadlock occurs, transaction manager of DB2 breaks it by terminating one of the transaction & allows the other to process. Later the application programmer will restart the terminated transaction.

Lock & Latch

A true lock is handled by DB2 using an external component called IRLM. However, when it is practical, DB2 tries to lock pages without going to IRLM. This type of lock is known as LATCH. Latches are internally set by DB2 without going to IRLM.

Initially latches are used to lock only DB2 index pages & internal DB2 resources. DB2 V3 uses latching more frequently for data pages also. Latches are preferred when a resource serialization is required for a short time. Locks & Latches guarantee data integrity at the same level.

Advantage of Latch over Lock:

1. Cross-memory service calls to IRLM are eliminated.
2. Latch requires about one third the numbers of instructions as a lock.

Lock Promotion

When lock is promoted from lower level to upper level, it is called lock promotion. DB2 acquires ‘U’ lock to update a row. But at the same concurrent processes can access the row in ‘S’ lock. During update, it needs ‘X’ lock & to get the ‘X’ lock, it waits for concurrent processes to release ‘S’ locks. It promotes the current lock from ‘U’ to ‘X’ & this is called LOCK PROMOTION ON MODE. (‘U’ – Update in pending & ‘X’ – Update is progress)

When the number of page locks exceeds predefined limit, it is promoted to table or table space level & this is LOCK PROMOTION ON SIZE.

Lock Strategy

DB2 decides the lock to be applied based on the following:

1. Lock Size declared at the time of table space creation.
2. Type of SQL statement – SELECT or UPDATE.
3. Explicit LOCK statement in the program.
4. ACQUIRE & RELEASE option chosen by user at the BIND time.
5. Isolation level specified at bind time.
6. The NUMLKTS & NULLKUS limit.

System Catalog Tables

DB2 catalog is contained in a single database (DSNDB06). The 54 tables in the DB2 catalog collectively describe the objects & resources available to DB2.

Prior to Version5, there was an additional database called Communication Database (CDB) was used for establishing & documenting distributed DB2 connections. In the version 5, it has been renamed & rolled into DB2 catalog.

Each & every table has its own importance. So explaining all the tables & the columns in it is highly impossible. Refer the query section to know about some of the frequency used queries on SYSIBM tables.

1. SYSIBM.SYSDUMMY1 is a dummy table. When you want to do some arithmetic operation or retrieving system information like current-date, you can query this table.
   SELECT 10.25*20.25 FROM SYSIBM.SYSDUMMY1;

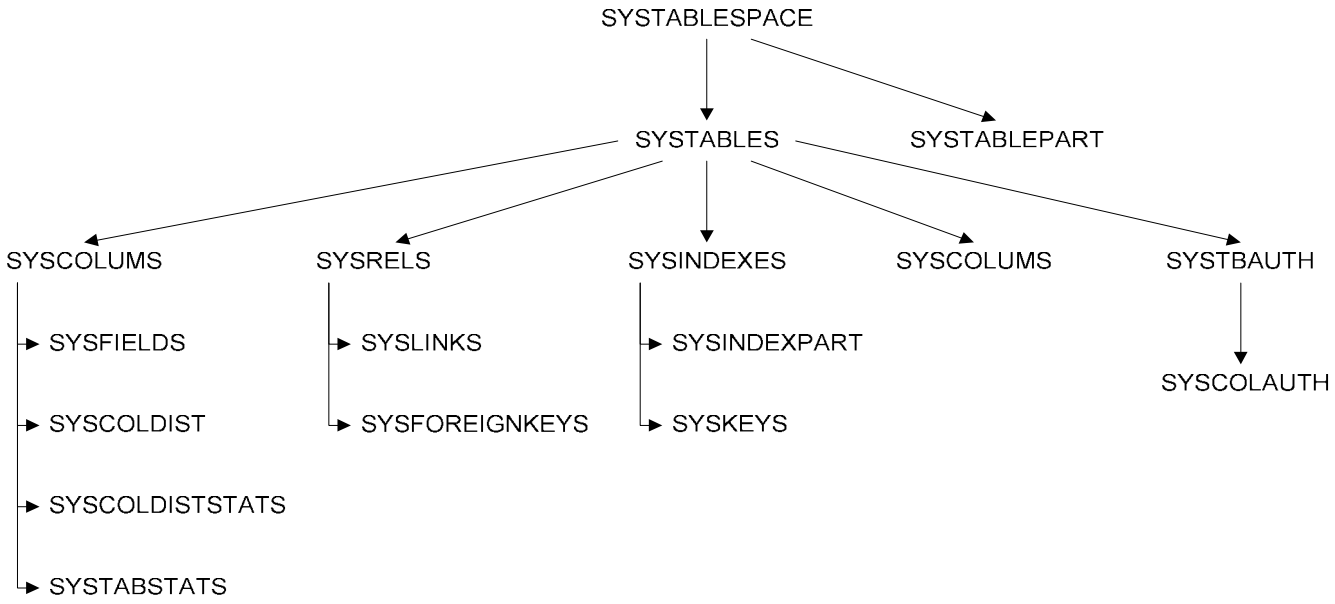
2. When a DBRM is bound to PLAN, all the SQL statements are places into SYSYTMTDB2 table. When a DBRM is bound to PACKAGE, all of its SQL statements are placed into SYSPACKSTMT table. Information about the DBRMS that were bound to PLAN or PACKAGE, are stored in SYSDBRM table. DBRM just created but not yet bound is not registered anywhere. (Remember, pre-compiler don't use DB2 at all & DBRM is an output of pre-compiler).

3. Information about plan is stored in SYSDBRM, SYSPACKAUTH, SYSPACKLIST, SYSPLAN, SYSPLANAUTH, SYSPLSYSTEM, SYSSTMT & SYSTABAUTH.

4. Information about the package is stored in SYSPACKAGE, SYSPACKAGE, SYSPACKAUTH, SYSPACKDEP, SYSPLSYSTEM, SYSSTMT & SYSTBAUTH.

5. DB2 Catalog stores only information about the PLAN. The executable form of PLAN, called Skeleton cursor table (SKCT), is stored in the DB2 directory in the SYSIBM.SCT02 table.

6. Information about Table space/Table & indexes are stored in the following Tables.

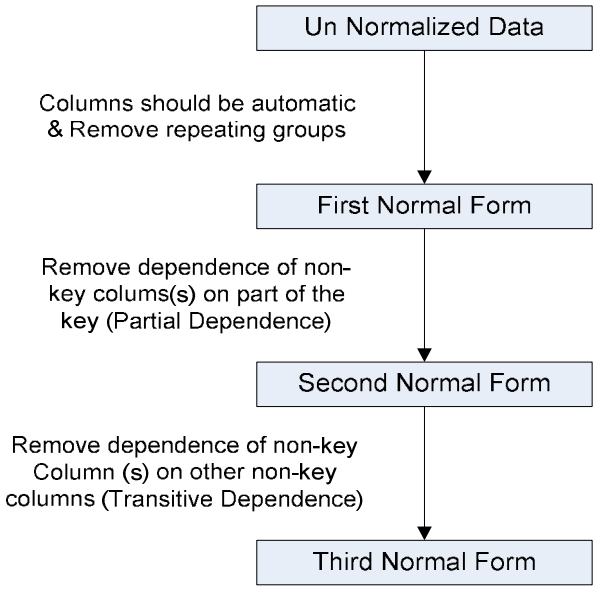


Normalization

Data normalization describes the process of designing a database & organizing data to take best advantage of relational database principles. It is the process of putting one fact in one appropriate place. This optimizes the update at the expense of retrievals.

When a fact is stored in only one place, retrieving many different but related facts usually requires going to many different places. This tends to slow the retrieval process but update is easy & faster as that fact you are updating exists in only one place.

The process involves five levels. Tables are usually normalized till level three.



The three levels can be memorized using the statement below:

"The values in a row are dependant on the key, the whole key & nothing but the key, so help me CODD".

In the fourth level, multi-values dependencies are removed & in the fifth level, remaining anomalies are removed.

De-normalization

De-normalization is the process of putting one fact in more than one place. It is the reverse process of normalization. This will speed up the retrieval process at the expense of data modification. De-normalization is not a bad decision when a completely normalized design is not performing optimally.

Triggers (Available in version 6)

Trigger is a piece of code that is executed in response to a data modification statement (insert / update / delete). To be a bit more precise: triggers are event-driven specialized procedures that are stored in & managed by the RDBMS. Each trigger is attached to a single, specified table. Triggers can be thought of as an advanced from of "rule" or "constraint" written using an extended form of SQL. Therefore triggers are automatic, implicit & non-bypass able.

Nested Triggers

A trigger can also contain insert, update & delete logic within itself. Therefore, a trigger is fired by a data modification, but can also cause another data modification, thereby firing yet another data modification, thereby firing yet another trigger. When a trigger contains insert, update, and/or delete logic, the trigger is said to be a nested trigger. Most DBMSs, however, place a limit on the number of nested triggers that can be executed within a single firing event. If this were not done, it could be quite possible to having triggers firing triggers ad infinitum until all of the data was removed from an entire database!

Triggers cannot be attached to the following tables:
A system Catalog Table, PLAN_TABLE, STATEMENT_TABLE, DSN_FUNCTION_TABLE view, Alias, Synonym, Any table with a three part name.

Trigger in place of RI.

Triggers can be coded, in lieu of declarative RI, to support ALL of the RI rules. We can specify only ON DELETE rules in the DDL statement. UPDATE & INSERT rules of RI can be implemented using triggers.

Sample Trigger:

```
CREATE TRIGGER SALARY_UPDATE
    BEFORE UPDATE OF SALARY ON EMP
    FOR EACH ROW MODE DB2SQL

WHEN (NEW.SALARY > (OLD.SALARY * 1.5))
BEGIN ATOMIC
    SIGNAL SQLSTATE '75001' ('Raise exceed 50%');
END;
```

The trigger executes once for each row. So if multiple rows are modified by a single update, the trigger will run multiple times, once for each row modified. Also, the trigger will be run BEFORE the actual modification occurs. Finally, take special notices how NEW & OLD are used to check values before & after the update.

Triggers Vs Stored Procedures.

Triggers are also similar to stored procedures. Both consist of procedural logic that is stored at the database level. However, stored procedures are not event-driven and are not attached to a specific table. A stored procedure is explicitly executed by invoked a CALL to the procedure (instead of implicitly being executed like triggers). Additionally, an asked procedure can access many tables without being specifically associated to any of them. DB2 has supported stored procedures since version 4.

DB2 Utilities

Most of the time, application programmer doesn't need any knowledge on DB2 utilities. Backup, Reorganization, Recovery & maintenance of database are taken care by DBA of the project.

We are briefing the DB2 utilities just too familiar with it.

DSNUPROC is the DB2 catalogued procedure that executes the program DSNUTLIB. DSNUTLIB is a master program & it calls other programs based on the first word or control card. The first word of control card should be utility that needs to be invoked.

Most of the DB2 utility programs are divided into four major categories.

Utility	Purpose
COPY (Backup & Recovery)	It is used to create image copy backup dataset for a complete table space or a single partition of a table space. It can be of type FULL or incremental image copy. Successful copy details are loaded in SYSIBM.SYSCOPY.
MERGECOPY (Backup & Recovery)	It is used to create full image copy from incremental image copies.
QUIESCE (Backup & Recovery)	It is used to record a point of consistency for related application or system tables. Usually done before image copy.
RECOVER	It is used to restore DB2 table space & indexes to a specific point in time. It uses

(Backup & Recovery)	image copy & DB2 logs information to roll back the table space content to the old one. RECOVER INDEX generated new index data from current table space data. It is actually regeneration than restoring. So it is followed by RECOVERY TABLESPACE which is restoring.
LOAD (Data Organization)	It is used to accomplish bulk inserts to DB2 tables. It can add rows to a table retaining the current date or it can replace the existing data with new data.
REORG (Data Organization)	It is used to re-clustering the data in table space & resetting the free space to the amount specified in the CREATE DDL & deletes & redefining VSAM datasets for STATGROUP defined objects.
CHECK, REPAIR, REPORT, DIAGNOSE	These are data consistency utilities. They are used to monitor, control & administer data consistency errors.
RUNSTATS	It collects statistical information for DB2 tables, table spaces, partitions, indexes & columns. It places this information into DB2 catalog tables. The DB2 optimizer uses the data in these tables to determine optimizer uses the data in these tables to determine optimal access path for the SQL queries. CATMAINT, MODIFY & STOSPACE are the other catalog manipulation utilities.

How the load card of LOAD utility looks?

```
LOADS DATA INDDN (SYSREC)
RESUME YES
ENFORCE CONSTRAINTS
INTO TABLE DB1.EMPTABLE
(EMPNAME POSITION (1) CHAR (20),
 EMPADDRESS POSITION (25) VARCHAR
 EMPPROJECT POSITION (50) CHAR (3))
```

RESUME YES ➔ Resumes all the records from first to last.
RESUME NO ➔ Table should be empty.
REPLACE ➔ It overrides the previous records set with current record set.

```
 1                25                50
XXXX              XXXXXXXXXXXXX      XXXXX
YYYY              YYYYYYY            YYYYY
```

LOAD Data statement describes the data to be loaded. In the above example, SYSREC is coded & it should be one of the DDNAME in JCL, which points to actual dataset.

Data are loaded in the order they appear in the dataset. Automatic data conversion between compatible data types is done if needed.

What are COPY PENDING & CHECK PENDING status?

COPY PENDING & CHEC PENDING status on the table space restricts any updates on table space.

If you LOAD or RESORG the table space with the option LOG NO, then the table space get into COPY PENDING status. The meaning is, an image copy is needed on the table space.

If you LOAD the table space with ENFORCE NO option, then the table space get into CHECK PENDING status. The meaning of table space is loaded without enforcing constraints. CHECK utility needs to be run on the table space.

COPY Utility

COPY TABLESPACE EDSDB.TRG007TS
FULL YES|NO YES FOR FULL IMAGE & NO FOR INCREMENTAL
SHRLEVEL CHANGE

MERGECOPY Utility

MERGECOPY TABLESPACE EDSDM.TRG007TS
NEWCOPY (YES/NO)
YES. Merging all the incremental copies with the available full image copy & create a new image copy.
NO. Merging all the incremental copies & prepares a single incremental image copy.

SQL – Guidelines for better performance

1. Pre-test all your queries in SPUFI or QMF before placing in program.
2. Avoid qualifiers for the tables in the program. Give it on the BIND card.
3. Unless really needed, avoid usage of DISTINCT, UNION & ORDER BY.
4. Avoid WHENEVER.
5. Code predicates on index-able columns – code the most restrictive predicated first – Use equivalent data types in the predicates.
6. Do not use arithmetic expressions in a predicate – DB2 will not use index in this case.
7. Prefer BETWEEN is stead of '<=' and '>='
8. Prefer 'IN' in stead of 'LIKE'.
9. Avoid the usage Not. (Except with NOT EXISTS).
10. Join using SQL instead of programming logic.
11. Use JOINS than sub-query.
12. Minimize the number of tables in the JOIN.
13. When a SELECT statement is coded, use FOR FETCH ONLY. This enables block fetch option DB2 & lead to efficient retrieval.
14. If the cursor is used for update, use FOR UPDATE OF clause in DECLARE statement itself.
15. Though DECLARE CURSOR can be given in procedure division, code it in working storage section as a good practice. DECLARE cursor is not an executable statement & procedure division is for executable statements.
16. For existence checking, use constant. (WHERE EXISTS (SELECT 1 FROM TAB1).
17. Give a thought on dropping indexes before large insertion. It avoids unorganized indexes & poor performance.
18. Don't USE select * - If there is any change in DB2 table structure, then the program needs to be modified though it doesn't need the field. There is unnecessary I-O overhead.
19. Do the arithmetic operations while selection the column(s) than doing in programming language?

CICS (**C**ustomer **I**nformation **C**ontrol **S**ystem).

History
IBM launched the initial version of CICS in 1968.
It is a Database/Data Communication control System where an application program can concentrate on the application processing without worry about OS, hardware and others. Initially CICS was on macro and later upgrades to command level.

This book is written on CICS/MVS V2R3.

Task & Transaction

Task is a unit of work & transaction is an entry that initiated the execution of task. The transaction identifies the transaction in CICS.

Conversation & Pseudo Conversation.

Program can communicate with user by a pair of SEND & RECEIVE commands. But in between the SEND & RECEIVE, that is during the human response time, all the resources are held by the program.

Once the user entered the information in the screen, the program proceeds further. This mode of communication is called Conversational mode. It is un-famous for the resource wastage.

In Pseudo conversation mode, whenever there is a need for conversation with user, the program logically terminates there, releases the resources held by it & pass control information to next transaction & the next transaction automatically started once user entered the information in the screen.

It is actually multi task operation but looks like conversation from user point of view.

For easy understanding we would say the communication in telephone line is conversational mode (Telephone line is in usage through out the communication.) IRC as Pseudo conversation (Once the message sent, the line is freed and it again gets the resource when the other side replied.)

Multitasking & Multithreading

Operating systems allows execution of more than one task concurrently & this is called Multitasking. If the one or more concurrent tasks use the same copy of the program, then is called Multithreading. So it is obvious that multitasking is a Subset of multithreading.

Reentrant & Quasi- Reentrant

If the same copy of the program is used by multiple tasks, then the system should take care of to proper reentrance after the SVC/CICS interruption.

Reentrant program is defined as a program that does not modify itself so that it can reenter to itself & continue processing after an interruption by the SVC call of OS. Batch program are non-reentrant. Reentrancy under CICS environment is called quasi entrant as the interruption in CICS may involve more than once SVC calls or no SVC at all.

COBOL program should be complied with RENT option for reentrancy in Multithreading environment.

CICS Control Program & Control Tables

CICS Nucleus consists of IBM-Supplied CICS control programs & corresponding user-specified CICS control Tables. The important tables with respect to development point of view are given below:

Control Table	Function
PCT	The transaction & the main program associate with the transaction should be registered in program table. Task control Program(KCP) refers PCT.
PPT	All the CICS programs & maps have to be registered in Processing Program Table. File Control Table(PCP) refers PPT.
FCT	All VSAM file used in the CICS program has to be registered in File Control Table. File Control Program (FCP) refers FCT.
DCT	Transient data queues should be predefined in Destination Control Table. Transient Data Program refers DCT.
TST	If you want to recover Temporary storage queues during system crash, then they should be registered in Temporary Storage Table.
RCT	If any DB2 commands are used in the program, then the PLAN should be registered here.
SNT	User-ID & Password should be registered in the Sign On Table.
TCT	All the terminals should be registered in Terminal Control Table.
PLT	All the program that need to be automatically started during CICS start up & Shut down should be listed in Program List Table.
JCL	Control Information of system logs & journal files is stored in Journal Control Table. Journal Control Program refers to JCT.

CICS-DB2-COBOL Program – Compilation.

COBOL program can’t recognize CICS commands. So all the CICS commands are coded within EXEC CICS & END-EXEC scope. The program is first passed to CICS Translator & the translator will convert all the CICS commands to COBOL call statements & this modified source is passed to COBOL compiler.

IF your program has DB2 commands then the sequence of preparing load module would be DB2 pre-complier, CICS Translator, COBOL Compiler & Link editor. Logically speaking the order of DB2 pre-compiler & CICS translator can be revered also. But as a convention we are first doing pre-compilation.

If translation is done first, CICS translator tries to recognize to DB2 statements & would issue diagnostic messages. The other reason is most of the compilation first, all the DB2 statements are converted into COBOL call statements in first phase itself & the transaction time would be less.

COBOL-CICS program should be compiled with RENT RESIDENT NODYNAM & LIB options.

General Syntax Of CICS Statement

```
EXEC CICS function
     [(option(argument value)]
     [(option(argument value)]
END-EXEC.
```

Restricted COBOL commands in CICS environment

OS SVC Triggering statements	ACCEPTS CURRENT-DATE DATE DAY DISPLAY
I/O statements	OPEN CLOSE READ WRITE REWRITE DELETE START
SORT statement	RETURN RELEASE

VS COBOL 2 allows STOP RUN & that returns control back to CICS.

MAP DESIGN

Before get into program design, let us see how maps (screens) are designed in CICS. Most of all installation use tools like SDF for screen designing. The tools generate BMS macros for the designed screen. We brief the BMS macros involved in the map design. BMS is acronym for Basic Mapping Support.

MAP & MAPSET

A screen designed thru BMS is called MAP. One or a group of maps makes up a MAPSET.

PHYSICAL MAP & SYMBOLIC MAP

Physical maps control the screen alignment, sending & receiving of CONSTANTS & data to & from a terminal. They are coded using BMS macros, assembled & link edited into CICS LOAD LIBRARY. They ensure the device independence in application programs.

Symbolic maps define the map fields used to store the VARIABLE data referenced in COBOL program. They are also coded used BMS macros. But after using assembling, they are place in a COPY library & then copied into CICS programs using COPY statement. They ensure device & format independence to the application programs.

BMS Macro Coding Sheet

Since BMS map definitions are purely assembler macros, the following coding convention must me maintained.

```
|        |     |                                                          |
1        10    16                                                        72
(1-8) Label  (10-15) Macro name       (16-71)Operands              72-continuation.
```

Macro name. There are three macros used in BMS coding.

1. DFHMSD – Defining Mapset.
2. DHMDI – Defining Maps.
3. DFHMDF – Defining Fields.

All the BMS coding starts with DFHMSD followed by line or more DFHMDI. One or more DFHMDF follow DFHMDI.

Label: Label is name of the operation. If the macro is DFHMSD, it specifies map-set name. If the macro is DFHMDF then is specifies field name.

Operands: Define the parameters for the macro being invoked.

Continuation: Continuation of macro is achieved by ‘X’ in 72nd column of the current line & continues from the 16th column of next time.

DFHMSD

It is used to define a mapset with its character & to end a mapset. So you will find two DFHMSD in any BMS coding. The important operands are below.

TYPE.

It should be DSECT for symbolic map generation, MAP for physical map generation & FINAL to indicate the end of mapset. Alternatively symbolic parameter & SYSPARM can be coded in the TYPE of defining DFSMSD & the value can be overridden in the PARM parameter of assembly procedure. This avoids the change in the BMS coding.

MODE.

IN for input maps like order entry screens & OUT for output maps like screens & INOUT for input-output maps like update screens.

LANG.

In specifies the language in which the symbolic map is to be generated. It can be COBOL, PLI, ASM or RPG.

STORAGE.

AUTO is used to acquire separate symbolic map area for each mapset. BASE=MAP-IOAREA allows multiple maps from more than 1 mapset to share same storage area. MAP-IOAREA will be redefined multiple times to achieve it.

TIOAPEX.

It should be 'YES' to reserve the prefix space of 12byte for BMS commands to access TIOA properly. This is required for command level CICS.

CTRL.

Device control request are place here. Multiple parameters are separated by comma. FREEKB is used unlock the keyboard. FRSET is used to reset the MDT of all the fields in all the maps to zero. ALARM is used to set an alarm at screen display time. PRINT is used to send the mapset to printer.

TERM.

If anything other than 3270 terminal is used for display of screens, then it should be coded here. This ensures device independence by means of providing the suffix. SUFFIX is used to specify suffix for the terminal & it should correspond to TCT entry of the terminal.

DFHMDI.

It is used to define a map with its characteristic in a mapset. There can ne any number of DFHMDI. Some of the important operands of DFHMDI are below:

SIZE - It has two arguments namely length & breadth & as a whole the size of the map is specified here.
LINE - The map starting line is mentioned here.
COLOUMN - The map starting column width the LINE is mentioned here.
JUST - RIGHT or LEFT is coded here to inform the justification of the map within mapset,

1. The above four parameters decides the size & location of map within map set CTRL & TIOPEX can be also coded in DFHMDI. Value of DFHMDI overrides the value of DFHMSD.

DFHMDF

It is used to define a field with its characteristic in a map. There can be any number of DFHMDF within DFHMDI. Some of the important operands of DFHMDF are below:

POS.

It has two arguments that decided the position of the fields. The two arguments are line & column. It is the position where the attribute byte of the fields starts.

LENGTH.

The length of the field is coded here. It excludes the attribute character.

ATTRIB.

All the input & output fields are prefixed by one byte attribute field that defines the attributes of the field. Some of the attributes are:

1. ASKIP/PROT/UNPROT – Mutually exclusive parameters that define the type of the fields. UNPROT is coded for input & output fields. PROT is coded for output & stopped field. ASKIP is coded for screen literals & skipper fields. The cursor automatically skipped to next fields & so you cannot enter data into skipper field.
2. NUM – 0-9, Periods & - are the only allowed characters.
3. BRT/NORM/DRK – Mutually exclusive parameter that define the intensity of the field.
4. IC – Insert Cursor. Cursor will be positioned on display of map. If IC is specified in more than one field of a map, the cursor will be place in the last field.
5. FSET – Independent of whether the field is modified or not, it will be passed to the program. MDT is set for the field.

JUSTIFY

RIGHT is default value. Code LEFT for numeric fields.

PICIN & PICOUT

If defines the picture clause of the symbolic map in COBOL & useful for numeric field editing.

INITIAL

The default value of the fields is coded here. When the MAP is sent, this value will appear in the field. The constant information like TITLE is coded using INITIAL keyword of field definition. To avoid data traffic, these constant information fields should not be coded without LABEL parameter. If there is no LABEL parameter, then symbolic map will not generated for those fields as they are unnamed fields.

How a field looks like in a Symbolic map

Let the label of one DFHMDF is EMPNAME in the map EMPDET & the length of it is 20. The respective symbolic map would look like follows.

```
01 EMPDETI.
      02 FILLER              PIC X(12)         TIOAPFX = YES creates this 12 byte filler.
      02 EMPNAMEL            PIC S9(04) COMP   Length field
      02 EMPNAMEF            PIC X             Flag byte
      02 FILLER REDEFINES EMPNAMEF.
            03 EMPNAMEA       PIC X             Attribute byte.
      02 EMPNAMEI            PIC X(20)         Actual field(Input)
Other fields…
01 EMPDFTOO REDEFINES EMPDETI.
      02 FILLER              PIC X(12)         TIOAPFX = YES creates this 12 byte filler.
      02 FILLER              PIC X(03).
      02 EMPNAMEO            PIC X(20)         Actual field(Output)
Other fields…
```

So four fields are generated for every named field in the BMS.

Fieldname+L.

It has length of the fields entered by the user during the input operation.

Fieldname+I.

It is the actual input field that carries the entered information. The value of this field is X’00’ if no data is entered. The space corresponds to X’40’.

Fieldname+A.

It is attribute byte. It defines the attributes of the field.

Fieldname+F.

If is a flag byte. It has X’00’ by default. It will be set to X’80’ if the user modifies the field but no data is sent. That is when the user pressed clear key over the field.

Fieldname+O.

This field should be populated in the program before sending the screen to display. Please note that the words INPUT(RECEIVE) & OUTPUT(SEND) are with respect to program.

Move on Attribute Field

We have just seen that attribute field is of 1 byte that specified the attributes of the field. 1 byte is of eight bits & the values in each bit has its own meaning.

For Example,
Bit 2 – ‘0’ indicates the field is unprotected & ‘1’ indicates the field is protected.
Bit 3 – ‘0’ indicates the field is alphanumeric & ‘1’ indicates the field is numeric.
Bit 2 & 3 – ‘11’ indicates that the field is Auto-skip.
Bit 4 & 5 – ‘00’ indicates Normal intensity & non-detectable.
‘01’ indicates Normal intensity & detectable.
‘10’ indicates High intensity & detectable.
‘11’ indicates Dark & Non-detectable.
Bit 7 – ‘0’ indicates MDT is OFF ‘1’ indicated MDT is ON.

Modified Data Tag(MDT).

MDT is 1 bit field of the attribute byte. The program can receive only the fields with MDT ‘1’ on RECEIVE. Effective use of MDT can reduce the data traffic drastically in the communication line.

MDT can be SET/RESET in the following ways.

1. When the used modifies the field, the MDT of the field is automatically set to ON.
2. CTRL=FRSET of DFHMSD or DFHMDI will RESET the MDT to ‘OFF’ for all the fields in the mapset or map. FSET keyword of the attribute operand definition for the specific field.
3. Before sending the screen, by overriding the MDT bit of attribute byte of field the MDT can be set to ‘ON’.

If you are specific on the values of some fields independent of whether the user has modified or not, code them with FSET. One good example is default values for the input fields (like Order received date). If the user finds default value (current date) in the screen & it is fine with his requirement, then he won’t touch the field & MDT will not be set. The program cannot receive the field as MDT is OFF. But actually the program needs this field. So this field should have been defined with FSET.

CURSOR Positioning

Positioning of cursor is an important area in screen design. By default, the cursor will be place in the first unprotected field.

Static Positioning

If IC is coded in the attribute operand of DFHMDF macro, then the cursor will be place in that field. If IC is coded in more than 1 field, then last field with IC will get the cursor.

Symbolic Positioning

Move -1 to length of the fields where you want place the cursor & send the map with CURSOR option. This is device independent method & recommended to control the cursor position dynamically.

Relative Positioning

Send the map with CURSOR(value) option. This will set the cursor at the position coded by 'value', relative to the first column of the screen. This is device dependant method of dynamic cursor positioning & not recommended. When there is change in screen layout, the program needs to be modified.

CURSOR(30) will place the cursor in the 30th column. (First column in ZERO).

Cursor Position

EIBPOSN of DFHEIBLK contains the offset of cursor position in the screen when the data was transferred to program from the terminal.(relative to zero). It is half word binary field.

Dynamic attributes setting

Any attribute of the field can be modified in the program by setting the bits of attribute field properly. Changing the attribute in the program in a dynamic method. CICS provides the standard attribute character list in the form of copybook. (DFHBMSCA). This can be copied into our application program using copy statement & we can use the variables in the copybook to change the attribute of a field in the program.

For example, if the information entered by the user in a particular field is failed in validation, then as a usual practice, in addition to throw an error message in the bottom of the screen, you may wish to hi-light the field that is in error.

MOVE DFHUNIMD to FIELD(A) ➔ will achieve this.

SKIPPER & STOPPED field

Skipped is unlabelled 1 byte field with the auto-skip attribute & stopped is unlabelled 1 byte field with protected attribute.

Skipped field skips the cursor to next field when the current field is filled up to its full length whereas stopped field stops entering any more character after the length of the field is reached. The user has to press RESET key to proceed further.

Skipped field is used to speed up the user entry & recommended to code at the end of each unprotected field. Stopper field is coded based on requirement & importance of the field. Stopper field usually follows the last unprotected field in the screen.

BMS INPUT-OUTPUT OPERATIONS

Basic BMS input-output operations are carried out using the following commands. RECEIVE MAP, SEND MAP, SEND CONTROL, SEND TEXT, SEND PAGE.

RECEIVE MAP

It is used to receive the information entered by the user into program. If INTO is not coded, then CICS automatically finds the symbolic map area (mapname+I) & places the mapped data. The values of the field can be read in the program by referring to filename+I.

EXEC CICS RECEIVE MAP (map name) MAPSET (mapset name) END-EXEC.

The option TERMINAL ASIS overrides the upper-case translation specified in the TCT. As we already discussed, if the user didn't modify any of the fields then MDT will be 'OFF' for all the fields. Zero bytes transmission results MAPFAIL exception error.

SEND MAP

This is used to send a map to a terminal. Before issuing this command, the application program should prepare the data in the symbolic map area. If FROM is not coded, then CICS automatically finds the symbolic map area (mapname+O) & takes the data to the terminal.

EXEC CICS SEND MAP (map name) MAPSET (mapset name) END-EXEC.

Other options in SEND MAP are:

ERASE: Erase the screen before displaying the MAP. When you first time throw the screen, it should be specified with ERASE. It will not be coded when you just want to display the

error message over the current screen. ERASEUP erases only the unprotected fields of the screen before displaying the MAP.

DATAONLY : Only symbolic map data is sent to the terminal.
MAPONLY : Only physical map data is sent to the terminal. FROM cannot be coded.

FREEKB, ALARM & FRSET can be also coded & the meaning is already discussed.

TEXT Building – SEND-TEXT, ACCUM & PAGING

Text streams can be sent to the terminal without any pre-defined BMS maps. This is called text building. Text streams can optionally have header & trailer.

Syntax the SEND TEXT command is:

```
EXEC CICS
      FROM(data-value)
      LENGTH(data-value)
      [HEADER(data-value)]
      [TRAILER(data-value)]
      [ERASE][ACCUM][PAGING]
END-EXEC.
```

When to code ACCUM?

The text stream can consist of multiple paragraphs & the building of message can happen in multiple stages. If you want to send the text stream only after the building of the entire paragraph being built. They are accumulated & will not be sent immediately.

Once the page is built with complete message of the entire paragraph, issue SEND PAGE command that will send the complete text stream to the terminal. HEADER is place in the first SEND TEXT & TRAILER is placed in the SEND PAGE.

When to code PAGING?

ACCUM alone Is enough if the text-stream can fit within a single page. But if the text stream is expected to exceed one page, then code PAGING in all the SEND-TEXT commands in addition to ACCUM option. The last SEND-PAGE command sends the first page of message to the terminal.

The user can enter paging commands or keys to walk through the pages. To use keys instead paging commands, the keys should be mapped with paging commands in the System Initialization Table(SIT). Usually there will be a mapping for PF7 & PF8 with page up & down commands respectively. PURGE MESSAGE is used to purge the message that is accumulated but not yet send.

Format of HEADER & TRAILER

If TEXT-HEADER is the variable used in HEADER command & the header is ‘ORDER INVENTORY – Page nn – ‘, then the variable TEXT-HEADER should be defined as follows in working-storage section.

```
01 TEXT-HEADER.
      05 FILLER         PIC S9(4) COMP VALUE 27.=>Length of the text.
      05 FILLER         PIC X VALUE ‘&’         =>Character identifying automatic
                                                  Page number in the text.
      05 FILLER         PIC X.                  => 1 byte control field used by BMS
      05 FILLER         PIC X(27) VALUE ‘ORDER INVENTORY – Page && - ‘.
                                                  =>Actual text in the header.
```

TRIALER also can be coded in the same way. Automatic –page-number- identifying character will be spaces for TRIALER working storage variable.

ACCUM & PAGING can be also used with SEND MAP & the meaning is same.

SEND-CONTROL

When the MAP is sent, we can specify device control options like FREEKB ALARM ERASE along with SEND-MAP. If one wants to issue the device control options before sending the map, SEND CONTROL will be useful. If is used to establish the device control options dynamically.

Syntax:
```
EXEC CICS SEND CONTROL
      [CURSOR(data value)] [ERASE|ERASEUP] [FREEKB] [ALARM] [FRSET]
END-EXEC.
```

PRINT Through BMS.

PRINT option of SEND MAP command allows the reports to be printed at the local printer connected to the terminal. In this case, MAP is sent to the printer & not to be terminal.

Syntax:
```
EXEC CICS SEND MAP ('MAP-NAME'
               MAPSET ('MAPSET-NAME)
               PRINT
               NLEOM
END-EXEC.
```

NLEOM option is recommended with print option. When this option is used,

1. BMS builds the data stream using new line characters & blank character to position the fields on the printer page.
2. The data stream is terminated by EOM (end of message) that stops printing. So printer buffer is efficiently used, allows larger pages to be printed from the same buffer that makes the printing faster.

Message Routing

A message can be routed to one or more terminals other than the direct terminal with which the program has been communicating. The message eligible for message routing is, a message constructed by the SEND MAP command with the ACCUM option.

ROUTE command establishes the message routing environment & the SEND PAGE command issued after ROUTE command sends the message to the destination.

Syntax:
```
EXEC CICS ROUTE [LIST(data-area)],[OPCLASS(data-area)],
                [INTERVAL(hhmmss)|TIME(hhmmss)],
                [TITLE(data-area)],[ERRTERM(name)]
END-EXEC.
```

LIST & OPCLASS name the route list & operator class codes respectively. INTERVAL/TIME determines the actual timing of message delivery in the time interval or the time respectively.
TITLE names the title field defines in the working storage section & ERRTERM specify the terminal ID where the error message (if any) to be sent.

Route list is terminal in working storage using the following convention. TTTTrrOOOsrrrrrr – 8 bytes named 'r' are declared as spaces. TTTT names the terminal identifier as in TCT & OOO specify the operator id as in SNT. S is status flag. Code as many 16 bytes fields as the destinations & indicate end of route list is with the declaration of half word binary field with -1 value.

The message can be routed to every terminal which users of the specified operator class are signed on. This is done using OPCLASS.

Pseudo logic in CICS program is achieved in the 3 ways.

1. Multiple programs & Multiple Transaction Ids.

The conversation program is logically & physically divided into multiple programs & each program is registered with one transaction ID. Each program issues RETURN command with next transaction ID. So after the screen is sent, the first program will be terminated. Once the user filled up the information in screen & press ENTER, the next program of the next transaction ID (returned by the first program) gets control & the same process is followed for n programs.

Advantage: Easy development & maintenance.

Disadvantage: Possible Code redundancy & number of PCT & PPT entries.

2. Multiple Transaction & Single Program.

First paragraph of the procedure division controls the flow the different sections of the program based on the transaction identifier stored in EIBTRNID. Every section ends with RETURN command with next transaction to be invoked on completion of the screen information by the user.

Thus the conversational program is logically divided than physical division.

Advantage: PPT entries are less compared to case 1.

Disadvantage: Number of PCT entries is still the same as case 1.

3. One Transaction & One Program. (Preferred way)

RETURN command has an optional parameter of COMMAREA & LENGTH. Using this parameter we can pass information between the transactions.

So first program of your program will check whether this is first entry or nth entry. If EIBCALEN is zero, this is first entry. So divert the program to the section that will do all initial processing & throw the first screen for the user.

In addition to this it updates COMMAREA with some control information like internal transaction identifier. This COMMAREA is passed along with RETURN TRANSID. So EIBCALEN is updated with LENGTH of COMMAREA. The second & nth entrances don’t have EIBCALEN as zero & so based on the information in the COMMAREA, transfer the control of different section of the program.

Task Initiation Process

1. User entered transaction identifier.
2. Terminal Control Program along with Terminal Control Table recognizes the incoming data from the terminal.
3. Storage Control Program acquires the storage for the terminal input output area (TIOA).
4. Terminal Control Program places the data from the terminal into TIOA & sets the pointer into TCT entry of the terminal.
5. If there is no task is associated with the terminal, then TCP gives the control to task Control Program, which realizes the transaction identifier in the data in TIOA. (KCP)
6. SCP acquires the storage area for task control area (TCA) in which KCP prepares the control data for this task.
7. KCP refer PCT to identify the program associated with the transaction.
8. Program Control Program (PCP) loads the program into main memory from the load library & gives the control back to KCP.
9. KCP gives the control application program loaded into main memory.
10. Program started processing a Task is initiated.

Ways of Data Passing between transactions.

i. By COMMAREA.

In the example below, Working storage item WS-ITEM of length WS-LENGTH is passed to the transaction EMPC. The program for the transaction ‘EMPC’ should have DFHCOMMAREA of size WS-LENGTH in the linkage section to receive the information passed by RETURN of this program.

```
            EXEX CICS
                  RETURN TRANSID('EMPC') COMMAREA(WS-ITEM) LENGTH(WS-LENGTH)
            END-EXEC.
```

WS-LENGTH is full word binary & the maximum length that can be specified is 64K. So we can pass data of maximum size 64K. But it is recommended not to pass more than 24k.

LINK & XCTL can also have use COMMARA to pass the information to the program being called & the concept is same.

ii. Queues.

There are 2 types of queues TSQ & TDQ. TSQ can be used as scratch pad in main memory. You can write as much as you want in TSQ & they will be available to all the transaction that is aware of the queue name. TSQ is primarily used to share huge amount of information across the transaction. Refer queues section for more details.

iii. TCTUA, TWA, CWA.

TWA – Transaction Work Area – One per Task – Work area associated with the task.

TCTUA – Terminal Control Table User Area – Work area associated with a terminal & defined as one per terminal in TCT.

CWA – Common Work Area – System work area defines by system programmer in SIT. (System Initialization Table)

We can use TWA, TCTUA & CWA for sharing the information across the transactions.

External address ability using BLL CELLS of OS/VS COBOL

Base Locator for Linkage is used to access the memory outside you working storage. If it is used in input commands like READ, RECEIVE the performance will be good as we would be directly accessing the input buffer than working storage item. If you want to access system area like CWA, TCTUA or CSA that exist outside your program, then you should use BLL.

CODE Linkage section as below:

```
O1 PARM-LIST.
      05 FILLER         PIC S9(08) COMP.
      05 TCTUA-PTR      PIC S9(08) COMP.
      05 TWA-PTR        PIC S9(08) COMP.

01 TCTUA-AREA.
      05 TCTUA-INFORMATION    PIC X(m).

01 TWA-AREA.
      05 TWA-INFORMATION      PIC X(n).
```

The mapping between the pointers & data area is done in linkage section as follows.

First filler points the current 01 level, which is PARM-LIST itself.
TCTUA-PTR points to next 01 level, which is TCTUA-PTR.
TWS-PTR points to next 01 level, which is TWA-PTR.

This will store the pointer of TWA in TWA-PTR & TCTUA in TUCTUA in TCTUA-PTR. As the mapping between BLL cells & areas is already done in linkage section, TCTUA-AREA can access m bytes of TCTUA, which exist outside your working storage & similarly TWA-AREA to access n bytes of TWA, which exist outside your working storage area.

To ensure the addressability, SERVICE RELOAD statement should follow whenever the content of BLL cell is changed in anyway. So there should be a SERVICE RELOAD statement after the ADDRESS statement as follows.

Enhancement by VS COBOL2.

ADDRESS of operator of VS COBOL2 easies the access of external items. The above requirement can be met in VS COBOL2 as follows. There is no need for PARM-LIST or SERVICE RELOAD.

```
LINKAGE SECTION.
01 TCTUA-AREA.
      05 TCTUA-INFORMATION    PIC X(m).

01 TWA-AREA.
      05 TWA-INFORMATION      PIC X(n).

PROCEDURE DIVISION.

EXEC CICS ADDRESS TCTUA (ADDRESS OF TCTUA-AREA)
                  TWA (ADDRESS OF TWA-AREA)
END-EXEC.
```

ADDRESS OF maps the TCTUA-AREA with TCTUA exists outside your working storage.

ASSIGN statement.

It is used to access the system value outside of application program.
Some of them are length of common areas, user-id.
CWALENG, TCTUALENG, TWALENG - assign to full word binary working storage item.
USERID - assign to X(08) field.

```
EXEC CICS ASSIGN
            USERID(WS-USERID);
END-EXEC.
```

The above command fetches the user id into WS-USERID.

EIB – Executive Interface Block.

When CICS translator translates your program, it adds DVHEIBLK copybook as the first in your linkage section. Whenever a task is initiated, the task related information could be accessed using the fields in the copybook. EIB is acronym for Executive Interface Block.

Some of the fields in EIB are:

```
EIBCALEN          : Length of DFHCOMMAREA.
EIBDATE & EIBTIME : Date & Time of Task initiation.
EIBAID            : The last attention identifier pressed by the user.
EIBFN             : Function code of the last command. (2 bytes)
EIBRCODE          : Response code of the last command. (6 bytes)
EIBTRMID          : Terminal ID.
EIBTRNID          : Transaction ID.
```

CICS PROGRAM LINKAGE SECTION

If you look into the translated source listing, the linkage section of CICS program might have the following.

1. DFHEIBLK (Introduced by Translator)
2. DFHCOMMAREA (Coded by you as the first entry in linkage section.)
3. PARM-LIST (BLL cells coded by you)
4. Multiple 01 level (The number of 01 entries is equal to number of BLL cells you have coded in PARM-LIST excluding the first BLL that points to PARM-LIST itself.)

GETMAIN & FREEMAIN.

GETMAIN command is used to obtain a certain amount of storage in the main memory.

```
EXEC CICS
      GETMAIN SET(ADDRESS OF LK-ITEM)
      LENGTH(data-value)|FLENGTH(data-value)
      INITIMG(data-value)
END-EXEC.
```

LK-ITEM is a linkage 01 item whose length is equal to length mentioned in the LENGTH parameter & that much memory will be allocated for you in the main memory. The initial value of this field will be set by INITIMG parameter.

```
EXEC CICS
      FREEMAIN DATA(LK-ITEM)
END-EXEC.
```

The above command will release the main memory occupied by you using GETMAIN.

These commands are highly useful in macro level coding where the quasi-reentrancy is the responsibility of the application programmer. Quasi-reentrancy is guaranteed in the CICS command level COBOL programs & so the need & usage of these commands are almost obsolete.

LINK XCTL & CALL

It is good practice to develop several functional modules rather than building one lengthy program that does all the functions. It makes the program tough to maintain, understand & debug. The size of the program will be large & it also occupies more resources. Functional splitting of sub modules & calling or transferring the control to them is achieved by LIN, XCTL & CALL.

CICS program are usually run at various levels. The first program, which gets the control from CICS, that is the program registered against the transaction in PCT, runs at first level. Linked & Called program are running one level lower than linking or calling program & so the RETURN or GO BACK in these programs will give the control back to Linking or Calling program. XCTL programs run at the same level * so the RETURN gives control to next upped level.

Syntax:

```
EXEC CICS
      LINK|XCTL PROGRAM(name)
      COMMAREA(ws-area)
      LENGTH(ws-length)
END-EXEC.
```

Ws-length is length of ws-area & ws-area has the information that needs to be passed to LIN/XCTL program. VS COBOL 2 allows reentrant program & so whatever is achieved using LINK or XCTL can be also achieved by COBOL CALL statement. The called program should be attached to the main program during link edit & so the size of the load module will be large in this case.

Prefer XCTL whenever possible as it involves fewer overheads. If that is not possible then based on sub-program size & possible modification, go for LINK or CALL. If the size o the program is less & used by only very few program go for CALL as it won’t increase the size much at the same time application will run fast.

If the program is expecting more changes, then calling program also need to be compiled for every change in the sub program. In case of LINK, the compilation of linked program is enough. Called CICS program need not be registered in PPT whereas LINK & XCTL program should be.

LOAD & RELEASE

LOAD is used to load the program or table that is compiled or assembled, link edited & registered in PPT. It is mostly used for loading the assembler table.

```
EXEC CICS
      LOAD PROGRAM(PGM1)
      SET(ADDRESS OF LK-ITEM)
      LENGTH(WS-LENGTH)
END-EXEC.
```

Program PGM1 is loaded & the address of the program or table is mapped to LK-ITEM & so the table can be accessed using the linkage are LK-ITEM. The size of the linage item is WS-LENGTH. If 'HOLD' keyword is coded in the LOAD command, then loaded program or table will be permanently resident until explicitly released. If its not mentioned, then termination of the task release the program or table.

```
EXEC CICS
      RELEASE PROGRAM(PGM1)
END-EXEC.
```

The above command is used to release the program or table.

AID & HANDLE AID

There are certain attention identifiers (AID) associates with every screen. Entry screens have ENTER & PF1(Help) keys. Query screens have PF7(Page up), PF8(Page down), PF1(Help) & so on. Based on the key pressed by the user, the processing should happen in the program. EIBAID of DFHEIBLK has the last user pressed key. It can be verified in the program after the RECEIVE command.

For verification of values in EIBAID, copy the standard AID list copybook provided by CICS in your program (COPY DFHAID). Now you can easily verify the key pressed as follows.

```
      EVALUATE EIBAID
            WHEN DFHENTER PERFORM PARA-1
            WHEN DFHPF1 PERFORM PARA-2
            WHEN OTHER PERFORM PARA-3
      END-EVALUATE.
```

The above checking is done in COBOL & this kind of check has to done after every RECEIVE command to properly route the flow. CICS provides its own routing processing based on the key pressed by HANDLE AID command. This will be effective through out the program & reduce the code redundancy. The routing will be automatically invoked on every RECEIVE.

```
      EXEC CICS
            HANDLE AID
            DFHENTER(PARA-1)
            DFHPF1(PARA-2)
            ANYKEY(PARA-3)
      END-EXEC.
```

HANDLE CONDITION & NO HANDLE

Every CICS command has its own exception list. One example is MAPFAIL exception of RECEIVE command. HANDLE CONDITION is used to transfer control to proper procedure on expected exceptions. HANDLE condition cannot track program interrupted ABEND like SOC4 or SOC7. It deals only with exception of CICS commands.

The conditions handled are effective from where it appears to the end of the program. One handle condition can be overridden by another condition. The main program conditions will not be effective in sub-programs.

```
      EXEC CICS
            HANDLE CONDITION
```

```
            MAPFIAL(PARA-1)
            PGMIDERR(PARA-2)
            LENGERR(PARA-3)
            ERROR(PARA-X)
      END-EXEC.
```

- ⇨ Any error condition other than MAPFAIL, PGMIDERR & LENGERR will transfer the control to PARA-X.

If you want to reset the diversion done by previous HANDLE condition, code the condition name but don't mention the paragraph name. Maximum 12 conditions can be coded in one HANDLE CONDITION statement.

If you want to deactivate all the condition for a particular CICS command, code NOHANDLE. There are possibilities for infinite loop when CICS command in the exception routine of the HANDLE CONDITION ends with same exception. In the above example, if the there is LENERR is PARA-3 for any of the CICS command coded there, if the there is LENGERR in PARA-3 for any of the CICS command coded there, then the control again come to PARA-3 & forms infinite loop. In such cases, NOHANDLE will be useful(in the ABEND routine CICS commands).

IGNORE CONDITION.

The syntax is same as HANDLE CONDITION but it causes no action to be taken if the specified condition occurs in the program. The control will be returned to the next instruction following the command, which encountered the exceptional condition. It will be effective from the place where it appears to the end of the program or until any HANDLE condition overrides it.

Maximum 12 conditions can be coded in one IGNORE CONDITION statement.

RESP.

Like the file status verification of batch COBOL program, SQLCODE verification of DB2 program, the success or failure of CICS command can be done using COBOL statements & this give more structured look for the program.

To achieve this, code RESP along with CICS command. CICS will place the result into the variable coded in RESP. It can be checked against DFHRESP(NORMAL) or DFHRESP(condition) & appropriately control the flow following the command.

```
      EXEC CICS
            RECEIVE MAP(A)
                    MAPSET(B)
                    RESP(WS-RCODE)
      END-EXEC

      EVALUATE WS-RECODE
            WHEN DFHRESP(NORMAL) PERFORM PROCESS-PARA
            WHEN DFHRESP(MAPFAIL) PERFORM PARA-1
            WHEN OTHER PERFORM PARA-X
      END-EVALUATE.
```

WS-RCODE should be defined as full word binary in working storage. If you code RESP, then HANDLE CONDITION will not be effective for those CICS commands.

PUSH & POP.

They are used to suspend & reactivate respectively, all the HANDLE CONDITION request currently if effect. In real life programming, you may want to deactivate all the active handle conditions & diversions for a subroutine(section) embedded in the program. This can be achieved using PUSH HANDLE. When you come out of the section, you can reactivate all the pushed commands & that can be done using POP HANDLE.

HANDLE ABEND

HANDLE CONDITION command intercepts only abnormal condition of the CICS command execution whereas HANDLE ABEND takes care of any abnormal termination within the program.

```
EXEC CICS
      HANDLE AID
      LABEL(ABEND-ROUTINE)RESTE|CANCEL
END-EXEC.
```

LABEL is to activate an exit. CANCEL is used to cancel the previously established HANDLE ABEND request. RESET is to reactivate the previously cancelled HANDLE ABEND request.

USER ABEND

In batch COBOL, the user ABENDS can be thrown by calling the assembler routine ILBOABN0 using AB-CODE whereas AB-CODE is working storage field of half word binary.

The following command is used to throw user ABENDS in CICS.

```
EXEC CICS
      ABEND ABCODE('9999')
END-EXEC.
```

The above is used to throw user ABEND '9999'.

FILE HANDLING

CICS support only VSAM & BDAM files. All the files, that you want to use in your CICS application, should be registered in FCT with their complete attributes. CICS commands refer the FCT entry name for doing operation on the file.

As all the files are already declared & defines in tables, coding File-Control Paragraph of ENVIRONMENT division or FILE SECTION of DATA DIVISION is meaningless & not required. Thus CICS frees the application program from ant data dependant coding.

The unit of IO during READ is one control interval. So even you read one record into your program, the complete control interval is read into main memory buffer. The size of control interval is preferred to be large for sequential processing & small for random processing.

The file should be OPEN to issue an I-O command. The status of the file can be queried using the master transaction CEMT. It can be OPENED, CLOSED, ENABLED or DISABLED. This explicit opening or closing can be done using CEMT. But this needs human intervention.

CICS Version 1.7 introduces automatic opening of the file if it is not in open status during the access. It is always better to close it the when you ho longer need it. DFOC(Dynamic File Open Close) can be done in program using the SET command or Lining to DFHEMTP.

DFOC by LINK COMMAND

The application program links the master terminal program(DFHEMTP) & passes CEMT parameter data required for the file open/close through COMMAREA. Receiving the data, DFHEMTP program opens/close the files specified, after which the control is returned to the application program.

```
EXEC CICS
      PROGRAM(DFHEMTP)
      COMMAREA(WS-COMMAREA)
      LENGTH(WS-LENGTH)
END-EXEC.
```

The file open command is written into WS-COMMAREA & its length is populated in WS-LENGTH.
File Open Command: SET DATASET(FCT-NAME) OPENED

DFOC by SET COMMAND

```
EXEC CICS SET
      DATASET(name)|FILE(name)
      OPEN|CLOSED
END-EXEC.
```

DFOC by BATCH JOB

DFOC can be achieved from a batch job, which issues the CEMT as the data for the JES Modify (F) command against the systems console terminal.

```
//F CICSJOB, 'CEMT SET DATASET(FCTNAME) OPENED'
```

RANDOM ACCESS OF FILES

READ:

```
EXEC CICS
      READ FILE(FCT-NAME) [UPDATE]              S1
      INTO(WS-ITEM)|SET(ADDRESS OF LK-ITEM)     S2
      LENGTH(WS-LENGTH)                         S3
      RIDFLD(WS-KEY)                            S4
      KEYLENGTH(WS-KEY-LENGTH) GENERIC          S5
      SYSID(SYSTEM-NAME)                        S6
      GTEQ|EQUAL                                S7
END-EXEC.
```

S1 - Reads the file & if update in intended then code UPDATE clause. This will get exclusive access over the complete control interval where the specific record exists, during the READ. Thus CICS ensures data integrity.

S2 - Read the file into working storage variable WS-ITEM. Alternatively, the address of the record in the input-output area can be mapped to linkage section variable by using SET command. Performance wise SET is better than READ INTO. VS COBOL2 supports ADDRSSS of keyword & makes the code easier. In the prior version COBOL, PARM list & SERVICE RELOAD should be used to achieve the same.

S3 – After the READ, the LENGTH of the record READ is updated into WS-LENGTH. WS-LENGTH is the working storage half word binary item.

S4 – Key of the record to be read is moves into WS-KEY, a working storage item & a part of record structure(WS-ITEM).

S5 – If partial key is used, then the length of the partial key should be moved to WS-KEY-LENGTH & the keyword GENERIC should be coded. This is optional for full-key read.

S6 – Remote system name where the request to be directed, is code here. (1-4 character name).

S7 – EQUAL is default. GTEQ can be coded when you know the full key, but you are not sure the record with that key exists in the file.

REWRITE

```
EXEC CICS
```

```
            REWRITE FILE(FCT-NAME)
            FROM(WS-ITEM)
            LENGTH(WS-LENGTH)
            SYSIN(SYSTEM-NAME)
      END-EXEC.
```

To release the exclusive access acquired during the READ with UPDATE, the records should be REWRITTEN using the above syntax. The parameters are self-explanatory.

After you read record, if you feel that you don't want to update it, then inform the same to CICS be issuing UNLOCK command so that CICS release the lock acquired by you on the record.

```
      EXEC CICS
            UNLOCK FILE(FCT-NAME)
      END-EXEC.
```

WRITE

The syntax of WRITE is same as REWRITE. There is a parameter RIDFLD with key-value is coded in the WRITE command. (Like S4 for READ command)

When you want to add a set of records whose keys are in ascending order, then qualify your first WRITE with another parameter MASSINSERT. This will get exclusive control over the file & provide high performance during the mass insert.

If you use MASSINSERT, then you should issue UNLOC to inform the completion of your additions. CICS releases the file on UNLOCK.

DELETE

1. The records read using READ with UPDATE can be deleted using

```
            EXEC CICS
                  DELETE DATASET(FCT-NAME)
            END-EXEC.
```

2. The record in the file can directly be deleted by providing complete key.

```
            EXEC CICS
                  DELETE DATASET(FCT-NAME)
                  RIDFLD(WS-KEY)
            END-EXEC.
```

3. Group of records can be deleted using partial key. After the deletion, the number of records deleted is updated in the variable coded in NUMERIC parameter. (WS-DEL-COUNT).

```
            EXEC CICS
                  DELETE DATASET(FCT-NAME)
                  RIDFLD(WS-KEY)
                  KEYLENGTH(WS-KEY-LENGTH) GENERIC
                  NUMREC(WS-DEL-COUNT)
            END-EXEC.
```

SEQUENTIAL READ

Sequential access of VSAM file under CICS is called Browsing. It has FIVE commands associated with it.

STARTBR

It establishes a position to start browsing.

```
            EXEC CICS
                  STARTBR FILE(FCT-NAME)
                  RIDFLD(WS-KEY)
                  KEYLENGTH(WS-KEY-LENGTH) GENERIC
                  REQID(Integer-value)
```

```
            SYSID(SYSTEM-NAME)
            GTEQ|EQUAL
      END-EXEC.
```

Meaning of the parameters is same as READ operation explanation. When you want to do multiple browsing concurrently over the same file, then use REQID. The first STARTBR cab de coded with REQID(1) & the second STARTBR can be coded with REQID(2).

One browse operation occupies one string of VSAM. If all VSAM strings are exhausted for one VSAM file, then the other transaction will have to wait for one of the strings to become free. So it is not recommended to use multiple browsing. Instead, once the browsing is completed. The syntax of RESETBR is same as STARTBR.

READNEXT & READPREV

```
      EXEC CICS
            READNEXT|READPREV FILE(FCT-NAME)
            RIDFLD(WS-KEY)
            KEYLENGTH(WS-KEY-LENGTH)
            REQID(Integer-value)
            SYSID(SYSTEM-NAME)
      END-EXEC.
```

READNEXT is used to read the records in the forward direction from the position established by STARTBR. READPREV is used to read the records in the reverse direction READPREV followed by READNEXT retrieves the same records once again.

During the browse, if you want to skip set of records, and then move the key of the next records you intended to read to RIDFLD & then issue READNEXT. This is called skip sequential read. Thus RIDFLD can be used as both input as well as output field.

As VSAM files are variables length in most of the cases, WS-LENGTH should be populated with maximum record length before issuing READ command. KEYLENGTH(0) will position the cursor at the beginning of the file.

If you specify READPREV immediately after STARTBR, then you should code key of a record that exists on the dataset. Otherwise NOTFND condition occurs on READPREV.

ENDBR.

If is used to terminate the current browse function.

```
      EXEC CICS
            ENDBR FILE(FCT-NAME)
            REQID(Integer-value)
            SYSID(SYSTEM-NAME)
      END-EXEC.
```

UPDATE during BROWSE

Update & browse are mutually exclusive functions. So if you want to update a record during the browse operation, first issue ENDBR. Then give random read using the key in hand with UPDATE option. REWRITE the record. Then issue STARTBR with the key in hand & continue the browsing.

ESDS-BROWSING

Define a full-word binary item in working storage the RBA of ESDS file. Move LOW-VALUES or HIGH-VALUES to the field for the forward or reverse read respectively. Issue STARTBR with RBA & EQUAL options. RIDFLD should point to defines RBA field. Now issues READNEXT or READPREV for forward or REVERSE read respectively.

GTEQ is invalid for ESDS.

```
      MOVE LOW-VALUES TO ESDS-RBA
      EXEC CICS
            STARTBR FILE(FCT-NAME)
            RIDFLD(ESDS-RBA)
            RBA EQUAL
      END-EXEC.
      MOVE 226 TO WS-LENGTH
      EXEC CICS
            READNEXT FILE(FCT-NAME)
            INTO(WS-ITEM)
            RIDFLD(ESDS-RBA)
            LENGTH(WS-LENGTH)
            RBA
      END-EXEC.
```

ESDS-WRITE

The syntax is same as KSDS WRITE command. Qualify the WRITE with RBA option. The record will be appended of the file & the RBA of the record is place into RBA field mentioned in the WRITE command(ESDS-RBA),

```
      MOVE 226 TO WS-LENGTH
      EXEC CICS
            WRITE FILE(FCT-NAME)
            RIDFLD(ESDS-RBA)
            LENGTH(WS-LENGTH)
            RBA
      END-EXEC.
```

ESDS-RANDOM ACCESS

If you now the RBA value of the record you want to access, then you can randomly access the ESDS file. The syntax is same as READ of KSDS file but qualify it with RBA option. RIDFLD should point to full word binary item pres-filled with RBA of the record to be accessed.

RRDS ACCESS

RRDS file can be accessed using RRN in place of RBA & populating the RIDFLD with RRN number. The field should be full word binary.

Alternate Index Access

Register the path in FCT & use the path name as file name to read the file randomly using alternate index. Please refer VSAM section of the book for understanding what exactly PATH is.

As the alternate key need not be unique, DUPKEY condition is vey common during the alternate index usage & so it had to be properly handled.

ASKTIME

EIBDATE & EIBTIME have the values at task initiation time. Upon the completion of ASKTIME, these two fields are populated with current date & time

```
      EXEC CICS
            ASKTIME
      END-EXEC.
```

FORMATTIME

FORMATTIME is used to receive the information of date & time in various formats.

```
      EXEC CICS
            FORMATTIME
            FORMAT TYPE(data-area)
      END-EXEC.
```

```
Format-Type : YYDDD, YYMMDD, YYDDMM,  MMDDYY, DAYOFWEEK, DAYOFMONTH, MONTHOFYEAR, YEAR,
              TIME, TIMESEP & DATESEP.
```

WS-DATE contains the current date in MM/DD/YY (Default DATESEP is '/')
WS-TIME contains the current time in HH:MM:SS (Default TIMESEP is ':')

DELAY & SUSPEND

It is used to delay the processing of a task for a specified time interval or until the specified time. If your task is doing some heavy CPU bounded work, it is a good to place the DELAY command in order to allow the other tasks to proceed.

```
EXEC CICS
      DELAY INTERVAL(HHMMSS)|TIME(HHMMSS)
END-EXEC.
```

SUSPEND is used to suspend a task. During the execution of the command, the task will be suspended & the 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.

```
EXEC CICS
      FORMATTIME
      FORMAT TYPE(data-area)
END-EXEC.
```

POST & WAIT EVENT

POST is used to request notification when the specific time had expired & the WAIT EVENT command is used to wait for an event to occur. Usually these two commands are used in pair, as alternative to the DELAY command.

```
EXEC CICS
      POST INTERVAL(HHMMSS)|TIME(HHMMSS)
      SET (ADDRESS OF POST-ECB)
END-EXEC.

EXEC CICS
      WAIT EVENT ECAADDR(ADDRESS OF POST-ECB)
END-EXEC.
```

CANCEL

It is used to cancel the interval control commands such as DELAY, POST & START which have been issued. The interval commands to be cancelled are identified using REQID.

```
EXEC CICS
      START REQID('STARTS')
END-EXEC.

EXEC CICS
      CANCEL REQID('STARTS')
END-EXEC.                          Cancels all the start issued.
```

START & RETRIEVE

START command is used to start a transaction at the specified terminal & at the specified time or interval. Optionally, data can be passed to the to-be-initiated transaction.

```
      EXEC CICS
            START TRANSAID('EMPC') TERMID('TST1')          S1
            TIME(123000)|INTERVAL(003000)                  S2
            FROM(WS-ITEM) LENGTH(WS-LENGTH)                S3
            RTRANSID(EMPD') RTERMID('TSTO')                S4
            QUEUE('EMPDET')                                S5
      END-EXEC.
```

S1 - EMPC is to be started at the terminal TST1.
S2 - TIME & INTERVAL are mutually exclusive. If TIME(HHMMSS) is coded, then the than start at the terminal at the specific time. In the above example it is 12:30. If INTERVAL(HHMMSS) is coded then the task will start after the expiry of the interval. In the example, the task starts after 30 minutes.
S3 - This program passes WS-ITEM of size WS-LENGTH to the task EMPC.
S4 - Transaction ID EMPD & Terminal Id TSTO are passed to EMPC. EMPC can receive this & may use for next triggering.
S5 - TSQ queue name can be passed from this transaction. EMPDET is passed in our case.

The data passed by START commands is received by the new transaction using RETRIEVE command upon the expiry of START command.

```
      EXEC CICS
            RETREIVE INTO(WS-TIME)|SET(ADDRESS OF LK-ITEM)
            LENGTH(WS-LENGTH)
            RTRANSID(WS-TRAN) RTERMID(WS-TERM)
            QUEUE(WS-QUEUE)
      END-EXEC.
```

END & DEQ.

ENQ gets the exclusive control over the resource & DEQ releases the exclusive control acquired over the resource. As CICS takes care of exclusive access over the resource needed, it is advisable not to use it in application programs.

But sometimes it may be needed in few cases like updating TSQ records or getting access of the printer.

```
      EXEC CICS
            ENQ|DEQ RESOURCE(QUEUE-NAME)
      END-EXEC.
```

SYNCPOINT & TWO-PHASE-COMMIT

During the program processing, if you want to roll back the changes made by you, then you can code SYNPOINT with ROLLBACK. This will backed out tell resource modification done since the last sync point.

```
      EXEC CICS
            SYNCPOINT
            ROLLBACK
      END-EXEC.
```

SYNCPOINT in CICS-DB2 environment is called as two-phase commit. The changes made in the resource that are directly under the control of CICS are committed in the first phase & the changes made to DB2 environment are committed in the second phase.

Queues

There are two types of queues available in CICS.

1. Temporary Storage Queue(TSQ).
2. Transient Data Queue(TDQ).

Their properties are listed below:

TSQ	1. No need to predefine anywhere. By default, TS queues are created in main memory. So the content cannot be recovered after system crash. If you want to recover, then you should define then in TST. 2. Records can be randomly accessed using ITEM option of READQ. READ is not destructive. 3. Deletion of queue deletes all the records in it. Deletion of recoverable TSQ should follow sync point before next WRITEQ. 4. Primarily used as scratch pad memory facility for ant purpose. Example: Page-up & Page-down logic, Passing huge data between the phases of transaction, Review & correction of multiple order entry screens.
TDQ	TDQ should be predefined in DCT. 1. READ is destructive & only sequential. Once the record is READ, it will be logically deleted & cannot be read again, 2. Automatically Task Initiation: When number of records in the queue exceeds the TRIGLEV defined in the DFHDCT entry of the queue, the transaction coded in the TRANSID of DFHDCT is automatically triggered. This ATI is possible only in TDQ. 3. There are two types of TDQ – Intra & Extra partition. The DCT entry identifies the type of queue from TYPE parameter. 4. Intra Partition Queue are used within a CICS region & DELETEQ deletes all the records physically in the queue, 5. Extra Partition Queues are used across the regions & systems. If you want to pass some data from CICS to batch or receive data from Batch to CICS, you should go for extra partition queue. 6. In the same program TDQ cannot be opened in both input & output mode. TSQ are preferred over TDA for data passing. TDA is used for batch interface or on ATI requirement.

MRO & ISC

Within one processor, there could be more than one CICS region, each of which runs independently under the same operating system. This is called Multi region Operation(MRO). It is achieved by four inter communication facilities.

1. Function Processing.
2. Asynchronous Processing.
3. Transaction Processing.
4. Distributed Transaction Processing.

One CICS in the one processor can communicate with other CICS in other processor or other non-CICS system regardless of where the other processor physically located. This is called Inter System Communication (ISC). It requires Sophisticated Communication Network based on the System Network Architecture (SNA).

MASTER TRANSACTION

CEMT

This is Master Terminal Transaction. It is menu drive & east-to-use transaction but due to its nature of manipulating the CICS environment, most of the transactions are restricted to application programmer or end user.

```
CEMT INQUIRE|SET|PERFORM
CEMT INQ TRAN|PRO|FILE- Display information from PCT|PPT|FCT
CEMT INW TASK -         Display the active running in the region.
```

CEMT SET Command can be used to reset the values of PCT|PPT|FCT.

PERFORM has to be handled carefully. CEMT PERFORM SHUTDOWN will shut down the entire CICS region.

Frequently used commands:
CEMT SET PR(PGM-NAME) NEW – To create a new copy of an application program.
CEMT SET DA(File-name) CLO/OPEN – To close/open a file from CICS.

CECI:

This is Command Interpreter Transaction. CICS commands can ne pre-tested using this command before placing them into the program.

CERR:

This is browse transaction. TSQ can be browsed using this transaction.

CEDF:

This is Diagnostic Facility Transaction. If you want to debug your program step by step, then before typing transaction ID type CEDF so that debug mode will be set. It intercepts a transaction & termination of a program, before & after execution of any EXEC CICS or EXEC SQL commands.

CAMC:

This transaction is used to get ABEND codes & messages.

Different ways of initialing a transaction in CICS

1. By entering transaction Identifier.
2. By START command.
3. By RETURN TRANSID.
4. By registering the program in PLT, they will be automatically initiated during CICS startup.
5. Automatic Task Initiation of TDQ.
6. PF or PA keys could be defines in PCT to initiate a CICS transaction.

CICS Dump Analysis

Dump usually consists of Dump Header Area, TCA(User Area), TCA(System Area), Trace Table , Transaction Storage & Program Storage.

At the top the ABEND dump, there is a dump header area, which provides ABEND code, TASK(the transaction ID), Program Status Word(PSW) & the contents of register at the time is ABEND.

PSW.

It is a double word field containing the status information at the time of ABEND. 32nd bit says the addressing mode in use. If the addressing mode is 31 bits, then the bit will be ‘1’ & the bits 33 to 63 contains the next sequential instruction(NSI), which would have been executed if the ABEND had not occurred.

First make sure that you have program compilation & link edited SYSPRINTS of the program ABENDED & dump prints of the ABENEDED transaction is with you. Program should be compiled with LIST option.

Finding the statement Caused the ABEND

1. Identify the address of next sequential instruction from the PSW.
2. From the program storage, identify the starting address of program storage.

3. Subtract the starting address of program storage from the NSI address identified in the first step; you will get the displacement of the NSI from the beginning of program storage.
4. Find the displacement of application program from the beginning of the load module in the link edit list & subtract this value from the calculated NSI displacement to get the real NSI displacement.
5. Find the statement in the compilation listing that corresponds to the OFFSET calculated in the previous step. The statement one above is the statement that caused the ABEND.
6. Analyze the statement for the cause of ABEND. This might have involved one or more fields. Most of the times, you can easily come to conclusion the cause of the ABEND once you identified the statement. If you could not, then you may check the value of the field at the time of ABEND as follows.

Finding the value of the field in error

1. Find the Base Locator(BL), Displacement & Length of the field from the MAP of compilation listing.
2. Find the register number corresponding to base locator from the bottom of compilation listing.
3. Find the address of this register in the dump.
4. Add the displacement to the register of the address to get the address of the field.
5. Locate the address in the dump to get the value of the field.

CICS TABLE ENTRIES

FCT

All the VSAM files, PATH between base cluster & alternate index & any BDAM datasets are to be registered in FCT to access them in the application program.

```
DFHFCT      TYPE = DATASET|FILE,
            ACCMETH = BDAM|VSAM,
            DATASET = name|FILE=name,
            SERVREQ = (ADD,BROWSE,DELETE,READ,UPDATE)
            [FILESET = (ENABLED|DISABLED,OPENED|CLOSED)],
            [BUFND = n+1],
            [BUFNI = n],
            [STRNO = n],
            [DSNAME = name],
            [DISP = OLD|SHR],
            [BASE = name],
```

ACCMETH defines data access method.

Name mentioned in the FILE parameter is the one with which we use in the CICS file commands. The name should correspond to the DDNAME of the file specified in the JCL of the CICS job itself. If DSNAME & DISP are coded, then JCL entry is not needed.

SERVREQ lists the authorized input/output operational against the file. If other than the specified services are requested in the file control command, then INVREQ condition will occur.

FILESET indicates the initial file status when CICS initially starts. BUFND & BUFNI indicate the number of data buffers & index buffers respectively. Default value is one for BUFNI & two for BUFND. STRNO is the number of VSAM strings by which concurrent access to the file is allowed.

If the file being registered is path, then BASE indicates the FCT name of the base cluster.

PPT

All the CICS application programs & BMS map sets must be registered in PPT. If the program is not registered here, then the program is unrecognizable to CICS.

```
DFHPPT      TYPE=ENTRY,
            PROGRAM=name|MAPSET=name
            [PGMLANG=(ASSEMBLER|COBOL|PLI)]
            Options..
```

ASSEMBLER is the default program language. All the map-set are considered as written in assembler.

PCT

The control information of all CICS transaction must be registered in program control table(PCT)

```
DFHPCT      TYPE=ENTRY,
            TRANSID=name,
            TASKREQ=xxxx,     (PF1-PF24,PA1-PA3)
            PROGRAM=name,
            [DTIMOUT=mmss]
            [RTIMOUT=mmss]
            [RESTART=NO|YES]
            [TRANSEC=1|DECIMAL], (1-64)
            [DUMO=YES|NO],
            …Other options..
```

TRANSID – Transaction Identifier Name(1-4 character)
PROGRAM – Name of the program associated with the transaction.
TASKREQ – Key associated with the transaction initiation.
DTIMOUT – Defines the timeout (waiting time) in case of deadlock.
RTIMOUT – Defines the time limit within the user has to respond. If he didn’t then the task will be cancelled.
RESTART – Automatic transaction is applied after the completion of the transaction recovery from the abnormal termination of the transaction.

TCT

All the terminals, which are to be under CICS control, should be registered in TCT. Terminal Control Program uses this table for identifying the terminal & performs all input/output operations against there terminals.

```
DFHTCT      TYPE=ENTRY,
            ACCMETH=VTAM,
            TRMIDNT=name,
            TRMTYPE=type,
            FEATURE=(UCTRAN...)
            Options...
```

TRMIDNT is (1-4) characters terminal ID.
TRMTYPE is type of terminal. Ex. IBM3270.
FEATURE indicates the terminal services CICS offers. UCTRAN is used to translate all the lowercase characters to upped case characters.

DCT

TDQ control information is registered here. Destination control program uses this table for identifying all TDQ’s & performs input/output operations against them.

Intra Partition queue can be defined as follows

```
DFHDCT      TYPE=INTRA,
```

```
            DESTID=name,
            [TRANSID=name],
            [TRIGLEV=name],
            [REUSE=YES|NO]
```

TYPE can be INTRA (Intra partition TDQ)

ATI:

TRANSID & TRIGLEV are used for automatic task initiation. If the number of records in TDO exceeded the number of records mentioned against TRIGLEV, then transaction ID mentioned against TRANSID will be invoked automatically.

Once a record of the intra partition TDQ is read by a transaction, the record is logically removed from, the queue. It REUSE=YES is coded, then the space occupied by it can be used by other records in future but the logically deleted records are not recoverable.

Extra Partition queue can be defined as follows

```
DFHDCT      TYPE=EXTRA,
            DESTID=name,
            DSCNAME=name-2,
            [OPEN=INITIAL|DEFERED],

DFHDCT      TYPE=SDSCI,
            DSCNAME=name-2,
            [TYPFILE=INPUT|OUTPUT|BACK],
```

OPEN=INITIAL means the file will be open at the CICS start-up time & DEFERED means the file will be closed until it is specifically opened by CEMT.

DSCNAME defines data control with TYPE=SDSCI & the same DSCNAME, which in effect indicates DDNAME of extra partition dataset in JCL or CICS JOB itself.

TYPFILE indicate the file to be INPUT, OUTPUT or READ BACKWARD(RDBACK)

ABEND CODES

When a CICS task is terminated abnormally, an ABEND code will be shown with the abnormal termination message. All such ABEND codes can be grouped into two categories.

1. ABENDS that are related with exception conditions.
2. ABENDS that are not related with exceptional conditions.

Selected ABENDS that are related with CICS exceptions conditions:

ABEND	Condition	ABEND	Condition
AEIA	ERROR	AEID	EOF
AEIJ	NOSTAT	AEIK	TERMIDERR
AEIL	DSIDERR	AEIM	NOTFND
AEIP	INVREQ	AEIQ	DUPKEY
AEIR	NOSPACE	AEIS	NOTOPEN
AEI9	MAPFAIL	AEXL	DISABLED
AEYH	QIDERR	AEYL	FUNERR
AEYQ	SYSIBERR	AEIT	ENDFILE
AEIU	ILLOGIC	AEIV	LENERR
AEIW	QZERO	AEIZ	ITEMERR
AEI0	PGMIRERR	AEI1	TRANSIDERR
AEI2	ENDDATA	AEYY	NOTALLOC

ABENDS not related with CICS exception conditions.

ABEND	Condition
ASRA	ABEND due to program interruption.
ASRB	ABEND due to OS ABEND intercepted by CICS.

AKCT	Read time-out encountered.
AKCS	Deadlock time out encountered.
AICA	Transaction abended as a runaway task.
AEY9	Unsupported command issued
AEY7	RACF failure
ABMP	PA/PF key not defines for page retrieval
ABMG	Requested BMS service not included as SYSGEN.

INTERVIEW QUESTIONS

1. TSQ & TDQ – Difference & when will you go for what?
2. Automatic Task Initiation – What it means?
3. What are the ways of initiating a transaction in CICS?
4. What are the ways, data can be passed between transactions?
5. Task initiation process in CICS.
6. LIN,XCTL & CALL difference & when you prefer what?
7. Importance of RCT table?
8. What is ASRA? How do you solve?
9. Definition of Pseudo Conversation?
10. Difference between Symbolic & Physical map?
11. What is MDT? Ways of setting & resetting MDT?
12. Ways of positioning the cursor in screen.
13. Few error codes, you faced in CICS development?
14. How the exceptions are handled in CICS?
15. How to submit a JCL from a CICS program?
16. How do you open file in CICS?
17. What are the ways you can release the exclusive access acquired over the control interval during READ-UPDATE?
18. How many conditions can be coded in one HANDLE/IGNORE condition?
19. What is two-phase commit?
20. How would you read the last record of ESDS VSAM file?
21. What is the limitation of COMMAREA & why this limitation?
22. How do you get current date & time, user-id in CICS environment?
23. How to read 10^{th} record of TSQ directly & how to delete the 10^{th} record of the TSQ?
24. What is MAPFIAL?
25. What are PUSH & POP & when they are useful?
26. What is the compiler options with which your COBOL program has to be compiled to use in CICS region?
27. What is reentrant & quasi-reentrant?
28. What happens when the program linked/XCTLed is not found in PPT?
29. All the COBOL programs with CICS commands need to be registered in PPT though they are called by CALL statement.(Y/N)
30. Which is the EIB field that determines whether the COMMAREA was send or not?
31. PPT & PCT contents
32. How to read the file using partial key?
33. When REWRITE issued without READ UPDATE, what will happen?
34. What is transaction deadlock?
35. How will you execute the coded CICS-DB2-COBOL program? Explain from compilation stage.
36. Why DB2 pre-compilation is done before CICS translation? What will happen if I reverse the order?
37. Which is the parameter of the DFHFCT macro that is used to translate all lower-case character to upper-case for 3270 terminals?
38. Which is the AID that will not be identified in the ANYKEY option of the HANDLE AID command?
39. What facility connects CICS & DB2?