**i-One Technologies**

***IBM MAINFRAME Technical Book***

**IBM MAINFRAME**

**Introduction of Mainframes:**

* IBM mainframes were started in 1950's.
* 70 % of the business data all over the world is maintaining by the mainframe systems only.

**Features of the Mainframe System:-**

**1.** Huge amount of data can be stored

2. High processing speed of 80 MIPS(millions of instructions per second) on any huge data.

3. Data security with no virus threats.

**Technical skills we need to learn the Mainframe:-**

**COBOL:** Common business oriented language. This is the programming language for business applications.

**JCL :**Job controlling language. Compilation and execution will be done by using this language.

**DB2:**Data Base 2 (Relational data base management system)

**VSAM:**Virtual storage access method ( This is used in the file concepts).

**CICS :** Customer information control system ( it is the online system in the Mainframe)

Usually computer is having some amount of memory called Hard disk. Initially the hard disk is not at all having any data. We will install operating system and drivers which are required.

1. Generally the time to take to install the operating system like XP, Windows, Linux. Those will take maximum of 1 to 5 hrs respectively. Where as for the mainframe systems to install the Z/OS, OS/360, OS/390, It will take 30-40 days. Then it will works 40 years or life long.
2. The windows and linux operating systems may attack by virus. Where as in mainframes O.S will not attack or crashed because of virus. That means

**''**Mainframe systems are highly secure''

1. Initially the 'Hard disk' contains nothing other than installation of O.S, Drivers.

Where as in system memory may contains lot of programs with the operating system.

1. Here we are using JCL (job controlling language) for the means of communication between the programs written in the cobol and the operating system. By using the JCL's the programs will will executes by using Buffer (RAM). Which is created by the operating system.

* If any updates are there for the database the operations will be done for the DB2 or VSAM.
* If any updates are there for the programmes then the updations will be done for the COBOL or CICS.

''This is the overall procedure we are following in the Mainframe Systems''

**How do you connect from a P.C to the Mainframe System?**

* We need to install emulation software on the P.C . Which creates mainframe environment
* By specifying the IP address of the mainframe system and through internet we can connect to the mainframe system.
* **Emulators:**

RUMBA, HUMMING BIRD, EXTRA, QWS,, FIRE

VISTA ---> we are using this for the lab.

**RACF**(Resource access control facility):--Security feature

* It is a future in mainframe which controls the resource access.
* The resources can be a user id (or) DB2 Tables (or) a file (or) application related programs and other data from users.
* Through RACF the system administrator provides a control access on data to the users.
* The user id use to logon to mainframes is called RACF-id/ Mainframe-id/ Tso-id/ User-id.

**Note:-** ''ID and Password maximum of 8 characters long.''

**TSO:--**(Time sharing option)

Mainframe system allows multiple users to access it on time sharing basis.

**Note:--**If password is typed incorrectly 3 times, it will be revoked.

JCL (Job Control Language)

* JCL acts as an interface between the programming language and the operating system. i.e it is the corner of the program to the operating system.
* Every task (program) executed on the mainframe system is treated as a JOB and submitted by JCL to operating system.

**Uses of JCL** :--

1. It is used to compile and execute programs.
2. It is used to create data sets, GDG, VSAM clusters.
3. It is used to copy data from one dataset to another dataset.
4. It is used to compare two files or two PDS members.
5. It is used to sort and merge the data in a file.

**What are the different JCL Statements & functions:--**

There are 8 JCL Statements.

1).JOB

2).EXEC

3).DD

4).PROC(Used to start the instreamproc)

5).PEND(Used to end the instreamproc)

6).JCLLIB(Used to call Catalogued proc).

7).SET(Used in symbolic overriding)

8).NULL(Used to make the job end).

Note: JOB, EXEC, DD are the mandatory JCL statements to execute any job in JCL.

**Structure of JCL***: --*

All JCL statements can consists of up to five types of fields

**//** the two forward slashes are required at the beginning of each JCL statement in Columns 1 and 2.

***Name field -*** This is an optional field. If coded, should start at Column 3. It maximum length is 8.

***Operation field -*** Which indicates the operation that is to be performed

***Operand -*** This field must appear after Operation field. Must start at 16 column

***Comments -*** Comments begin one space after the last operand. Our simple example has no comments.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 2 | 3…………………10 | 11 | 12…………..15 | 16 17 ……. | 71 | 72………………………….80 |
| // | Name Parameter |  | operation | operands |  | comments |

The important input operations are JOB, EXEC, DD

* **JOB** Statement: -- It uniquely identifies the JCL in the mainframe system and also provides information like account information, the output device to which messages are sent, the type of messages, time allotted for a job’s execution.
* **EXEC** Statement: -- It is used to identify the program which is to be executed. we can specify maximum 255 EXEC statements in a JOB.
* **DD** Statement: -- It is data definition statement used to describe the dataset characteristics. We can specify a maximum 3273 DD statements within a single JOB.

**POSITIONAL PARAMETERS:**--

These are the mandatory parameters and must be specified in a particular order only. i.e those must be specified in a particular position. So, these are called positional parameters.

Positional parameters in JOB statement:--

1. Accounting information: -- It specifies the account number or account code to which billing has to be done for executing the programs on the mainframe system.

//AGY0157A  JOB  A123   
Here A123 is the account number to  which the CPU time utilized on the Mainframe will be billed.   
//AGY0157A JOB (,QUASAR)   
  Here QUASAR is the additional accounting information. The leading comma indicates the absence of the positional parameter Account no.   
//AGY0157A JOB  (A123,'QUASAR S') 

1. Program name: -- It specifies the coder name who has coded the JCL.

Ex:-- coding the account number and program name.

//JOB1 JOB ‘A1123’,’LOAD JOB’CLASS=A

Like the above we can write the position parameters.

**KEYWORD PARAMETERS :**--

Keyword parameters are those, which can appear in any order. They are identified by writing the keyword. The following keyword parameters are important for the JOB statement.   
 - CLASS, PRTY, MSGCLASS, MSGLEVEL, TYPRUN, TIME, NOTIFY

Keyword parameters in JOB statement :--

* **CLASS**:-- A CLASS parameter categorizes the JOB based on the factors like CPU consumption time and CPU resource utilization etc…

CLASS parameter has a range of values A-Z ‘ 0-9.

|  |  |
| --- | --- |
| **CLASS** |  |
| A | 1 MIN |
| B | 5 MINS |
| C | 30 MINS |
| D… | 1 HOUR |
| Z | 3 HOURS |
| 0 | Max CPU Resources |
| 1 | Min CPU Resources |

Note:--

* JOBS submitted with in the same class parameter will be queued and jobs executed one after the other.
* JOBS submitted with the different class parameter will be in a separate queue and are executed simultaneously with the other class jobs.

**PRTY:**-- It specifies the priority to a job execution. ‘PRTY’ has a range of values ‘0-15’

* Highest priority value ,the first preference it given for execution.

(if priority high execution is also high)

(if priority low then execution is also low)

Note:--

1. When two jobs are submitted, the job which is submitted first will be executed first.
2. If the jobs are submitted at the same time then based on the CLASS parameter the jobs will be executed.
3. If the jobs are submitted with same CLASS parameter and at the same time then the job is executed based on the ‘PRTY’ parameter.
4. If jobs are submitted with same CLASS parameter with same priority and at the same time then they are kept under wait and after the wait time the operator cancels the job.

**MSGCLASS: --**

It specifies the output device to which messages are sent. It has the range of values A-Z, 0-9.

MSGCLASS = A 🡪 printer

= X 🡪 spool

= 0 🡪 tool1

= 1 🡪 tool2

**MSGLEVEL: --**

It specifies the ‘type’ of messages to send and ‘when’ those messages have to be send.

MESLEVEL = (Statements, Messages)

Statements: -- Type of messages

0 🡪 only job related messages

1 🡪 all JCL related messages includes PROCS and system messages

2 🡪 all JCL related messages excluding PROCS and system messages

Messages: -- when messages have to be sent.

1. 🡪 upon successful execution of JOB
2. 🡪 upon successful or unsuccessful execution of JOB

Default: -- MSGLEVEL = (1,1)

**NOTIFY: --**

It specifies the user id to which the notification about the successful or unsuccessful execution of the JOB is sent.

Note:--Because of the “NOTIFY=&SYSUID” we are getting the ‘MAXCC=?’ value for that particular ‘SYSUID’.

* NOTIFY = &SYSUID will substitute the user-id with which the user has been logged in and so the notifications has send to same user-id.
* If we specify a different user-id (NOTIFY=USER ID) then notifications will be sent to this user-id, but the spool remains with the user-id.

**REGION: --**

It specifies the amount of work space required for a job or step execution.

* REGION must be specified in ODB quantities in Kilo or Mega bytes
* The mega region space can be allocating by specifying REGION= 0K or 0M.

Note: -- If the specified Region space is enough then job will abend with S106.

**TIME: --**

It specifies the maximum time allocated for a job or step execution.

Syntax:-- TIME = (Minutes ,Seconds)

=(10,10)

Note:-- Maximum time can be allocated by using

TIME = 1440 Hrs.

= 248 days

= NOLIMIT (or) MAX

**TYPERUN: --**

It tells the system what needs to be done to the JOB.

1. “TYPERUN=SCAN”

It checks JCL syntax errors or compile JCL.

1. “TYPERUN=HOLD”

It checks JCL syntax errors but keeps the JOB on HOLD without executing it.

|  |  |
| --- | --- |
| **JOB NAME** | **STATUS** |
| JOB1 | ON-HOLD |

Note: To release the JOB on hold specify the line command ‘a’ beside the job name.

1. “TYPERUN=COPY”

It copies the JOB to particular location.

**RESTART: --**

It is used to start the execution of the JOB from a particular step

// JOB1 JOB…………RESTART=STEP3.

// STEP1 EXEC PGM=PGM1

------------

-------------

// STEP2 EXEC PGM=PGM2

// STEP3 EXEC PGM=PGM3

Note:-- A JOB can have maximum of 255 EXEC statements.

**EXECStatement**: -- It is used to identify the program which is to be executed. In EXEC

statement we can specify maximum of 3273 DD statements in job.

**Positional Parameters in EXEC Statements: --**

**PGM: --** It specifies the program name to be executed.

**Keyword Parameters in EXEC Statements: --**

REGION, COND, TIMEare the common parameters in JOB and EXEC Statements.

Note:--

1. REGION , COND and TIME if specified in JOB statement then they are applicable to all the steps.
2. REGION , COND and TIME if specified in EXEC statement then they are applicable to only that particular step.
3. If REGION , COND and TIME are specified in both JOB and EXEC statements then that parameter value specified in the JOB override that of steps
4. “COND” specified in JOB and EXEC statements, then parameter value specified in step will override that of ‘JOB’.

// STEP1 EXEC PGM=PROG1,PARM=’10-12-2012’,

REGION=5K,TIME=10, COND=(08,LE)

PARM: -- It is used to pass the data from JCL to COBOL. We can pass the data a maximum of ‘100’ characters.

**Types of Libraries in Mainframe:**

There are 3 types of libraries.

1).System Libraries(SYS1.LINKLIB):

2).Private or User libraries:

3).Temporary libraries:

1).System Libraries: To execute the programs which are designed by IBM we doesn’t require JOBLIB & STEPLIB.

2). Private or User libraries: To execute the programs which are user defined we should use JOBLIB & STEPLIB.

3). Temporary libraries: To execute the programs which are designed for a particular project(for a particular period of time) we should use JOBLIB & STEPLIB.

**JOBLIB: --**

It specifies the load library in which all the steps, program load modules are searched.

**STEPLIB: --**

It specifies the load library in which a particular steps load module is searched.

// STEP2 EXEC PGM=PROG2

// STEPLIB DD DSN=FSSO45.IONETECH.LOADLIB,DISP=SHR

Then it checks in JOBLIB,if not found.

* If both JOBLIB and STEPLIB are specified then system first checks in STEPLIB and if not found then it checks in production libraries (or) system abend code S806.

S806 🡪 load module not found.

**COND: --**

It is used to control the execution of the steps.

// JOB1 JOB……….

// STEP1 EXEC PGM=….

// STEP2 EXEC…..

// STEP3

// STEP4

Note:-- In general step2 will executes even though the step1 is successful or not. For

Controlling this type of execution and some of the situation needs this ‘COND’ parameter.

COND=(COMPARISION CODE, RELATIONAL OPERATOR, RETURN CODE OF

PREVIOUSLY EXECUTED STEP)

Note:-- Return code specifies the status of a steps execution. It has a range of values.

00,04 🡪 upon successful execution of the step

08,12,……4095 🡪 upon unsuccessful execution of the step

If the ‘COND’ specified in the step is false then the step executes. And if it is true then the step is bypassed.

TRUE 🡪 Bypassed

FALSE 🡪 Executes

“COND=EVEN” it executes the step even if the previous steps executed successfully or unsuccessfully.

“COND=ONLY” it executes the step only if the previous step executes unsuccessful.

Note: COND=(00,LE) is always ‘TRUE’ condition it bypass the step all the time.

COND=(00,GT) is always ‘FALSE’ condition it executes the step all the time.

**DD Statements: --**

It is used to describe the characteristics of a dataset.

//DISK1 DD DSN=FSS045.IONETECH .FILE

Positional parameters - \*

DATA

DUMMY

DYNAM

Keyword Parameters - DSN

DISP

UNIT

SPACE

DCB

VOLUME

**DSN: -**

* It specifies the dataset name.
* It can be maximum of ‘44’ characters long
* Dataset name can contain alphabets, numeric data and national characters( @,$,#)
* Data set is made of qualifiers which can be maximum of 8 characters.
* A qualifier can begin with alphabetic or national characters and cannot end with ‘-‘.
* The first qualifier is called high level qualifier, which is usually the Project ID.
* The second qualifier is the module-id or application change request id.
* The qualifier is the type of the dataset.

Syntax 🡪 DSN=dataset name

Example JCL 🡪 //MYJOB JOB (ER3),'RAMESH R'

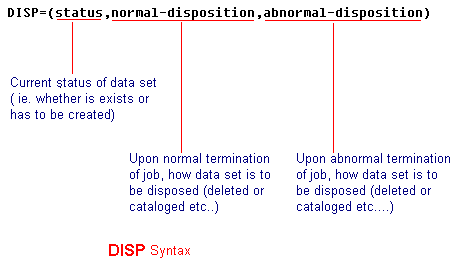
//STEP1 EXEC PGM=COBPROG

//INFILE DD DSN=TEST.GLOB.MYLIB

TEST.GLOB.MYLIB will be used in the COBPROG program

**DISP: --**

It tells the status of the dataset and what needs to be done to it upon successful or unsuccessful execution of the step.



**Parameter on the DISP statement**

**Status Normal Disposition Abnormal Disposition**

NEW DELETE DELETE

OLD CATLG CATLG

MOD UNCATLG KEEP

SHR KEEP UNCATLG

PASS

**STATUS:**

NEW - Dataset will be created. (file should not exists)

OLD - Dataset should exists.

SHR - Dataset should exists & Dataset can be used by other jobs also.

MOD - Dataset will be created If not exists.

**NORMAL DISPOSITION:**

(Happened upon successful execution of job step)

DELETE - Dataset should be deleted

CATLG - Dataset will be cataloged

UNCATLG - Dataset will be removed from system catalogs

KEEP - Dataset will be retained (This parameter should be used with permanent data sets)

PASS - Dataset is to be passed subsequent job step in the same job(This parameter should be used with temporary datasets)

**ABNORMAL DISPOSITION:**

(Happened upon unsuccessful execution of job step)

DELETE - Dataset should be deleted

CATLG - Dataset will be cataloged

UNCATLG - Dataset will be removed from system catalogs

KEEP - Dataset is to be kept

EXAMPLE JCL --> //MYJOB JOB (E674),'IONETECH

//STEP1 EXEC PGM=COBPROG

//INFILE DD DSN=TEST.GLOB.TEMP,

// DISP=(NEW,CATLG,DELETE)

In this example ,

DISP=(NEW,CATLG,DELETE)

NEW - TEST.GLOB.TEMP is not exists, it will be created

CATLG - Upon successful execution of job step, data set will be cataloged

DELETE - If job terminated abnormally, dataset will be deleted

**‘Temporary Datasets’:--**

The dataset created by the “step is available until the JOB is executed. After the completion of the JOB, the dataset is deleted.

“The temporary dataset is mentioned as ‘&&’ and **DISP=(NEW,PASS)**

* These are used for generating the reports.

//DISK DD DSN=&&TEMPDS,DISP=(NEW,PASS)

**DCB: --**(Data control block)

//DISK1 DD DCB=(LRECL=140,BLKSIZE=1400,RECFM=FB,DSORG=PS)

“LRECL” :-- It specifies the logical record length of the dataset.

BLKSIZE 1400

Number of records per block = --------------- = -------------- = 10 Records

LRECL 140

Note: -- Maximum records length in the block 🡪 32767

Maximum block size 🡪 32767

* When read statement is executed for the first time one block of records are returned from DASD

(system memory) to buffer.

**DSORG: --**

It specifies the dataset organization.

PS: physical sequential file

PO: partitioned organization PDS is created.

**RECFM: --**

1. It specifies the format of the records in the file whether they are fixed or variable
2. Accessing in fixed file is fast compared to variable file.
3. In fixed file some memory wastage. In fixed file no memory wastage.

RECFM = FB 🡪 Maximum records per block

= F 🡪 One record per block

=VB 🡪 Maximum records per block

= V 🡪 One record per block

EXAMPLE JCL 🡪 //MYJCL JOB (E3445),'RAMESH'

//STEP1 EXEC PROG=COBPROG

//INFILE DD DSN=TEST.GLOB.LIB

// UNIT=234,

// DSN=(LRECL=80,RECFM=FB, BLKSIZE=800, BUFNO=30)

[**UNIT PARAMETER**](http://www.mainframegurukul.com/srcsinc/drona/programming/languages/jcl/jcl.chapter6.html)**:**

In IBM Mainframe environment, All devices have an address assigned to them at the time they added to the system. Devices can be referenced using these addresses. UNIT parameter is used to specify this address.

Syntax 🡪 UNIT=device\_address/device\_type/device\_group\_name/TAPE

EXAMPLE JCL 🡪 //MYJOB JOB (R345),'IONETECH

//STEP1 EXEC PGM=COBPROG

//INFILE DD DSN=TEST.GLOB.TEST.LIB,

// UNIT=SYSDA

In above example, COBPROG is executed, and a file INFILE which may

reside on any of the devices which are grouped under the symbolic name

SYSDA will be accessed

Usually tape devices are used to take the backup of data.

[**VOL PARAMETER**](http://www.mainframegurukul.com/srcsinc/drona/programming/languages/jcl/jcl.chapter6.html)**:**

This parameter is used to identify the volume serial number on which dataset is reside. The VOL dataset is used with disk and tape datasets.

Syntax 🡪 VOL= volume label/data set label

Sub parameters used with VOL parameter

SER - Specification of serial number

REF - Referencing VOL specification from a prior step

PRIVATE - Allowing access to volume by single user

RETAIN - Inhibiting dismounting of volume until end of job

SEQ - Specification of sequence in which volumes are to be mounted

EXAMPLE JCL: 🡪 //MYJOB JOB (E454),'IONETECH’

//STEP1 EXEC PGM=COBPROG

//INFILE DD DSN=TEST.GLOB.TEMP,

// VOL=SER=(VOL1,VOL2,VOL3)

In this example the data set called DATA3 resides on 3 volumes whose

serial numbers are VOL1,VOL2,VOL3. The Operating system will request

that all volumes be mounted at the same time.

[**SPACE PARAMETER**](http://www.mainframegurukul.com/srcsinc/drona/programming/languages/jcl/jcl.chapter6.html)**:**

The SPACE parameter is used to allocate space for datasets. We can allocate space in Cylinders/Tracks/Blocks

Syntax 🡪 SPACE=(CYL,(primary,secondary,directory),RLSE,CONTIG,MXIG,ROUND)

Instead of CYL, We can use TRK or BLK

Meaning of Sub Parameter

TRK - Requesting space in track

CYL - Requesting space in cylinders

PRIMARY - Primary storage to be allocated at the time of dataset created

SECONDARY - Additional storage to be allocated, If primary storage is not sufficient

DIRECTORY - Space for recording of name and location of Partitioned data sets

RLSE - Request for release of space previously allocated unused space after completion of job

CONTIG - Request for contiguous space

MXIG - Request for large area of contiguous space

ROUND - Request for entire cylinder for storage of data set

EXAMPLE JCL 🡪 //MYJOB JOB (W345),IONETECH'

//STEP1 EXEC PGM=COBPROG

//INFILE DD DSN=TEST.GLOB.LIB

// UNIT=4560

// SPACE=(CYL,(30,4))

In this example, 30 cylinders are requested as primary space and 4 additional cylinders as secondary

space.

**DIRECTORY BLOC**K**S:** --

* It is used to store the members information.
* If directory block size is 0 then PS is created.
* If directory block size> 0 then PDS is created
* Each directory block stores 5 members information.
* Members information is nothing but the member name, creator name and when it was created.

**RLSE:** --

It is used to release any unused space from the dataset.

**RETPD: --(R**etention period)

Itspecifies the life of the dataset.

RETPD=DDDD

Where DDDD indicate number of days varying from 0001 till 9999

Ex: -- RETPD = 2345

The dataset is deleted after 2345 days

**EXPDT: --(**Expiration date)

EXPDT=YYDDD.

Where YY indicate year from 00 till 99

DDD indicate no.of days varying from 001 to 365.

Ex: -- EXPDT = 12045

12 INDICATES 2012, 045 means 45 days.

Note: -- VOL-SER, RETPD, EXPDT are mandatory for to create the data sets on TAPE file.

**POSITIONAL PARAMETERS IN DD STATEMENTS: --**

**SPECIAL DD STATEMENTS**

1. DUMMY or DSN=NULLFILE

2. Concatenating Data sets

3. Passing data to cobol program using - SYSIN DD \*

4. SYSOUT

5. SYSUDUMP

6. SYSABEND

7. SORTIN

8. SORTOUT

9. JOBLIB

10. STEPLIB

11. SYSUTN(N=1 to 9)

12.SYSPRINT

[**1. DUMMY or DSN=NULLFILE**](http://www.mainframegurukul.com/srcsinc/drona/programming/languages/jcl/jcl.chapter7.html)

Sometimes we need to test the program, without using actual datasets.

Then we can use DUMMY or DSN=NULLFILE.

If we use DUMMY, operating system simulates the presence of a file. When you reading operating system sends end of file request to program.

Example JCL 🡪 //MYJOB JOB (W345),'KRISHNA REDDY'

//STEP1 EXEC PGM=COBPROG

//INFILE DD DUMMY

[**2. CONCATENATING DATA SETS**](http://www.mainframegurukul.com/srcsinc/drona/programming/languages/jcl/jcl.chapter7.html)

In JCL, we can concatenate different files by giving their name one after another. All data sets concatenated must be of the same type. For example, partitioned data sets can be concatenated only with partitioned data sets.

Example JCL 🡪 //MYJOB JOB (W345),'IONETECH'

//STEP1 EXEC PGM=COBPROG

//INFILE DD DSN=TEST.GLOB.FILE1

// DD DSN=TEST.GLOB.FILE2

// DD DSN=TEST.GLOB.FILE3

In program, we will read it as a single file. Concatenation of three files done by operating system.

Note: --

Maximum of 255 sequential data sets can be concatenated together   
 Maximum of 16 partitioned data sets can be concatenated together

[**3. PASS DATA TO COBOL PROGRAM USING - SYSIN DD \***](http://www.mainframegurukul.com/srcsinc/drona/programming/languages/jcl/jcl.chapter7.html)

This is the one of the way of passing data to program. There are two ways to pass data.

Syntax1 🡪 //MYJOB JOB (W234),'RAMESH'

//STEP1 EXEC PGM=COBPROG

//SYSIN DD \*

/\*

Syntax2 🡪 //MYJOB JOB (E345),’IONETECH'

//STEP1 EXEC PGM=COBPROG

//SYSIN DD DATA

/\*

[**4. SYSOUT**](http://www.mainframegurukul.com/srcsinc/drona/programming/languages/jcl/jcl.chapter7.html)

The SYSOUT parameter is used to send the output which is generated during job execution.

Syntax 🡪 //ddname DD SYSOUT=class

EXAMPLE JCL 🡪 //MYJOB JOB (R456), ‘IONETECH '

//STEP1 EXEC PGM=COBPROG

//INFILE DD SYSOUT=A

In this example, COBPROG is executed and all generated outputs are directed to class ( here it

is letter A).

Note: --

If \* used with SYSOUT (SYSOUT=\*) parameter,The class assigned to the MSGCLASS parameter will be used to SYSOUT.

[**5. SYSUDUMP**](http://www.mainframegurukul.com/srcsinc/drona/programming/languages/jcl/jcl.chapter7.html)

SYSUDUMP is used to dump the content of various registers, variables and datasets accessed at the time of abnormal termination, into a dataset. The dump is in hexadecimal.

Syntax 🡪 //SYSUDUMP DD .....

EXAMPLE JCL: 🡪 //MYJOB JOB (W345),'KONDALU'

//STEP1 EXEC PGM=COBPROG

//SYSUDUMP DD DSN=TEST.PROD.LIB

[**6. SYSABEND**](http://www.mainframegurukul.com/srcsinc/drona/programming/languages/jcl/jcl.chapter7.html)

SYSABEND is used to dump the contents of various registers variables, datasets accessed and **the nucleus** at the time of abnormal termination. The dump is in hexadecimal.

Syntax 🡪 //SYSABEND DD .....

EXAMPLE JCL: 🡪 //MYJOB JOB (WE345),'IONETECH'

//STEP1 EXEC PGM=COBPROG

**7.SORTIN**

SORTIN is special dd name for SORT utility. SORTIN used to specify the input data set name.

Syntax 🡪 //SORTIN DD DSN= .....

EXAMPLE JCL: 🡪 //MYJOB JOB (W345),'IONE'

//STEP1 EXEC PGM=SORT

//SORTIN DD DSN=FSS045.TEST.PSIN

**8.SORTOUT**

SORTOUT is special dd name for SORT utility. SORTOUT used to specify the output data set name.

Syntax 🡪 //SORTIN DD DSN= .....

EXAMPLE JCL: 🡪 //MYJOB JOB (W345),'IONE'

//STEP1 EXEC PGM=SORT

//SORTOUT DD DSN=FSS045.TEST.PSOUT

**9. JOBLIB**

JOBLIB used to specify the Load library data set at the job level.

Syntax: //JOBLIB DD DSN= --------------------

EXAMPLE JCL: 🡪 //MYJOB JOB (W345),'KONDALU'

//JOBLIB DD DSN=FSS045.TEST.LOADLIB

//STEP1 EXEC PGM=COBPROG

**10. STEPLIB**

STEPLIB used to specify the Load library data set at the step level.

Syntax: //STEPLIB DD DSN= --------------------

EXAMPLE JCL: 🡪 //MYJOB JOB (W345),'KONDALU'

//STEP1 EXEC PGM=COBPROG

//STEPLIB DD DSN=FSS045.TEST.LOADLIB1

**11. SYSUTN(N=1 to 9)**

SYSUT1 & SYSUT2 are the special DD Names for IEBGENER.

Syntax: //SYSUT1 DD DSN=FSS045.TEST.PSIN

//SYSUT2 DD DSN=FSS045.TEST.PSOUT

EXAMPLE JCL: 🡪 //MYJOB JOB (W345),'KONDALU'

//STEP1 EXEC PGM=IEBGENER

//SYSUT1 DD DSN=FSS045.TEST.PSIN

//SYSUT2 DD DSN=FSS045.TEST.PSOUT

**INSTREAM and CATALOGUED PROCEDURES**

In JCL, We have an important concept reusability in the form of Instream and Cataloged procedures, Often, in work environments users can utilized same JCL. Using Instream /Cataloged procedures.

We can reuse the JCL code which was stored in another data set, in our data set.

Syntax for executing procedure

🡪 EXEC [PROC=]procedure-name

[**INSTREAM PROCEDURE**](http://www.mainframetutorials.com/drona/programming/languages/jcl/jcl.chapter8.html)**:**

A JCL Procedure is a pre-written segment of code that you can include in your JOB. You code Instream data set within the job and use it in that job as many times as you want.

An Instream procedure JCL Example:

//FSS045 JOB JOBNAME,’IONETECH’,NOTIFY=&SYSUID,MESCLASS=X,

// MESLEVEL=(1,1)

//MYPROC PROC

//STEP1 EXEC PGM=COBPROG

//SYSOUT DD SYSOUT=\*

//INFILE DD DSN=FSS045.IONETECH.JCL,DISP=SHR

//OUTFILE DD DSN=(NEW,DELETE,DELETE),

// UNIT=SYSDA,SPACE=(CYL,(20,10))

//SYSOUT DD SYSOUT=\*

//INFILE1 DD DSN=FSS045.IONETECH.JCL,DISP=SHR

// PEND

//STEP1 EXEC PGM=MAINPGM

//MYFILE DD DSN=FSS045.IONETECH.SPACE.JCL,DISP=SHR

//STEP2 DD MYPROC

//STEP3 DD MYPROC

//

[EXPLANATION](http://www.mainframetutorials.com/drona/programming/languages/jcl/jcl.chapter8.html):

Instream procedure should be defined, before any EXEC statement defined

Instream procedure starts with PROC and ends with PEND statements

Instream procedure is executed when we main JCL called.

Note: --

The maximum number of Instream procedures you can in any job is 15.

[**CATALOGED PROCEDURE**](http://www.mainframetutorials.com/drona/programming/languages/jcl/jcl.chapter8.html)**:**

Pre-written segment of code (which is stored as an member of PDS), which you can use as many times you want in any job in the system. IBM supplies a utility program called IEBUPDTE; this program places cataloged procedures into partitioned data sets. These procedures are placed inside a system library called SYS1.PROCLIB.

* Cataloged procedure is a PDS member in which PROC statements are stored.
* This PROC can be executed by any no. of times by any no. of JOBS.

Developing Catalog Procedure

**STEP1:**

Write an Cataloged procedure in MYLIB.EXAMPLES.TEST(CATALOG1)

//CATLOG1 PROC

//STEP1 EXEC PGM=COBPROG

//INFILE DD DSN=TEST.GLOB.LIB,

// DISP=SHR

//OUTFILE DD DSN=TEST.GLOB.SPACE.LIB,

// DISP=SHR

**STEP2 :**

Write Main JCL which will call out CATALOG1 JCL

//MYJOB JOB (WE234),'RAMESH',CLASS=A

// JCLLIB ORDER=(MYLIB.EXAMPLES.TEST) 🡨 Attention

//STEP1 EXEC CATALOG1 🡨 Attention

//

**EXPLANATION**

When you executing CATALOGED PROCEDURE, If you not specified where it is with JCLLIB statement, it will search for this procedure in system procedure library SYS1.PROCLIB

There many IBM-supplied procedures that compile, link, and run programs

Note: --

The following statements cannot be included within the procedure  
 JOB  
 DD \* or DD DATA  
 JES2 or JES3 control statements

|  |  |
| --- | --- |
| **INSTREAM PROCEDURE** | **CATALOGED PROCEDURE** |
| 1.Statements are defined and executed with in a  Single JCL. | 1. Defined in a separate PDS member. |
| 2. Executed by a single JOB only. | 2. Executed by multiple JOBS. |
| 3.Can be executed for a maximum of 15 times | 3.Any number of times |
| 4. PEND statement is required. | 4. Not required. |
| 5. No JCLLIB is required. | 5. JCLLIB is required. |

**CONTROL CARD: --**

1. It is a PDS member or PS in which the Instream data is stored.
2. Using control card will reduce coding effort and avoid duplication of work.

**Q).** Can we pass Instream data to a PROC?

Ans: - YES, we can pass Instream data to a PROC by using PARM& Control card.

**SYMBOLIC PARAMETERS: --**

These parameters are used to substitute the values in the DD statements.

These are identify by using ampersand symbol ‘&’.

Note: --

1. If symbolic parameters values are specified in both JOB and PROC then the value specified in JOB overrides that of PROC.
2. If we are the cataloged PROC then the JESJCL of SPOOL will contains the “XX” in place of the expanded PROC as per the above.
3. If we are calling the Instream PROC then the JESJCL of SPOOL will contains the “++” in place of expanded PROC.

**OVERRIDING PARAMETER: --**

These parameters are used to override the already existing values for the EXEC statements parameters except “PARM”.

**Syntax: --**

**\*\*\*\*\*\***stepname.parameter=new value

**UTILITIES**

**1.IEFBR14: --**

This is nothing more than the null program. This program is used to code functions that are commonly available on DD statement, such as the creation, deletion and updating of datasets, without having to execute any explicit program.

**2.IEBGENER: --**

This utility is used to copy data from one PS to another PS.

Ex: -- // FSS045AB JOB NOTIFY=&SYSUID

//S1 EXEC PGM=IEBGENER

//SYSPRINT DD SYSOUT=\*

//SYSUT1 DD DSN=FSS045.RAVINDRA.PS300,DISP=SHR

//SYSUT2 DD DSN=FSS045.RAVINDRA.PS500,DISP=MOD

//SYSIN DD DUMMY

Note: --

SYSUT1,SYSUT2 are system defined DD names.

1. If more than one file is specified under single DD name, system treats files as a single file. This is called as concatenation of datasets.

Ex: -- //STEPNAME DD DSN=FSSO45.IONETECH.FILE1

// DD DSN=FSS045.IONETECH.FILE2

1. If the files are of different record lengths there specifying the files in the descending order of their record lengths. Otherwise the data truncation will be occurred.

**3.IEBCOPY: --**

It is used to copy members data from one PDS to another PDS.

Ex: -- //FSS045AB JOB NOTIFY=&SYSUID

//S1 EXEC PGM=IEBCOPY

//SYSPRINT DD SYSOUT=\*

//DD1 DD DSN=FSS045.RAVINDRA.PDS1,DISP=SHR

//DD2 DD DSN=FSS045.RAVINDRA.PDS2,DISP=SHR

//SYSIN DD \*

COPY INDD=DD1,OUTDD=DD2

/\*

To copy only selected members: --

//SYSIN DD \*

COPY INDD=DD1,OUTDD=DD2

SELECT MEMBER=(MEM4,MEM5)

/\*

To exclude members for copying: --

//SYSIN DD \*

COPY INDD=DD1,OUTDD=DD2

EXCLUDE MEMBER=(MEM4,MEM5)

/\*

**4. IDCAMS: --**

It is the primary vehicle used by the AMS program to process VSAM and NONVSAM datasets.

We can delete multiple number of datasets at a time. Whereas by using IEFBR14 we can delete only one dataset at a time.

//JOB1 JOB

//STEP1 EXEC PGM=IDCAMS

//SYSIN DD\*

DELETE ‘FSS045.IONETECH.SOURCE’

DELETE ‘FSS045.KONDALU.SOURCE’

/\*

//

**5.SORT: --**

Sort utility is used to sort the data in a file.

//FSS045AB JOB NOTIFY=&SYSUID

//S1 EXEC PGM=SORT

//SYSPRINT DD SYSOUT=\*

//SORTIN DD DSN=FSS045.IONETECH.PS,DISP=SHR

//SORTOUT DD DSN=FSS045.IONETECH.PS1,DISP=SHR

//SYSOUT DD SYSOUT=\*

//SYSIN DD \*

SORT FIELDS=(1,14,CH,A)

/\*

1. To COPY data from one file to another file.

//FSS045AB JOB NOTIFY=&SYSUID

//\* COPY THE DATA FROM INPUT FILE TO OUTPUT FILE \*\*\*\*/

//S1 EXEC PGM=SORT

//SYSPRINT DD SYSOUT=\*

//SORTIN DD DSN=FSS045.RAVINDRA.PS300,DISP=SHR

//SORTOUT DD DSN=FSS045.RAVINDRA.PS400,DISP=SHR

//SYSOUT DD SYSOUT=\*

//SYSIN DD \*

SORT FIELDS=COPY

/\*

2. To SORT the input file based on the multiple conditions.

//FSS045YY JOB NOTIFY=&SYSUID

//S1 EXEC PGM=SORT

//SYSPRINT DD SYSOUT=\*

//SORTIN DD DSN=FSS045.RAVINDRA.PS2,DISP=SHR

//SORTOUT DD DSN=FSS045.RAVINDRA.PS3,DISP=SHR

//SYSOUT DD SYSOUT=\*

//SYSIN DD \*

SORT FIELDS=COPY

INCLUDE COND=(5,2,CH,EQ,C'A1') AND (10,2,CH,EQ,C'B1')

/\*

1. To OMIT the records based on the condition

//FSS045YY JOB NOTIFY=&SYSUID

//S1 EXEC PGM=SORT

//SYSPRINT DD SYSOUT=\*

//SORTIN DD DSN=FSS045.IONETECH.PS2,DISP=SHR

//SORTOUT DD DSN=FSS045.IONETECH.PS3,DISP=SHR

//SYSOUT DD SYSOUT=\*

//SYSIN DD \*

SORT FIELDS=COPY

OMIT COND=(5,2,CH,EQ,C'A1')

/\*

3. To REMOVE the duplicate records in the input file

//FSS045YY JOB NOTIFY=&SYSUID

//S1 EXEC PGM=SORT

//SYSPRINT DD SYSOUT=\*

//SORTIN DD DSN=FSS045. IONETECH.PS2,DISP=SHR

//SORTOUT DD DSN=FSS045. IONETECH.PS3,DISP=SHR

//SYSOUT DD SYSOUT=\*

//SYSIN DD \*

SORT FIELDS=(1,10,CH,A)

SUM FIELDS=NONE

/\*

4. To write the duplicate records into another file

//FSS045AB JOB NOTIFY=&SYSUID

//S1 EXEC PGM=SORT

//SYSPRINT DD SYSOUT=\*

//SORTIN DD DSN=FSS045. IONETECH.PS2,DISP=SHR

//SORTOUT DD DSN=FSS045. IONETECH.PS3,DISP=SHR

//SYSOUT DD SYSOUT=\*

//SYSIN DD \*

SORT FIELDS=(1,10,CH,A)

SUM FIELDS=NONE,XSUM

/\*

//SORTXSUM DD DSN=FSS045. IONETECH.PS,DISP=SHR

5. To SPLIT the input file into multiple output files.

//FSS045YY JOB NOTIFY=&SYSUID

//S1 EXEC PGM=SORT

//SYSPRINT DD SYSOUT=\*

//SORTIN DD DSN=FSS045. IONETECH.PS2,DISP=SHR

//SORTOF01 DD DSN=FSS045. IONETECH.PS,DISP=SHR

//SORTOF02 DD DSN=FSS045. IONETECH.PS3,DISP=SHR

//SORTOF03 DD DSN=FSS045. IONETECH.PS4,DISP=SHR

//SYSOUT DD SYSOUT=\*

//SYSIN DD \*

SORT FIELDS=COPY

OUTFIL FILES=01,STARTREC=1,ENDREC=3

OUTFIL FILES=02,STARTREC=4,ENDREC=6

OUTFIL FILES=03,STARTREC=7,ENDREC=10

/\*

6. To COPY only the particular fields from input file

//FSS045YY JOB NOTIFY=&SYSUID

//S1 EXEC PGM=SORT

//SYSPRINT DD SYSOUT=\*

//SORTIN DD DSN=FSS045.IONETECH.PS2,DISP=SHR

//SORTOUT DD DSN=FSS045.IONETECH.PS,DISP=SHR

//SYSOUT DD SYSOUT=\*

//SYSIN DD \*

SORT FIELDS=COPY

INREC FIELDS=(1:1,5,6:5,10)

/\*

**REFERBACK: --**

It is used to take the attributes of an already existing dataset.

Syntax: --

“DCB=\*.STEPNAME.DD NAME”

Refer back can be used for DCB, SPACE, VOL-SER parameter.

//FSS045AB JOB NOTIFY=&SYSUID

//S1 EXEC PGM=IEBGENER

//SYSPRINT DD SYSOUT=\*

//SYSUT1 DD DSN=FSS045.IONETCH.PS1,DISP=SHR

//SYSUT2 DD DSN=FSS045.IONETECH.PS3,

// DISP=(NEW,CATLG,DELETE),

// DCB=\*.S1.SYSUT1,

// SPACE=(TRK,(10,10),RLSE)

//SYSIN DD DUMMY

Note: --

By using \* we can assign the parameters of the previous PDS or PS dataset by specifying like

DCB=\*.STEPNAME

**GDG(GENERATION DATA GROUP)**

Generation Data Groups or GDGs are a group of data sets which are related to each other chronologically and functionally. These related Data Sets share a unique Data Set Name.

Every GDG data set has a Generation number and Version number assigned to each data set.

**Uses: --**

1. Used to take back-up of data.
2. It automates the process of creating and deleting datasets.
3. We need not change the JCL when we need to create a new file.

**EXAMPLE** -- 'MYLIB.LIB.TEST.G0001V00'

'MYLIB.LIB.TEST.G0002V00'

'MYLIB.LIB.TEST.G0003V00' 🡨 Current Version

Generation Number 🡪GaaaaVnn

aaaa is between 0000 to 9999

nn is between 00 to 99

In JCL, we refer current version with 0 (Ex. MYLIB.LIB.TEST (0))

New version going to create with +1 (Ex. MYLIB.LIB.TEST (+1))

Older versions refer with -1 -2 -3 etc....

(Ex. MYLIB.LIB.TEST (-1)) 🡨 OLDER VERSION

**Example for where we can use this GDGs.**

Usually, In production environment, for every month we need to run jobs to create reports for that month.

Let us suppose for January, We can code it **MYLIB.LIB.TEST.JAN**

for FEB, We can code it **MYLIB.LIB.TEST.FEB**

for MAR, We can code it **MYLIB.LIB.TEST.MAR**

So, Every month we need change dataset name in JCL, before submitting the job. Whenever we entered into another year, we need to delete old years of datasets.

We need to do above task carefully, If we use GDG, It will take care following things

* + It will maintain all generation of data sets
  + It will delete/uncatalog older generation
  + Very easily, we can refer current and older versions of data sets
  + No need of change the JCL every time when we submit

**CREATION OF GDG: --]**

Before using GDG , We need to create GDG index and model. IDCAMS (the 'AMS' stands for Access Method Services), utility is used to create GDG index.

**Example JCL for creating GDG index**

//MYJOB JOB (W234),'IONETECH'

//STEP1 EXEC PGM=IDCAMS

//SYSIN DD \*

DEFINE GDG(NAME(MYLIB.LIB.TEST) -

LIMIT(10) -

NOEMPTY -

SCRATCH)

/\*

//

In this example, IDCAMS utility is used to create an index for a GDG called MYLIB.LIB.TEST. The number of generations that can exist in this GDG is limited to ten. NOEMPTY parameter is used to specify, once the limit is reached, the system is instructed to uncatalog the oldest generation data set within the GDG. SCRATCH parameter is used to specify to physically delete the data set which was uncatalogued.

**PARAMETERS WE CAN PASS TO IDCAMS**

NAME - This parameter is used to specify the name of the data set that is to be created.

LIMIT - This parameter is used to specify the total number of generations that the GDG may

Contain

EMPTY/NOEMPTY - These two parameters are mutually exclusive. EMPTY specifies that all existing generations of the GDG are to be uncataloged whenever the generations of GDG

Reached the maximum limit NOEMPTY specifies that only the oldest generation of the GDG is to be uncataloged if the limit is reached

SCRATCH/NOSCRATCH - These two parameters are mutually exclusive. SCRATCH parameter specifies that whenever entry of the GDG is removed from the index, it should be deleted physically and uncataloged. NOSCRATCH parameter specifies that whenever entry of the GDG is removed from the index, it should be uncataloged, not physically deleted.

Note: --

SCRATCH and NOEMPTY are default parameters

**CREATING MODEL**

Once the index has been created, a model data set must be created. This model data set contains specifications for the DCB sub parameters for all data sets that will belong to that GDG. Programmer can override this default values if he want.

**EXAMPLE JCL**

//MYJOB JOB (W983),'KRISHNA REDDY'

//STEP1 EXEC PGM=IDCAMS

//SYSIN DD \*

DEFINE GDG( -

NAME(MYLIB.LIB.TEST) -

LIMIT(10) -

NOEMPTY -

SCRATCH)

[**USING GDG**](http://www.mainframetutorials.com/drona/programming/languages/jcl/jcl.chapter9.html)

To use created GDG in our JCL, we need to use name (with +1 for new generation) which we used in DEFINE GDG command. (i.e. MYLIB.LIB.TEST)

**EXAMPLE JCL**

//MYJOB JOB (SD345),'KRISHNA REDDY'

//STEP1 EXEC PGM=COBPROG

//INFILE DD DSN=MYLIB.LIB.TEST(+1),

// DISP=(NEW,CATLG,DELETE),

// UNIT=SYSDA,

// SPACE=(TRK,(20,10),RLSE),

// DCB=(LRECL=80,RECFM=FB,BLKSIZE=800)

The program COBPROG is executed. A new generation data set is created via the statement

//INFILE DD DSN=MYLIB.LIB.TEST(+1)

Since we used (+1) with GDG name, it creates a new generation data set.

The DISP parameter must be set to CATLG for all new generation data sets ,

DISP=(NEW,CATLG,DELETE)

We used MODEL.DCB in DCB parameter to instruct system to use Sub parameters specified in model GDG.

Note: --

The DSN and UNIT parameters must be coded for all new generation data sets

[**4. ALTERING GDG DEFINITION**](http://www.mainframetutorials.com/drona/programming/languages/jcl/jcl.chapter9.html)

Sometimes there are situations where we need to change the attributes of GDG. These types of tasks can be performed using ALTER command. We will use IDCAMS utility to alter GDG attributes.

In last section, I have create MYLIB.LIB.TEST GDG with NOEMPTY SCRATCH Sub parameters, now I want to change them to EMPTY NOSCRATCH respectively.

Here is the JCL that will do this:

//MYJOB JOB (WE345),'KRISHNA'

//STEP1 EXEC PGM=IDCAMS

//SYSPRINT DD SYSOUT=A

//SYSIN DD \*

ALTER MYLIB.LIB.TEST EMPTY NOSCRATCH

/\*

//

In this example, the ALTER statement is used to modify the features of the GDG called MYLIB.LIB.TEST. Any generations that may exist for that GDG will now contain the modified features as well. Any new generations that are created for this GDG will now be created based on these new features.

[**DELETING GDG**](http://www.mainframetutorials.com/drona/programming/languages/jcl/jcl.chapter9.html)

We can delete a generation of GDG with IEFBR14, Here is the JCL to do that

//MYJOB JOB (ER456),'RAMESH'

//STEP1 EXEC PGM=IEFBR14

//DEL1 DD DSN=MYLIB.LIB.TEST(0) 🡨 Current Version

// DISP=(OLD,DELETE,DELETE)

//

In this example JCL, the program IEFBR14 is executed. The current generation of MYLIB.LIB.TEST is deleted.

To delete GDG index/generations , We need to use DELETE command in IDCAMS utility. There are two sub parameters we can use with DELETE command.

They are PURGE and FORCE

PURGE sub parameter is used in conjunction with DELETE statement to delete the GDG index, even if its retention period has not expired.

FORCE parameter can be coded on the DELETE statement to delete the GDG index, the model, and all related generation data sets, if they exist.

**EXAMPLE:** JCL for FORCE

//MYJOB JOB (W234),'KRISHNA'

//STEP1 EXEC PGM=IDCAMS

//SYSIN DD \*

DELETE (MYLIB.LIB.TEST) GDG FORCE

/\*

MYLIB.LIB.TEST GDG index, the model and all related generation data sets will be deleted upon successful execution of this job step.

Note: --

Maximum of 255 data sets exist within one GDG.

[**IMPORTANT PRACTICAL QUESTIONS**](http://www.mainframetutorials.com/drona/programming/languages/jcl/jcl.chapter9.html)**:**

Q. In my JCL, In step1 I will going to create a new generation data set for that i gave GDG name (+1). In

Step2 I want to use same data set created by previous step? What number i should give to refer that data

Set (i.e. 0 or +1 or +2)? (STEP1 EXECUTED SUCCESSFULLY)

A. +1

Q. Why?

A. Even step1 executed successfully, it is not become the current generation of GDG. At the end of the

Job only it will become the current version of GDG. So within the job we need to refer it as new

Generation only, even that step completed successfully.

**COBOL**

(COMMON BUSINESS ORIENTED LANGUAGE)

# HISTORY

Developed by a group called CODASYL (Conference on Data Systems language) in 1959.

Best programming Language for Business Applications

First COBOL compiler was released in December 1959

First ANSI Approved Version – 1968

Modified ANSI Approved Version – 1974 (OS/VS COBOL)

Modified ANSI Approved Version – 1985 (VS COBOL-2)

**Advantages**

The First programming Language developed for Business Applications.

Structured programming Language

Platform Independent

Free Source

**Disadvantages**

Very lengthy coding

Can’t be used for scientific calculations

# Structure of COBOL Programming Language

DIVISIONS

SECTIONS

STATEMENTS

SENTENCES

PARAGRAPHS

## DIVISIONS in COBOL

1. IDENTIFICATION DIVISION

2. ENVIRONMENT DIVISION

3. DATA DIVISION

4. PROCEDURE DIVISION

**1.IDENTIFICATION DIVISION**

It provides program details to the system

It has no sections but has following paragraphs

1. PROGRAM-ID.PGM. Specifies the program name.
2. Author.PRADEP . Specifies the developer name who coded

theProgram

1. Date-written.19-03-2012./19thMAR2012. Specifies the date on which program has

been written

1. Date-compiled.19-03-2012. Specifies the date on which program has

Been Compiled

1. Installation.boa Specifies the client name for whom the

Programis being developed.

**2.ENVIRONMENT DIVISION**

It has 2 sections.

a). CONFIGURATION SECTION

It has following Paragraphs.

i). SOURCE-COMUPTER.IBM3278.Specifies the computer model used for compilation of program.

ii). OBJECT-COMPUTER.IBM3270.Specifies the computer model used for execution of program.

b). INPUT-OUTPUT SECTION

This section is mandatory when Files are used. It has two Paragraphs.

i). FILE-CONTROL Files are declared in this paragraph.

ii). I-O-CONTROL Check points are set on the file data.

**3.DATA DIVISION**

All kinds of fields and variable declarations are done here.

It is used to define the data that needs to be accessed and processed by the program.

It has following Sections.

a). FILE-SECTION

File related fields are declared here.

Describes the record structure of the File

SYNTAX:

|  |
| --- |
| DATA DIVISION.  FILE SECTION.  FD INFILE.  01 IN-REC PIC X(80).  FD OUTFILE.  01 OUT-REC PIC X(80). |

b). WORKING-STORAGE SECTION

Temporary or Intermediate variables are declared in this section.

SYNTAX:

WORKING-STORAGE SECTION.

01 ENAME PIC A(5) VALUE 'ANIL'.

c). LINKAGE SECTION

Fields that are used to pass data between programs are declared in this Section.

SYNTAX:

LINKAGE SECTION.

01 LS-A PIC X(3).

d). REPORT SECTION

e). SCREEN SECTION

**4. PROCEDURE DIVISION**

It has no system defined sections or paragraphs.

All the COBOL executable statements/logics/instructions are written in this Section.

The execution begins from this section.

The user defined Section name should be unique within the program.

The user defined Paragraph name should be unique within the Section.

Note: Identification and Procedure Divisions are mandatory, Data Division will be used as per requirement.

**STEPS INVOLVED IN DEVELOPING COBOL PROGRAM:**

**DESIGN**

**CODE**

**COMPILE -** Process of identifying syntax errors & converting source code into object code.

**LINK EDIT -** Linking object codes of one or more related programs to single executable load module.

**EXECUTE**

**COBOL STRUCTURE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 2 3 4 5 6 | 7 | 8 9 10 11 | 12……………..72 | 73…………80 |
| Sequence number | Special Line   * \* / | Area A  Divisions/sections/paragraphs/  Level numbers | Area B  Cobol Statements | Comments |

Note: 7th column is used for below purposes.

\* comment line

- Continuation of a string in next line

/ print from the next page

1 2 3 4 5 6 7 8 9 10 12 13 ………………………….

0 0 0 0 0 1 IDENTIFICATION DIVISION.

0 0 0 0 0 2 PROGRAM-ID. SAMPLE.

0 0 0 0 0 3 DATA DIVISION.

0 0 0 0 0 4 WORKING-STORAGE SECTION.

0 0 0 0 0 5 01 EMP-NAME PIC X(10) VALUE 'ANIL'.

0 0 0 0 0 6 PROCEDURE DIVISION.

0 0 0 0 0 8 DISPLAY EMP-NAME.

0 0 0 0 0 9 DISPLAY 'EMP-NAME'.

0 0 0 0 1 0 DISPLAY 'EMPLOYEE NAME IS:' EMP-NAME.

0 0 0 0 1 1 STOP RUN.

**FIELD DECLARATIONS**

WORKING-STORAGE SECTION.

01 EMP-NAME PIC X(10) VALUE 'IBM'.

Level No Field Name Picture Clause Data Type Data Length Literal

**🡪DATA LENGTH** – Specifies the max no. of characters allocated to a field. For N characters system

allocates N bytes of memory.

🡪**FIELD NAME** used to identify specific memory in buffer & can max 32 characters long. It can be

Alphabetic/numeric/national characters. Field name cannot end with –

🡪**LEVEL NO** describes the data hierarchy in COBOL. We can use level numbers 01 to 49 & special

Level numbers like 66, 77,88.

66 – Renames

77 – Elementary

88 – Condition

🡪**VALUE** clause is used to assign a value to a field or variable statically & reduces the compilation time.

Ex: 01 EMP-NAME PIC X(10) VALUE 'IBM MAINFRAME'.

🡪**DISPLAY** is used to display content of a field in output, can also display more than one field’s content.

Ex: DISPLAY ENAME.

DISPLAY ‘ESAL IS:’ESAL.

DISPLAY ENAME,ESAL.

🡪**Literal** is a constant data that is assigned to a field. For alpha & alpha numeric literal must be

enclosed in quotes.

🡪**ACCEPT** statement is used to assign data to a field dynamically.

Accept statement can take only one variable at a time.

To accept N values we need to write N Accept statements.

Accept can take modified values dynamically without changing the program.

Ex: ACCEPT EMP-NAME.

ACCEPT EMP-SAL.

🡪**STOP RUN** is used to terminate the program and pass the control to the operating system. We can use

more than one STOP RUN in a program but only one STOP RUN will be executed.

**PICTURE CLAUSE**

Group items are defined by a level no. & a name which is followed by a period.

Elementary items must be described with a picture clause.

Picture clause specifies the type of data contained in an elementary item & size of the field.

Ex: WORKING-STORAGE SECTION

01 I PIC 9(2).

**DATA TYPES**

It controls the entry of the data into a field. There are 3 data types in COBOL.

1). Alpha Numeric (X):

It allows Alphabets (A-Z), Numerical(0-9) and all other special characters.

The Maximum data length can be 32767.

Data alignment is right justified.

Data is stored from left byte onwards remaining fields are spaces.

Ex: 01 EMP-ADDRESS PIC X(16) VALUE '0/1,IONE'.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 0 | / | 1 | , | I | O | N | E |  |  |  |  |  |  |  |  |

2). Alphabetic (A):

It allows only Alphabets (A-Z).

Maximum data length is 32767, Minimum is 1.

Data alignment is left justified.

Default values are spaces.

Ex: 01 EMP-NAME PIC A(16) VALUE 'VISHAL'.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| V | I | S | H | A | L |  |  |  |  |  |  |  |  |  |  |

3). Numeric (9):

It allows only numeric data (0-9).

Maximum data length is 18, Minimum is 1.

Data alignment is right justified.

Default values are zero’s

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 2 | 3 |

**\*\*ON SIZE ERROR** is executed only when there is a data truncation in the resultant field.

SCOPE TERMINATOR:

It is used to terminate a COBOL statement instead of period (.)

Ex: IF COMPUTE EVALUATE PERFORM

END-IF. END-COMPUTE. END-EVALUATE. END- PERFORM.

**Arithmetic Operations Cobol:**

Cobol can perform arithmetic operations likeAddition,Subtraction,Multiplication,Division…..etc

Syntax:

compute d= a +b + c

compute d = a - b - c

compute d = a \* b \*C

COMPUTE D= A/B/C

**Example1:-**

Identification division.

Program-id.

Data division.

01 a 9(2) value 4.

02 b 9(2) value 5.

03 c 9(2) value 9.

04 d 9(2) .

Procedure division.

Initialize d

compute d= a +b + c

Display ‘value OF D IS’ d

compute d = a - b - c

Display ‘value OF d IS’ d

compute d = a \* b \*C

Display ‘value OF d IS’ d

COMPUTE D= A/B/C

Display ‘value OF d IS’ d

Display ‘value OF A IS’ a

Display ‘value OF B IS’ b

Display ‘value OF C IS’ c

Stop run.

**Example2:-**

Identification division.

Program-id.

Data division.

01 b 9(2) value 5.

01 c 9(2) value 3.

01 d 9(2) value 6.

01 e 9(2) value 2.

04 a 9(3) .

Procedure division.

Initialize a

Compute a=(((b+c)-d)/e)

Display’value of aIS ’ a

Stop run.

**Edited Picture clause:**

Edited picture clause used to accept the negative values& Packed decimal values, for suppressing of zeros.

**Type1:**This edited pic clause used for display purpose either is pic class or neither value depends upon the situation. They are 2 types of pic clause.

-9(3) 🡪 for display negative value

s9(3) 🡪 for aCCEPTING negative value

Note: -9(3) is an edited Picture clause.

**Syntax:**

01 ws-a pic -9(3)

01 ws-b pic s9(3)

Procedure division.

Move -121 to ws-b

Move ws-b to ws-a

Display ‘ws-a=’ ws-a

**Example1:**

identification division

program-id.

Data division

Working-storage section.

01 ws-a pic -9(3).

01 ws-b pic s9(3).

Procedure division.

Initialize ws-a ws-b

Move -121 to ws-b

Move ws-b to ws-a

Display ‘ws-b’ ws-b

Display ‘ws-a’ ws-a

Stop run.

**Type2:**

To display the decimal values should use the edited picture clause.

🡪 9(2).9(2) is anEdited pic clause

🡪 9(2)v9(2) is not Edited pic clause.

**Syntax:**

01 ws-b pic 9(3)v9(3) value 99.22

**Example**:-

identification division

program-id.

Data division

Working-storage section.

01 ws-a pic 9(2).9(2)

01 ws-b pic 9(2)v9(2) value 99.22.

Procedure division.

Initialize ws-a

Move ws-b to ws-a

Display ‘point’ ws-b

Display ‘ws-a’ ws-b

Stop run.

o/p:- ws-a🡪99.22

ws-b🡪9922

**Type3:**

Using edited picture clause suppressing of zeros is possible.

**Synatx:**

01 WS-A PIC Z9(3).

Note:

Z can suppress one zero in the value.

ZZ suppress 2 zeros in the input value.

**Example:**

identification division

program-id.

Data division

Working-storage section.

01 ws-a pic z9(5).

Procedure division.

Initialize ws-a

Move 0123 to ws-a

Display ‘ws-a’ ws-a

Move 00123 to ws-a

Display ‘ws-a’ ws-a

Stop run.

**IF:** It is used to conditionally execute the COBOL statements. If condition is true it’ll execute the IF part

otherwise it’ll execute the ELSE part.

SIMPLE IF: IF COND

END-IF.

NESTED IF: IF COND

ELSE

IF COND

ELSE

END-IF.

END-IF

**EVALUATE**

It is used to execute the statements based on condition.

When statement is used to specify the condition. We can use maximum of 255 when conditions

When Other is always a true condition and hence must be specified as last when condition.

THRU or THROUGH is used to specify a range of values.

END-EVALUATE is the mandatory scope terminator.

IDENTIFICATION DIVISION.

PROGRAM-ID. EVAL.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 GROUP1 PIC 9(2).

88 A VALUE 1 THRU 5.

88 B VALUE 6 THRU 10.

88 C VALUE 11 THRU 15.

88 D VALUE 16 THRU 20.

PROCEDURE DIVISION.

ACCEPT GROUP1

EVALUATE TRUE

WHEN ( GROUP1 > 1 AND GROUP1 < 5)

DISPLAY 'VALUE OF A IS:' GROUP1

WHEN ( GROUP1 > 6 AND GROUP1 < 10)

DISPLAY 'VALUE OF B IS:' GROUP1

WHEN ( GROUP1 > 11 AND GROUP1 < 15)

DISPLAY 'VALUE OF C IS:' GROUP1

WHEN ( GROUP1 > 16 AND GROUP1 < 20)

DISPLAY 'VALUE OF D IS:' GROUP1

WHEN OTHER

DISPLAY 'INVALID VALUE '

END-EVALUATE.

STOP RUN.

**Continue Statement:**

If the condition is true then CONTINUE statement execute the statements which are after the last Scope terminator.

**Syntax:**

If(ws-b>ws-a)

Continue

Else (ws-b>ws-c)

Display’value of b’ws-b

End-if

Display’hello’

Stop run.

**Example:**

identification division

program-id.

Data division

Working-storage section.

01 ws-a pic 9(1) value 8.

01 ws-b pic 9(1) value 9.

01 ws-C pic 9(1) value 5.

Procedure division.

If(ws-b>ws-a)

Continue

Else if(ws-b>ws-c)

Display’value of b’ws-b

Else if(ws-c>ws-a)

Display ‘value of c’ws-c

End-if

End-if

End-if

Display’hello’

Stop run.

**NEXT SENTENCE:**

If the condition is true then NEXT SENTENCE execute the statements which are after the period(.).

**Syntax:**

If(ws-b>ws-a)

Continue

Else (ws-b>ws-c)

Display’value of b’ws-b

End-if

Display’hello’

Stop run.

**Example:**

identification division

program-id.

Data division

Working-storage section.

01 ws-a pic 9(1) value 8.

01 ws-b pic 9(1) value 9.

01 ws-C pic 9(1) value 5.

Procedure division.

If(ws-b>ws-a)

Continue

Else if(ws-b>ws-c)

Display’value of b’ws-b

Else if(ws-c>ws-a)

Display ‘value of c’ws-c

End-if

End-if

End-if

Display’hello’.

DISPLAY ‘HI’

Stop run.

**MOVE**

MOVE statement is used to copy data from one field to another field or a literal to a field.

There can be more than one receiving field but only one sending field.

There are 4 types of MOVE.

ELEMENTARY MOVE: Moving data from one elementary field to another elementary field.

IDENTIFICATION DIVISION.

PROGRAM-ID. ELEM.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 GROUP1.

02 A PIC X(5) VALUE 'AB123'.

02 B PIC A(4) VALUE 'PQRS'.

02 C PIC 9(3) VALUE 456.

01 GROUP2.

03 D PIC 9(3).

03 E PIC X(2).

03 F PIC A(3).

PROCEDURE DIVISION.

MOVE A TO D.

DISPLAY D.

MOVE B TO F.

DISPLAY F.

MOVE C TO E.

DISPLAY E.

STOP RUN.

**GROUP MOVE:** Moving data from one group to another group.

IDENTIFICATION DIVISION.

PROGRAM-ID. ELEM.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 GRP1.

02 A PIC X(5) VALUE 'AB123'.

02 B PIC A(4) VALUE 'PQRS'.

02 C PIC 9(3) VALUE 456.

01 GRP2.

03 D PIC 9(3).

03 A PIC X(2).

03 B PIC A(3).

PROCEDURE DIVISION.

MOVE GRP1 TO GRP2.

DISPLAY GRP1

DISPLAY GRP2

STOP RUN.

**CORRESPONDING MOVE:** Moving data from 1 group to another group only if field matching is there.

IDENTIFICATION DIVISION.

PROGRAM-ID. ELEM.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 GRP1.

02 A PIC X(5) VALUE 'AB123'.

02 B PIC A(4) VALUE 'PQRS'.

02 C PIC 9(3) VALUE 456.

01 GRP2.

03 D PIC 9(3).

03 A PIC X(2).

03 B PIC A(3).

PROCEDURE DIVISION.

MOVE CORRESPONDING GRP1 TO GRP2.

DISPLAY GRP1

DISPLAY D

DISPLAY GRP2

STOP RUN.

**MOVE REFERENCE MODIFICATION:**

It is used to move only a part of a field to a part of another field.

IDENTIFICATION DIVISION.

PROGRAM-ID. ELEM.

DATA DIVISION.

WORKING-STORAGE SECTION.

02 A PIC X(13) VALUE 'PRAVEEN KUMAR'.

02 B PIC X(7).

02 C PIC X(5).

PROCEDURE DIVISION.

MOVE A(1:7) TO B.

MOVE A(9:5) TO C.

DISPLAY B

DISPLAY C

STOP RUN.

**REDEFINES**

Redefines clause can be used to allow the same storage location to be referred by different data-names. Redefines is sharing of memory.

**Syntax:**

WORKING-STORAGE SECTION.

01 GRP1.

05 A PIC X(3) VALUE 'IBM'.

01 GRP2 REDEFINES GRP1.

05 B PIC X(10).

**Note:**

1). Redefines is possible for elementary item to elementary item.

2).Redefines is possible for Group item to Group item.

3). Redefines is not possible for Sub-elementary item to Sub-elementary item.

**RENAMES**

Renames Clause provides the programmer with the capability of regrouping elementary data items.

It resembles the Redefines clause, except that it can form a new grouping of data items which combines several items.

Renames clause is always initiated at level no 66.

Ex:

ID DIVISION.

PROGRAM-ID. EDIT.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 GRP1.

05 A PIC X(3) VALUE 'IBM'.

05 GRP2.

07 B PIC X(10) VALUE 'MAINFRAMES'.

66 R RENAMES A THRU B.

PROCEDURE DIVISION.

DISPLAY R

STOP RUN.

**Note:**

1). Renames is not possible for elementary item to elementary item.

2).Renames is not possible for Group item to Group item.

3). Renames is possible for Sub-elementary item to Sub-elementary item.

**PERFORM:**

It is used to reuse the statements within the program.

Perform can be coded as inline or outline perform.

|  |  |
| --- | --- |
| **INLINE**   1. Statements are written between PERFORM and END-PERFORM 2. The statements between PERFORM and END-PERFORM cannot be reused anywhere else in the program 3. No periods must be coded between PERFORM and END-PERFORM | **OUTLINE**   1. Statements are written within a paragraph   PERFORM PARA1  STOP RUN.  PARA1  STMNT1,  STMNT2   1. The statements between paragraph can be used any no. of times & anywhere in the program 2. Periods can be coded. |

There are four types of PERFORM statements

1. SIMPLE PERFORM:

|  |  |
| --- | --- |
| **INLINE**  IDENTIFICATION DIVISION.  PROGRAM-ID. INPERM.  DATA DIVISION.  WORKING-STORAGE SECTION.  01 A PIC X(10).  PROCEDURE DIVISION.  PERFORM  ACCEPT A  DISPLAY A  END-PERFORM.  STOP RUN. | **OUTLINE**  IDENTIFICATION DIVISION.  PROGRAM-ID. INPERM.  DATA DIVISION.  WORKING-STORAGE SECTION.  01 A PIC X(10).  PROCEDURE DIVISION.  PERFORM PARA-1.  PERFORM PARA-1.  STOP RUN.  PARA-1.  ACCEPT A  DISPLAY A. |

2. PERFORM TIMES:

|  |  |
| --- | --- |
| **INLINE**  IDENTIFICATION DIVISION.  PROGRAM-ID. INPERM.  DATA DIVISION.  WORKING-STORAGE SECTION.  01 A PIC X(10).  PROCEDURE DIVISION.  PERFORM 5 TIMES  ACCEPT A  DISPLAY A  END-PERFORM.  STOP RUN. | **OUTLINE**  IDENTIFICATION DIVISION.  PROGRAM-ID. INPERM.  DATA DIVISION.  WORKING-STORAGE SECTION.  01 A PIC X(10).  PROCEDURE DIVISION.  PERFORM PARA-1 2TIMES  DISPLAY A.  PERFORM PARA-1 3TIMES  STOP RUN.  PARA-1.  ACCEPT A  DISPLAY A. |

3. PERFORM UNTIL:

|  |  |
| --- | --- |
| **INLINE**  IDENTIFICATION DIVISION.  PROGRAM-ID. INPERM.  DATA DIVISION.  WORKING-STORAGE SECTION.  01 A PIC X(10).  01 I PIC 9(1) VALUE 1.  PROCEDURE DIVISION.  PERFORM UNTIL I>5  ACCEPT A  DISPLAY A  ADD +1 TO I  END-PERFORM.  STOP RUN. | **OUTLINE**  IDENTIFICATION DIVISION.  PROGRAM-ID. INPERM.  DATA DIVISION.  WORKING-STORAGE SECTION.  01 A PIC X(10).  PROCEDURE DIVISION.  PERFORM PARA-1 UNTIL A>20  ACCEPT A.  PERFORM PARA-1 UNTIL A<5  STOP RUN.  PARA-1.  ACCEPT A  DISPLAY A |

4. PERFORM VARING UNTIL:

|  |  |
| --- | --- |
| **INLINE**  IDENTIFICATION DIVISION.  PROGRAM-ID. INPERM.  DATA DIVISION.  WORKING-STORAGE SECTION.  01 A PIC X(10).  01 I PIC 9(1) VALUE 1.  PROCEDURE DIVISION.  PERFORM VARYING I FROM 1 BY 1 UNTIL I>5  ACCEPT A  DISPLAY A  END-PERFORM.  STOP RUN. | **OUTLINE**  IDENTIFICATION DIVISION.  PROGRAM-ID. INPERM.  DATA DIVISION.  WORKING-STORAGE SECTION.  01 A PIC X(10).  PROCEDURE DIVISION.  PERFORM VARING I FROM 1BY 1 UNTIL I>5  ACCEPT A  DISPLAY A |

**ARRAYS:**

Array is a collection of similar data items.

Arrays are used for much efficient data processing

Occurs clause is used to declare the array.

Each element is identified by the occurrence number.

In COBOL we can partially use upto 7-dimensional array.

We cannot declare OCCURS clause in 01 level as array is a occurrence of fields and 01 level is used to identify a field.

ARRAY LIFE CYCLE

1). Declaring the array

2). Loading data into array

3). Fetching data into array

SINGLE DIMENSIONAL ARRAY:

1). Declaring Array

01 ARRAY

03 STUD-DET OCCURS 10 TIMES

05 STUD-ID PIC X(10).

05 STUD-NAME PIC X(10).

2). Loading data into Array

WORKING-STORAGE SECTION

01 I PIC 9(2).

PROCEDURE DIVISION

PERFORM VARING I FROM 1 BY 1 UNTIL 1>10

ACCEPT STUD-ID(I).

ACCEPT STUD-NAME(I).

END PERFORM.

3). Fetching data from Array.

PERFORM VARING I FROM 1 BY 1 UNTIL I>10

DISPLAY STUD-ID(I).

DISPLAY STUD-NAME(I).

END-PERFORM.

TWO DIMENSIONAL ARRAY:

1). Declaring Array subscript

01 ARRAY

02 STUD-CLASS OCCURS 5 TIMES

03 STUD-DET OCCURS 5 TIMES

04 STUD-ID PIC X(10).

04 STUD-NAME PIC X(10).

2). Loading data into Array

WORKING-STORAGE SECTION

01 I PIC 9(2).

01 J PIC 9(2). index

PROCEDURE DIVISION

PERFORM VARING I FROM 1 BY 1 UNTIL 1>5

PERFORM VARING J FROM 1 BY 1 UNTIL 1>5

ACCEPT STUD-ID(I).

ACCEPT STUD-NAME(I).

END PERFORM.

END-PERFORM.

3). Fetching data from Array

PERFORM VARING I FROM 1 BY 1 UNTIL 1>5

PERFORM VARING I FROM 1 BY 1 UNTIL 1>5

DISPLAY STUD-ID(I).

ACCEPT STUD-NAME(I).

END PERFORM.

END-PERFORM.

|  |  |
| --- | --- |
| **SUBSCRIPT**  1). It is the occurrence no. of an array element.  2). Must be declared a separate variable in W.S.S  3). It is incremented or decremented using  arithmetic operation.  4). Access is slow. | **INDEX**  1). It is the displacement value of an array element.  2). It is declared using INDEXED BY clause.  3). Incremented or decremented using set clause.  4). Access is fast. |

**SEARCH & SEARCH ALL**

|  |  |
| --- | --- |
| **SEARCH**  1). It is a linear search  2). Array elements are searched sequentially one  after another .  3). Data need not be in sorted order  4). Search is efficient than perform.  5). Multiple When conditions can be specified.  6). Any relational operator can be used.  (=, >, <, <=, >=)  Ex: PROCEDURE DIVISION.  INITIALIZE J,AC-NO  PERFORM VARYING J FROM 1 BY 1 UNTIL J>2  ACCEPT ACCT-TAB(I)  DISPLAY ACCT-TAB(I)  END-PERFORM.  ACCEPT AC-NO  SEARCH ACCT-TAB AT END DISPLAY 'SEARCH ENDED'  WHEN ACCT-NO(I) = AC-NO  DISPLAY ACCT-NO(I)  DISPLAY NAME(I)  DISPLAY AMOUNT(I)  END-SEARCH.  STOP RUN. | **SEARCH ALL**  1). It is a binary search  2). Array is split into two halves. Last element of  1st half & 1st element of 2nd  half are compared  with the desired value. Based on the  possibility of desired value in one half the  other half is ignored. Remaining half is again  split to compare until desired value is obtained.  3). Data must be in sorted order.  4). Search All is efficient than search.  5). One When condition can be specified.  6). Only = operator can be used.  Ex: PROCEDURE DIVISION.  INITIALIZE J,AC-NO  PERFORM VARYING J FROM 1 BY 1 UNTIL J>2  ACCEPT ACCT-TAB(I)  DISPLAY ACCT-TAB(I)  END-PERFORM.  ACCEPT AC-NO  SEARCH ALL ACCT-TAB AT END DISPLAY 'SEARCH EN  WHEN ACCT-NO(I) = AC-NO  DISPLAY ACCT-NO(I)  DISPLAY NAME(I)  DISPLAY AMOUNT(I)  END-SEARCH.  STOP RUN. |

**SORT:** Incobol SORT statement used to sort the input files write into output file either in ascending or desceding order.

syntax:- sort wrkfile on assending key eno of wrrec using infile giving outfile

**merge:** Incobol MERGE statement used to sort the multiple input files write into output file either in ascending or desceding order.

syntax:- merge wrkfile on accending key eno of wrkrec using infile1

infile2 giving outfile.

**Input Procedure & Output Procedure in SORT:**

it is used to select 10,000 records from a group of 1 lakh records based on the condition.

Syntax:

sort workfile on descending/ascending key eno

input procedure is 100 pAra giving out file.

This procedure is called edit before sort.

This procedure is efficient than output procedur.

100 para

Open input infile

Perform until ws-EOF

Read infile at end set ws-EOF to true

Not at end

If place =’chennai’

Release wrkrec

End-read

End-perform.

* 01 ws-ind pic x.

88 EOF value ‘y’.

* Set ws-EOF to true

Move ‘y’ to ws-EOF.

**Input procedure :-**

Identification division.

Program-id. Seqwrt.

Environment division.

Input-output section.

File-controle.

Select infile assign to dd1.

Select outfile assign to dd2.

Select wrkfile assign to dd3.

Data division.

File section.

Fd-infile.

01 inrec.

05 eno pic 9(3).

05 filler pic x.

05 ename pic a(10).

05 filler pic (66).

Fd outfile

01 outrec.

05 eno pic 9(3).

05 filler pic x.

05 ename pic a(10).

05 filler pic (66).

Sd wrkfile.

01 wrkfile.

05 eno pic 9(3).

05 filler pic x.

05 ename pic a(10).

05 filler pic (66).

Working-storage section.

01 ws-END pic x.

88 ws-END value ‘y’.

Procedure division.

sort workfile on ascending key eno OF WRKREC

input procedure is 100 pAra giving out file.

stop run.

100-pera

open input infile

perform infile ws-eof

Read infile at end set ws-eof to true

not at end

if ename =’haderabad’

release wrkrec from inrec

end-if.

end-read.

100-para-exit

exit.

**Purpose of input procedure IN sort:-**

Input file is having no of files with different names but we want the files with the quiew name ’haderabad’ it wil display the files present with that name is the output.

INPUT PROEDURE

Copied selected files to work file

sort

then movie to output file

OUTPUT PROEDURE

all the files are copied to work file

sort

selected files are copied to output file.

**Output procedure:**

Changes:-

Fd-OUTfile.

01 OUTrec.

05 eno pic 9(3).

05 filler pic x.

05 ename pic a(10).

05 filler pic (66).

Sd WRKfile

01 WRKrec.

05 eno pic 9(3).

05 filler pic x.

05 ename pic a(10).

05 filler pic (66).

Procedure division.

sort workfile on ascending key eno2 USING INFIL

OUTput procedure is 100 pAra .

stop run.

100-pera

open OUTPUT OUTfile

perform infile ws-eof

RETURN WRKfile at end set ws-eof to true

not at end

if ename =’haderabad’

WRITE OUTrec from WRKrec

end-if.

end-read.

100-para-exit

exit.

**SUB PROGRAMS:**

A sub-program is nothing but a COBOL program which can be reused any no. of times.

**CALL** statement is used to PASS control from one program to another.

The program which passes the control is called **MAIN** or **CALLING** program

The program which receives the control is called **SUB** or **CALLED** program

Common subroutines used in real time- Date routine, Error routines, Abend routines.

|  |  |
| --- | --- |
| **MAIN-PROGRAM**  IDENTIFICATION DIVISION.  PROGRAM-ID. ADDITION.  DATA DIVISION.  WORKING-STORAGE SECTION.  01 A PIC X(3).  PROCEDURE DIVISION.  MOVE 'XYZ' TO A  DISPLAY 'MAIN' A  CALL 'SUBPGM' USING BY REFERENCE A  DISPLAY 'MAIN' A  STOP RUN.  1). It is the CALLING program  2). Control & data are passed using CALL statement  3). Actual parameters are declared in working storage  Section  4). STOP RUN is specified. | **SUB-PROGRAM**  IDENTIFICATION DIVISION.  PROGRAM-ID. ADDITION.  DATA DIVISION.  LINKAGE SECTION.  01 LS-A PIC X(3).  PROCEDURE DIVISION USING LS-A.  MOVE 'GHJ' TO LS-A  DISPLAY LS-A  GOBACK.  1). It is the CALLED program  2). Control & data are received by USING statements  3). Formal parameters are declared in linkage section.  4). EXIT Program is specified. |

CALL BY REFERENCE:

When formal parameters are changed then the actual parameters also get effected, since they are allocated to same memory.

CALL BY VALUE:

When formal parameters are changed then actual parameters don’t get effected since they are allocated to different memory.

|  |  |
| --- | --- |
| **STATIC CALL**  1). The object codes of main & sub programs are  linked to same load module.  2). Compiler option is NO-DYNAM(default)  3). In terms of processing static call is efficient.  4). In terms of load module size static call is less  efficient.  5). In the program it is specified as CALL ‘PROG’ | **DYNAMIC CALL**  1). The object codes are linked during execution.  2). Compiler option is DYNAM  3). Less efficient.  4). More efficient.  5). Move PGM TO PGM  CALL PGM |

\*\*Compiler option is specified in the PARM parameter of main program compile JCL

//STEP1 EXEC PGM=IGYCRCTL, PARM=DYNAM.

**Computational clause:**

There are below types of Computational clauses in COBOL .

1).Comp

2).Comp-1

3).Comp-2

4).Comp-3

5).Comp-4

6).Comp-5

**1).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.

Syntax: 01 ws-a pic S9(8) usage comp.

Example:

identification division

program-id.

Data division

Working-storage section.

01 ws-a pic S9(8) usage comp.

Procedure division.

Initialize ws-a movie ‘12345678’ to ws-a

Display ws-a ‘length’ length of ws-a

Stop run.

0/p:-

12345678 length 0000000004(it will take only 4 bytes)

Note:- it will use for only storage purpose.

It will take only 4 bytes .

**2).Comp-1:**

Single word floating point item. PIC Clause should not be specified.

The sign is contained in the first bit of the of the leftmost byte and the

exponent is contained in the remaining 7 bits of the first byte. The last

3 bytes contain the mantissa.

**Syntax:** 01 ws-a usage comp-1.

3).**Comp-2:**Double word floating-point item. PIC Clause should not be specified.

7 bytes are used for mantissa and hence used for high precision

calculation.

**Syntax:**01 ws-ausage comp-2

4).**Comp-3:**Packed Decimal representation. One digit takes half byte.

PIC 9 (N) comp-3 data item would require (N + 1)/2 bytes. The sign is stored separately in the rightmost half-byte regardless of whether S is specified in the PICTURE or not.

C – Signed Positive D – Signed Negative F-Unsigned Positive.

**Syntax:**

01 ws-a pic 9(2)v9(4) usage comp 3.

Formula to calculate the number of bytes: Odd number= n+1/2

Even number=n/2+1

5).**Comp-4:** binary integer data

NOTE: binary, comp, comp-4 are all the same; values in these kinds of fields may

be truncated, based on compiler options and the particular work being done.

6).**COMP-5:** binary integer, but never truncated

**Copy book:**

It is used to store the predefined record structure. basically used to call the sub routines, Abend modules …..etc. After compilation the information present in copybook will be copied to main program.

Syntax: COPY membername

Example:

IDENTIFICATION DIVISION.

PROGRAM-ID. COPY1

ENVIRONMENT DIVISION.

COPY WORKCOPY

DATA DIVISION.

DISPLAY A.

DISPLAY B.

STOP RUN.

Copybook contains below information.

WORKCOPY:

WORKING-STORAGE SECTION.

01 A PIC X(5) VALUE ‘MANOJ’.

01 B PIC X(12) VALUE ‘IONE TECHNOLOGIES’.

**Character handling.**

**Type -1:** Converting old string to new string.

**Syntax:**

01 Ws-A pic x(12) value ‘dileep kumar’

Inspect ws-A converting ‘dileep’ to ‘senthel’

Display’ ‘ws-A

o/p: senthal kumar

it will change character by character.

**Type-2:** Replacing one Character with another Character.

**Replace:-**

**Syntax:**

Working-storage section.

77ws-A pic x(12) value ‘dileep kumar’

Procedure division.

Inspect ws-A replacing all ‘d’ by ‘r’

Display ‘ ‘ws-A.

o/p:- rileep kumar

**Type-3:** Counting the total number of Characters in a string.

**Syntax:**

inspect tallying ws-B for all ‘A’

**Type-4:** Replacing leading String with another String.

**Syntax:**

Ws-A =’dileep kumar’

Inspect ws-A replacingleading ‘dileep’ by ‘sendil’.

o/p:- sendil kumar

**Type-5** : Replacing leading Character with another Character.

**Syntax:**

ws-A replacingleading ‘0’ by ‘@’.

**Type-6:** Replacing ALL Characters with another Character based on BEFORE & AFTER.

**Syntax:**

Ws-A =’malayalam’

Inspect ws-A replacing all A by ‘0’ before ‘l’ after ‘m’

**String delimiter :-**

**Type-7:** Splitting a string into multiple strings using DELIMITED BY SPACE:

**Syntax:**

Ws-A=”dileep kumar”

Unstring ws-A delimited by space into ws-B,ws-C

o/p:- ws-b=dileep

ws-c=kumar

**FILES:**

Data is a collection of strings of characters which doesn’t have any meaning. Ex: pqr

Information which is also a collection of strings of characters which has a meaning. Ex: Anil.

Data on the mainframes is stored in the form of datasets.

Dataset is permanent memory space in the system memory.

Datasets are of two types.

**1). PDS** (PARTITIONED DATASET):

The PDS memory is divided into partitions called members. In these members we write the programs

Jobs, Copybooks etc. Every member should have some properties, those properties are stored in the

directory block.

**DIRECTORY BLOCK** (DB)**:**

Directory blocks stores members’ information like member name, creation date, who created it, size

of the member etc. Each directory block can hold 6 members information.

For PDS directory block size must be always greater than zero. DB>0

**2).PS** (PHYSICAL SEQUENTIAL DATASET):

The PS memory has no partitions and data is stored in the continuous

For PS DB is always zero. DB=0

A PS is usually referred as FILE in the program.

A file is a collection of records.

A record is a collection of fields.

A field is collection of characters.

One line of information in the field is treated as a record , which is identified in the using 01 level no.

USING FILES IN COBOL

1). Declaring the files.

2). Declaring the record structure of the files.

3). Opening the file

4). Process the file data

5). Closing the file

1). Declaring the files:

Files are declared in FILE-CONTROL paragraph under INPUT-OUTPUT section of

ENVIRONMENT division.

Select statement is used to declare the files that are used by the particular program.

To declare N no. of files we need N no. of select statements.

The logical file, physical hardware device names where the actual data is stored is linked using

ASSIGN clause. File name used in program is called logical file. It is just for program purpose &

doesn’t hold any data

Ex:

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

**FILE-CONTROL.**

**SELECT INFILE ASSIGN TO DD1.**

**SELECT OUTFILE ASSIGN TO DD2.**

2). Declaring the record structure of the files:

The record structure specified under FD of each file is to allocate memory for the file records in the

buffer.

Ex:

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT INFILE ASSIGN TO DD1.

SELECT OUTFILE ASSIGN TO DD2.

DATA DIVISION.

FILE SECTION.

**FD INFILE.**

**01 IN-REC PIC X(80).**

**FD OUTFILE.**

**01 OUT-REC PIC X(80).**

3). Opening the file:

Open statement is used to make the files available for the program to access the data.

Once the file is made available we can do operations on files by mentioning different modes.

**INPUT:** It allows only read access on file. Any no. of users can use the file when opened in INPUT mode

**OUTPUT:** It allows to write new records into the file. Only one user can open file in this mode.

**I-O:** It allows to read/modify existing data and write new data into a file. Only one user can access at a time

\*\*If write operation is used on existing file which is opened in OUT-PUT/I-O mode then new data written

overrides the existing data.

\*\*If write operation is used on existing file which is opened in EXTEND mode then new record will be

appended or added to existing data.

Ex:

PROCEDURE DIVISION.

**OPEN INPUT INFILE**

**OPEN OUTPUT OUTFILE**

PERFORM UNTIL EOF='Y'

READ INFILE

4). Process the file data:

After accessing the files we do operations on the files.

Ex:

PROCEDURE DIVISION.

OPEN INPUT INFILE

OPEN OUTPUT OUTFILE

PERFORM UNTIL EOF='Y'

READ INFILE

AT END

MOVE 'Y' TO EOF

NOT AT END

MOVE IN-REC TO OUT-REC

WRITE OUT-REC

END-READ

END-PERFORM.

5). Closing the file:

After doing all the operations we need to close the file in order to make it available for other users.

Ex:

PROCEDURE DIVISION.

OPEN INPUT INFILE

OPEN OUTPUT OUTFILE

PERFORM UNTIL EOF='Y'

READ INFILE

AT END

MOVE 'Y' TO EOF

NOT AT END

MOVE IN-REC TO OUT-REC

WRITE OUT-REC

END-READ

END-PERFORM.

**CLOSE INFILE**

**CLOSE OUTFILE**

STOP RUN.

**VSAM**

**Introduction about VSAM files:**

* VSAM stands for Virtual Storage Access Method
* VSAM is a data management system introduced by IBM in the 1970s
* Although there are still datasets that are best managed with the several other (non-VSAM) data management methods, VSAM is a major component of modern IBM operating systems.
* The management of data takes place as records in VSAM system and they are allowed to be of any length.
* VSAM supports fixed as well as variable length records.
* These records are placed in blocks which are termed as Control Intervals.
* Control Intervals are further placed as Control Areas which are still larger in size.

**Advantages of VSAM:**

* Records can be accessed sequentially or randomly or dynamically.
* Supports Batch and Online.
* Deletion of records results in them being physically removed from DASD.
* AMS commands can be executed like TSO Commands.
* VSAM files are shared across the Regions and systems.
* Independent to Storage device types.
* VSAM datasets are expanded across the volumes.
* Accessing the data is faster and easier.

**Disadvantages of VSAM:**

* VSAM Datasets require more storage space compared to other type of  
  datasets due to Control Information (CI) present in them.
* VSAM can’t be stored in TAPE Volume i.e., VSAM Stored only in DASD.
* We can’t browse, view, edit the VSAM datasets in ISPF.  (But we can delete the VSAM Datasets in ISPF).
* VSAM is not a database like DB2, IMS DB & IDMS.

**Types of VSAM files:**

Below are the types of VSAM files.

1).KSDS (Key sequenced data set).

2).ESDS (Entry sequenced data set).

3).RRDS (Relative record data set).

4).LDS (Linear data set).

**1). Key Sequenced Data Set:**

* In this method each record is located and accessed by specifying its key value which is a unique sequence of characters for each record at a fixed position.
* The KSDS cluster has two components in it
  + Index component
  + Data component
* The index component of KSDS cluster contains the list of key values for the records in the cluster with pointers to the corresponding records in the data component.
* The records in KSDS may be accessed sequentially, Random, Dynamic.
* The records of KSDS cluster may be of fixed length or variable length.
* Records may be added or deleted at any point.

**2). Entry sequenced data set:**

* The records in ESDS cluster are stored in the order in which they are entered into the dataset.
* Each record is referenced by its Relative Byte Address(RBA).
* In ESDS dataset of 100 byte records, the RBA of the first record is 0, the RBA of the second record is 100, the RBA of third record is 200…etc.
* ESDS cluster has one component i.e data component.
* The records in ESDS may be accessed sequentially, in order by RBA value, or directly, by supplying the RBA of the desired record.
* The record of ESDS cluster may be fixed length or variable length.
* Records cannot be deleted from ESDS cluster. They may only ba added(appended) to the end of the dataset.

**3). Relative record data set:**

* The records in RRDS cluster are stored in fixed length slots
* Each record is referenced by the number of its slot, which is a

number varying from 1 to the maximum number of records in thedataset

* The records in RRDS cluster may be accessed sequentially, inrelative record number order, or directly, by supplying the relativerecord number of the desired record
* The records of RRDS cluster must be of **fixed length**
* Records may be added to RRDS cluster by writing a new recordsinto an empty slot, and records may be deleted from an RRDScluster, thereby leaving an empty slot where the record that wasdeleted was previously stored.

**Creation of KSDS:**

//FSS045AN JOB NOTIFY=&SYSUID

//\*\*\*\* CREATION OF KSDS \*\*\*//

//S1 EXEC PGM=IDCAMS

//SYSPRINT DD SYSOUT=\*

//SYSIN DD \*

DEFINE CLUSTER(NAME(FSS045.RAVINDRA.KSDS)-

VOLUMES(FSSV04)-

CISZ(4096)-

RECORDSIZE(80,80)-

FREESPACE(10,10)-

KEYS(4,0)-

TRK(10,10)-

INDEXED)

/\*

**Creation of ESDS:**

//FSS045AN JOB NOTIFY=&SYSUID

//\*\*\*\* CREATION OF ESDS \*\*\*//

//S1 EXEC PGM=IDCAMS

//SYSPRINT DD SYSOUT=\*

//SYSIN DD \*

DEFINE CLUSTER(NAME(FSS045.RAVINDRA.ESDS1)-

VOLUMES(FSSV04)-

CISZ(4096)-

RECORDSIZE(80,80)-

TRK(10,10)-

NONINDEXED)

/\*

**Creation of RRDS:**

//FSS045AN JOB NOTIFY=&SYSUID

//\*\*\*\* CREATION OF RRDS \*\*\*//

//S1 EXEC PGM=IDCAMS

//SYSPRINT DD SYSOUT=\*

//SYSIN DD \*

DEFINE CLUSTER(NAME(FSS045.RAVINDRA.RRDS)-

VOLUMES(FSSV04)-

CISZ(4096)-

RECORDSIZE(80,80)-

TRK(10,10)-

NUMBERED)

/\*

**AMS Commands:**

AMS commands fall into two categories

1). Functional commands : to specify a task

eg : DEFINE, ALTER, DELETE, LISTCAT, REPRO, PRINT, EXPORT/IMPORT.

2). Modal commands : to control execution of functional commands.

eg : IF, SET etc.

DEFINE: Used to create VSAM files.

ALTER: Used to modify the VSAM file attributes.

E.g: //sysindd \*

alter(aggt.guntur2.ksds)-

volumes(mtptemp)

/\*

DELETE: Used to delete VSAM files.

E.g: //sysindd \*

delete(aggt.guntur2.ksds)

/\*

LISTCAT: It list CATALOG information.

E.g: //sysindd \*

LISTCAT ENTRIES(aggt.guntur2.ksds) all

/\*

REPRO: Used to copying the data from Sequential file to VSAM file and vice versa.

E.g: //sysindd \*

REPRO IDS(input data set)-

ODS(output data set)

/\*

PRINT: It list the data set information.

E.g: //sysindd \*

print IDS(aggt.guntur2.ksds) char

/\*

//sysindd \*

print IDS(aggt.guntur2.ksds) HEX

/\*

EXPORT/IMPORT: used for Loading/Unloading the VSAM Data sets.

Taking the backup for VSAM files.

E.g:

//SYSIN DD \*

EXPORT IDS(aggt.guntur2.ksds)-

ODS(ps file)

/\*

Here KSDS file is exported to ps file & KSDS is deleted.

The data presented in PS file is exported to ESDS or KSDS.

E.g:

//SYSIN DD \*

IMPORT IDS(aggt.guntur2.ksds)-

ODS(ps file)

/\*

**Sample Program of AMS Commands:**

//job card

//s1 exec pgm=idcams

//sysprintddsysout=\*

//sysindd \*

AMS COMMANDS

/\*

**Copying data from PS file to VSAM file:**

//job card

//s1 exec pgm=idcams

//sysprintddsysout=\*

//sysindd \*

REPRO IDS(input data set)-

ODS(output data set)

/\*

**Note: If the input data set is PS file then it will get load into VSAM.**

**Explanation about CA &CI :**

1). A control interval contains records,control information, and(in the case of KSDS clusters) possibly free space which may later be used to contain inserted records.

2). Control Interval in VSAM is a unit of data that is transferred when an I/O request is made between auxiliary storage and virtual storage.

3). CI concepts is similar to blocking for Non-VSAM files.

4). When a VSAM dataset is loaded, control intervals are created and records are written into them.

5). With KSDS clusters, the entire control interval isusually not filled. Some percentage of free space is left available for expansion.

6). With ESDS clusters, each control interval is completely filled before records are written into the next control interval in sequence.

7). With RRDS clusters, Control intervals are filled with fixed-length slots, each containing either an active record or a dummy record. Slots containing dummy records are available for use when new records are added to the dataset.

8). A group of control intervals makes up a control area.

9). For ESDS and RRDS clusters, control areas are filled with control intervals that contain records.

10). For KSDS clusters, some of the control intervals in each control area may consist entirely of free space that can be used for dataset expansion.

11). The number of control intervals per control area depends on how much space is reserved when the dataset is created.

**Usage of VSAM files in COBOL:**

Syntax to use VSAM files in COBOL:

FILE-CONTROL.

SELECT filnam1 ASSIGN TO external-reference.

SEQUENTIAL

ORGANIZATION IS INDEXED

RELATIVE

SEQUENTIAL

ACCESS MODE ISRANDOM

DYNAMIC

RECORD KEY IS data-name-1.

RELATIVE KEY IS dataname-2.

Notes:

🡪ORGANIZATION is SEQUENTIAL : The file is sequential file

🡪ORGANIZATION is INDEXED: The position of each logical record in the file is determined by the indexes created with the file and maintained by the system

🡪ORGANIZATION is RELATIVE : The position of each logical record in the file is determined by its relative record number.

🡪ACCESS MODE is SEQUENTIAL: Records in the file are accessed in the sequence of ascending record key values within the currently used key reference.

🡪ACCESS MODE is RANDOM: The value placed in a record key data item specifies the record to be accessed.

🡪ACCESS MODE is DYNAMIC: The records in the file can be accessed sequentially or randomly, depending on the form of specific input/output request.

🡪RECORD KEY Clause: Specifies the data item within the record that is the prime record key for an indexed file. The values contained in the RECORD KEY data item must be unique among all records in the file.Data-name-1 is the prime RECORD KEY data item. It must be defined within a record description entry associated with the file.

🡪ALTERNATE RECORD KEY Clause: Specifies the data item within the record that provides an alternative path to the data in an indexed file. Data-name-3 is the ALTERNATE RECORD KEY data item If the DUPLICATES option is specified, the values contained in the ALTERNATE RECORD KEY data item may be duplicated within any record in the file. Otherwise key values should be unique.

🡪RELATIVE KEY Clause Specifies the RELATIVE RECORD NUMBER for a specific logical record within a relative file.

🡪 Data-name-2 is the RELATIVE KEY data item. It must be

defined as an unsigned integer data item and must NOT be

defined in a record description entry associated with this file.

🡪 The Relative key is not part of record.

**Defaults:**

When the ORGANIZATION Clause is omitted, SEQUENTIAL ORGANIZATION is Assumed.

When ACCESS MODE Clause is omitted, SEQUENTIAL ACCESS MODE is assumed.

**Security &Relaibility in VSAM:**

SHARE OPTS=(A,B)

A 🡪 Across the region

B 🡪 Across the system

A:

1 🡪 Single read & single write

2 🡪 Multiple read & single write

3 🡪 Multiple read & Multiple write

4 🡪Same as 3 but it refreshes the buffer at every random acess.

B:

1 🡪 Single read & single write

2 🡪 Multiple read & single write

3 🡪 Multiple read & Multiple write

4 🡪Same as 3 but it refreshes the buffer at every random acess.

To omit the share option use (1,3)

Default Share option=(1,3)

**DB2(Data Base 2)**

Data base is a collection of data, advantages of database over files are;

* Large volumes of data can be stored in database whereas in files we can store lower volumes of data.
* Accessing data stored in files requires lot of program code, whereas accessing database is simple using SQL queries.
* Database supports high data security and data concatenation.

Database

E.no contact

E001 XXXXXX

E002 XXXXXX

E003 XXXXXX

File

E.no contact

E001 9885------

|  |  |
| --- | --- |
| In case of file while updating the records we will open the file in ‘COBOL, JCL’ old mode. Then nobody can access this file until it has been closed. | Whereas in case of D.B we can lock the record when updating table |

RDBMS (Relational database management system):

RDBMS supports accessing data from different entities by building a relationship between entities.

Address

Project (Emp-TB2)

SAL (pay roll)

* DB2 is developed by IBM in 1970’s
* When DB2sofware is installed on the mainframe it creates a database environment which has two components
* System space
* User space
* In the system space we have the DB2 catalog tables and DB2 directory tables.
* DB2 catalog tables contains the information about the user defined database and its objects.
* DB2 directory tables contain the information about the database objects and their storage location.
* How the tables are there structure is stored in DB2 catalog.
* Information about the tables, table spaces, views is stored in DB2 directory tables.
* In the user space the user defined database objects and their data is stored.
* When we are updating the data then it will update in the user space.

**System space**

|  |
| --- |
| **DB2 catalog tables**  Structure of the tables |
| **DB2 directory block**  Information of the objects |

**User space**

Tables

Table spaces objects

views

“Database” 🡪software creates the environment

Columns

Rows

View 2

View 1

Table 2

Table 1

Table space4

Table space2

Table space1

Table space5

Table space3

Data base

**System memory**

|  |
| --- |
| Volume  V121 |
| Volume  V122 |
| Volume  V123 |

|  |
| --- |
| Volume  V124 |
| Volume  V125 |

**Table space**:

Storage group1

Storage group2

Amount of space required to store the tables, views (simply

objects). This is also we will define once we create volume

sequels like V121, V122,V123… like that.

**Table:**

It is the entry where we store the data in the form of rows

and columns.

**Storage group:**

It is set of memory volumes allocated to a table space.

* Data in table spaces are stored in the form of pages.
* Page is an amount of space in table space which used to transit data from system memory to buffer.
* Page size can be 4k, 8k, 12k, 16k, ---------32k, (multiples)
* Page can have maximum of ‘127’ rows.

When using files In case of tables

**File Table**

Page1 Page2

Record blocks1 record blocks2

**Record blocks .**

**Table spaces are of three types;**

1. Simple table space
2. Segmented table space
3. Partitioned table space
4. **Simple table space**: if in a page of table space more than one tables data is stored then it is simple T.S

Simple table space

----pg1------ -------pg2------------pg3------

T1------------- --------------------------------

T2------------- --------------------------------

1. **Segmented table space:**

if any single table’s data is stored in a page then it is segmented T.S

|  |  |  |  |
| --- | --- | --- | --- |
| Table1 data  ------------  ------------ | Table2 data  ------------  ------------ | Table3 data  ------------  ------------ |  |

1. **Partitioned table space:**

If a particular group of data is stored in a set of pages called partition then it is called partitioned table space.

Ps1(N.I) Ps2(S.I)

---------- ------------- ------------ -------- ------------------ ----- -- - -- - - - -- - - -

----------- - - -- - - -- - - - - - -- - -- - - -- - - -- - - - -- - - - -- - - -- - -- - - - -- - -

Ps3(E.I) Ps4(W.I)

* Using partitioned space the amount of memory scan to be get reduced there by improve the DB2 performance.

DB2 data is accessed using SQL queries

**Types of SQL Queries:**

DDL (data definition language)

DML (data manipulation language)

DCL (data control language)

1. **DDL**

Create

Alter

Drop

1. **DML**

Insert

Update

Delete

Select

1. **DCL**

Grant

Revoke

**CREATE:** It is used to create data base objects.

**🡪Creating a data base:**

CREATE DATABASE DB123

**🡪Creating a table:**

CREATE TABLE EMP\_TBL

(EMP\_ID INT NOT NULL,

EMP\_NAME CHR(30),

EMP\_DEPT SMALL INT,

EMP\_SAL DECIMAL(11,2),

EMP\_ADDR VARCHAR(50),

EMP\_JOIN\_DT DATE NOT NULL WITH DEFA

EMP\_JOIN\_TM TIME,

TRANS\_TS TIMESTAMP.

PRIMARY KEY(EMP\_ID),

ON DELETE CASCADE)IN OZAGEN DB.0ZAGEN

**Data types DB2:**

SMALL INT 2BYTES S9(4) COMP

INT 4BYTES S9(9) COMP

CHAR(n) n BYTES X(n)

VARCHAR(n) (n+2)BYTES

01 VAR\_FIELD

49 LEN PIC S9(4)

49 TEXT PIC X(n)

DECIMAL ((M+1)/2)/(m/2)+1 S9(M-n)V9(N) COMP

DATE 10 BYTES X(10) [DD-MM-YYYY]

TIME 8 BYTES X(8) [HH-MN-SS]

TIME STAMP 26 BYTES X(26)[DD.MM.YY-HH.MN.SS.NNNNNNNN]

DIFFERENCES BETWEEN CHR AND VARCHAR

|  |  |
| --- | --- |
| **CHAR**  Access is fast since it has only the below 2 processes.  1.allocates the full memory  2.then the data is inserted into the field.  There is a memory wastage. | **VARCHAR**  Access is slow, since it has the below three processes  1.the count of character is stored in the length F1 first.  2.based on the count the many bytes are allocated.  3.then the data is inserted into the columns.  There is no memory wastage. |

**Primary key:**

* It is used to uniquely identify a row in the table.
* When primary key constraint set on a column it doesn’t allow duplicate or null values.

**Null values:**

It is a unknown value stored in a column when no value is specified.

**Constraint:**

Mechanism to control the data.

1. **Null constraint:** It is buy default any column in a table without any constraint is nullable. (allows null values) nullable.

**2. not null constraint: I**t does not allow null values to be stored and so a value must be specified**.**

**Not null with default:** If no value is a specified then instead of null value it stores a default value based on data type.

Note:

1. For SMALL INT

INT and 0’S are the default values

DECIMAL

1. FOR CHAR 🡪 SPACE
2. FOR DATE, TIME AND TIMESTAMP

DATE 🡪 CURRENT DATE

TIME 🡪 CURENT TIME

TIMESTAMP 🡪 CURRENT TIMESTAMP are default values.

**Alter:** It is used to modify the database objects.

Syntax:

ALTER EMP\_TBL ADD EMP\_DOB DATE

* This column will adds at the last in the table ‘EMP\_TBL’

Note:

* A column can be added only as the last column in the table, if we need to add the column in between the columns then we need to DROP and CREATE new table.

EMP-ID EMP-DOB

-------------------------------------

-------------------------------------

EMP.ID EMP.NAME

-------------------------------------

-------------------------------------

* we can change the data types and data lengths.

ALTER EMP\_TBL ALTER EMP\_SAL DECIMAL(11,2)DECIMAL

* we can increase the data length if a table is empty, but if the table is non empty we can only increase the (data length).

DROP:

It is used to permanently delete a data object in a database.

Syntax:

\*DROP TABLE EMP\_TBL

\*DROP VIEW V\_EMP\_TBL

DCL: data control language

* it issues permissions on DML commands permissions.

1. GRANT:

SYNTAX: GRANT INSERT, SELECT ON EMP\_TBL TO 0ZA046,

GRP1,

GRANT ALL ON EMP\_TBL TO 0ZA047,0ZA04

1. REVOKE:

SYNTAX: REVOKE INSERT ON EMP\_TBL

FROM 0ZA046

REVOKE ALL ON EMP\_TBL

FROM GRP1

**DML(DATA MANIPULATION LANGUAGE)**

1. INSERT: It is used to insert a new row into the table only one row at a time.

Syntax: INSERT INTO EMP\_TBL

VALUES(0314,’RAMESH’,-------------------)

Note: If all values are specified, then column names need not be specified.

Syntax: INSERT INTO EMP\_TBL

(EMP\_ID, EMP\_NAME)

VALUES(0314, ‘RAMESH’)

1. UPDATE:

It is used to modify the existing data into the table.

Syntax: UPDATE EMP\_TBL

SET EMP\_SAL=EMP\_SAL+1000

* This will update all EMP\_SAL column values in the table.

UPDATE EMP\_TBL

SET EMP\_SAL=EMP\_SAL+1000

WHERE DEPTNO=’01’

“WHERE”: Where clause is used to specify a condition based on which rows are selected.

1. DELETE: It DELETES the rows from the table.

SYNTAX: DELETE FROM EMP\_TBL

It deletes all rows

DELETE FROM EMP\_TBL

WHERE DEPT=’D1’

1. SELECT: It is used to retrieve rows from the table.

SYNTAX:1) SELECT \* FROM EMP\_TBL

ALL COLUMNS

This query retrieves all rows and all columns.

i) SELECT EMP\_ID, EMP\_NAME FROM EMP\_TBL

This query retrieves all rows with columns EMP\_ID, EMP\_NAME.

ii) SELECT \* FROM EMP\_TBL

WHERE DEPT=’D1’

This will retrieve all rows with DEPT value ‘D1’

iii) SELECT EMP\_ID FROM EMP\_TBL

WHERE EMP\_SAL BETWEEN 10000 AND 20000

Note: BETWEEN is used to specify a range of values.

The above query retrieves only those contain

‘NOT BETWEEN’:

* SELECT EMP\_ID FROM EMP\_TBL

WHERE EMP\_SAL NOT BETWEEN 10000AND 20000

This query retrieves those rows with ‘EMP\_SAL’ column values other than the range 10000 till 20000.

IN: (OR) It will acts as ‘OR’

* SELECT EMP\_ID FROM EMP\_TBL

WHERE DEPT IN(‘D1’,’D2’)

IN works as OR and the above query retrieves those rows which have depart value D1 or D2.

NOT IN:

* SELECT EMP\_ID FROM EMP\_TBL

WHERE DEPT NOT IN(‘D1’,’D2’)

It will retrieves those rows which are not having depory values D1 or D2.

DISTINCT: It is used to retrieve only unique values from the column values..

* SELECT DISTINCT

ORDER BY:

It is used to retrieve the rows in sorted order

* SELECT DISTINCT DEPT FROM EMP\_TBL

ORDER BY DEPT ASC/DESC

* When a select query executes the rows are retrieved from base table into a resultant table in the buffer.

Whatever modifications we do to the data are first done in the resultant table and then in the base table.

GROUP BY: It is used to group a set of similar column values.

Aggregate functions:

1. MAX: It gives the maximum column value

SELECT MAX (EMP-SAL) FROM EMP\_TBL

1. MIN: Itgives the minimum column values.

SELECT MIN (EMP\_SAL) FROM EMP\_TBL

1. SUM: It gives the sum of column values.

SELECT SUM (EMP\_SAL) FROM EMP\_TBL

1. AVG: It gives the average value from a set of
2. COUNT: It gives the count of rows or count of column values.

* SELECT COUNT(\*)

It will gives the roof rows.

* SELECT COUNT (EMP\_SAL) FROM EMP\_TBL

It counts no. of column values excluding null values.

Note: SELECT COUNT (DISTINCT DEPT)

GROUP BY:

D1

D2

D3

D1

D1

D2

D3

D3

D1

D1

D1

EX: SELECT DEPT COUNT (\*) FROM EMP\_TBL

GROUP BY DEPT

Having result

D2

D2

D2

D1 4

D2 3

D3 2

D3

D3

Ex2: SELECT CLASS, SECTION, COUNT \* FROM

GROUP BY CLASS, SECTION

1 A

1 A

1 B

2 A

2 B

2 A

1 A

1 B

2 A

1 A

1 A

1 B

1 A

1 B

1 A

1 A

1 B

COUNT:

2 A

2 B

2 A

2 B

2 B

2 A

2 A

2 A

1 B

1 B

1 A 3

1 B 2

2 A 3

2 B 1

Note: we should not use the combination of aggregate functions and column values without “GROUP BY”.

HAVING: It is similar WHERE clause in specifying a condition. But having works on a group of column values.

* SELECT CLASS, SECTION, COUNT(\*) FROM

GROUP BY CLASS, SECTION

HAVING COUNT (\*)=1

AND WHERE WORKS on a single row.

* Having must immediately follow group by

**Sample SQL Queries:**

1.Select \* from empanj order by empid desc;

2. Select \* from empanj order by empsal asc:

3. select empsal+500 from empanj where empdept=’e002’;

4. select empsal-500 from empanj where empdept like ‘m%’

5.select empdept,sum(empsal) from empanj group by empdept;

6. select avg (year(current\_date)-year (empdob)) fromempanj group by empdept;

7. select days (empdob) from empanj where empdept=’d005’

8. select \* from empang where elname like ‘r’;

9. selecherem empanj wt sum (empsal) from empanj;

10. select avg (empsal) from empanj;

11. select avg (empsal) from empanj where empdept=’m001’

12.select count (\*) from empanj where empdept=’d0001’

13.select empssal from empanj where fl name like ‘p%’ or flname like ‘%r’;

14. delete from empanj where fname=’priya’;

15.select qty from sp where ‘london’=(select city from s where s.s#=sp.s#);

16. select distinct p# from sp;

17. select p#,pname,color,weight,city from p where weight between 16 and 19;

18.select p#.pname ,colors,weight,city from p where weight>=16 and weight<=19;

19. select p#,pname,color,weight,city from p where weight in(12,16,17);

20. select p#,pname,color,weight,city from p where weight=12 or weight =16 or weight =17;

21.select lastname,deptname from emp inner join dept on empno on empno=mgrno;

22. select lastname,deptname from emp right outer join dept on empno on empno=mgrno;

23. select lastname,deptname from emp left outer join dept on empno on empno=mgrno;

24.select sname from s where s# in (select s# from sp where p#=’p2’);or (correlated subquery as)

25.select sname from s where ‘p2; in (select p# from sp where sp s#=s.s);

26.select sname from s where s# in (select s# from sp where p# in (select p# from p where color=’red’));

27.select s# from s where status<(select max(status) from s);

28. select s# ,status ,city from ssx where status>=(select avg(status) from s sy where sy.city=sx.city);

29.select count\* from sp;

30. select count (s#) from sp;

31. select count (distinct s#) from sp;

32.select p#,sum(qty) from sp group by p#;;

33. select p#,sum(qty), max(qty) from sp where s#<> ‘s’ group by p#;

34. select p# from sp group by p# having count(\*)>;

35.select p# from p where weight>16 union all

Select p# from sp where s#=’s2’;

36. select p# from p where weight>16 union all

Select p# from sp where s#=’s2’;

37. create view empview(empid,empname,elname,empdob,empsal,empdept,as select \* from empanj

where empdept like’d’;

38. select efname,elname,deptdesc from empnew,deptmast where empdept=dept;

39. select \* from empnew where empdept=’d001’;

40. select \* from deptmast where deptno not like ‘m’;

41.select empid,efname,elname,empdob,empsal,empdept,deptdesc from empnew,deptmast where

(empdept=deptno) and (empnew.empid not like ‘c’);

42.select avg (empsal) from empnew where empdept=’e0001’;

43.select ‘pf of’,efname,’is’, empsal\*12/100 from emp new;

44.select year(curent\_date)-year(empdob) from empnew;

45.select avg(year(current\_date)-year(empdob)) from empnew;

46.select empid ,emp name ,elname,empdob,empsal,empdept,year(current\_date)-year(empdob)

Year(current\_date)-year(empdob)=(select max(year(current\_date)-year(empdob)) from empnew);

47.select empid,efname,elname ,empdob,empsal,empdept,deptdesk fromempnew ,deptmast where

(empdept=deptno) and (deptmast.deptdesc=’finance’);

48.select avg (empsal) as salary\_average from empnew,deptmast,where (empdept=deptno) and (deptmast.deptdesk=’it’);

**SQLCA (structured query language communication area)**

* It is the communication area between COBOL and DB2, it is used to know the status of a DB2 statement.
* It is used in the programming as

EXEC SQL

INCLUDE SQLCA

END\_EXEC

* This is resolved during pre-compilation and the below structure is copied.

01 SQLCA

03 SQLAID PIC EX

03 SQLCODE PIC

03 SQLSTATE PIC

03 SQLERR PIC

03 SQLWARN PIC

* If SQLCODE=0

Successful execution & row found.

SQLCODE=+100

Successful execution & row not found.

SQLCODE=-VE

Unsuccessful execution.

COPY:-

* It is used to copy a predefined text during compilation
* Copy book is a PDS member in which we store a predefined text.

Query Management fecility(QMF)

Copy book are used for reusability of code

FD FILE1

COPY CPY1

INCLUDE:-

Include is similar to copy statements in resolving a predefined text during pre-compilation.

SPUFI QMF

1. Multiple queries can be executed at a time 1.Only a single queries is executed
2. The query and the output can be saved in a 2.can not be saved

dataset

1. We can limit the number of rows in the o/p 3.we can not limit the rows.

DCLGEN:-

It is the declaration generater a DB2 tool which generates DB2 equivalent COBOL variables called host variables.

* DCLGEN member is used in the program as

EXEC SQL

INCLUDE DECEMPPP

END=EXEC

🡪 Write program taking employee OT file as input , check if the ‘emppp’ already exits in the and if so, add the OT amount from file into the EMPPP table . If the employee does not exist insert it as the new row in the table.

CODING:

ID DIVISION.

PROGRAM-ID DB2-PGM2.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE CONTROL.

SECTION EMP-OT-FILE ASSING TO DISK-1

DATA DIVISION.

FILE SECTION.

FD EMP-OTOFILE.

01 EMP-OT-REC.

03 EMP-ID PIC 9(9).

03 EMP-NAME PIC X(30).

03 EMP-OT-ATM PIC S9(9)V9(2).

03 FILLER PIC X(50).

WORKING – STORAGE SECTION.

01 WE-EOF PIC X(1) VALUE

EXEC SQL.

INCLUDE SQLCA

END-EXEC.

EXEC SQL

INCLUDE DCLEMPPP

END-EXEC.

PROCEDURE DIVISION.

0000-MAIN-PARA

PERFORM 1000-INITIALISE-PARA

PERFORM 2000-READ-PARA

PERFORM 3000-PROCESS-PARA UNTIL WE-EOF=Y

PERFORM 4000-CLOSE-PARA

1000-INITIALISE-PARA

OPEN INPUT EMP-OT-FILE

2000-READ-PARA

READ EMP-OT-FILE

AT END MOVE ‘Y’ TO WE-EOF

3000-PROCESS-PARA

MOVE EMP-ID TO HV-EMP-IP.

EXEC SQL

SELECT EMP-SAL INTO : HV=EMP-SAL

FROM EMPPP

WHERE EMP-ID=:HV-EMP-ID

END-EXEC

EVALUATE SQLCODE

WHEN O

PERFORM 3200-UPDATE-PARA

WHEN 100

PERFORM 3400-INSERT-PARA

END-EVALUATE

PERFORM 2000-READ-PARA

3200-UPDATE-PARA

ADD EMP-OT-ATM TO HVEMP-SAL

EXEC SQL

UPDATE EMPP

SET EMP-SAL= :HV-EMP-SAL

WHERE EMP-ID OT HV-EMP-ID.

END-EXEC.

3400-INSERT-PARA

MOVE EMP-ID TO HV-EMP-ID.

MOVE EMP-NAME TO HV-EMP-ID.

MOVE EMP-OT-AMT TO HV-EMP-ID.

EXEC-SQL

INSERT INTO EMPPP(EMP-ID,EMP-NAME,EMP-SAL)

VALUES(:HV-EMP-ID,:HV-EMP-NAME, :HV-EMP-SAL)

END-EXEC.

9000-CLOSE-PARA.

CLOSE EMP-OT-FILE.

Compilation of DB2 program involves

1). Pre-compilation

2). Bind

3). Execution

1.Pre-compilaton:-

The pre-compilation utility is “DSNHPC’

1. It is the process of separating COBOL and DB2 statements generating modified SOURCE CODE and DBRM.
2. Te checks the syntax errors of DB2 statements and plans error free DB2 statements in DB2
3. It also places a “timestamp” token in modify source code and DBRM.

Modified souce code:-

It contains only COBOL statements with all the DB2 statements replaced by COBOL CALL

Statements.

\*EXEC SQL

\* SELECT ---

\*END-EXEC.

CALL ‘DSNHLI’ USING PTR1

2.BIND:-

The BIND utility is “IKJEFT01”.

1. Bind takes DBRM as input and checks the syntax errors of DB2 statemente. It also check whether the column specified table by comparing with that DB2 catalog tables.
2. Bind checks the authorization of the user to declare bind and execute queries.
3. The bind component optimizer invokes the run statement utility and using statistics generates the available access paths.
4. Optimizer will choose the best one among them and place it in the package.

The “time stamp” taken is carried forward into the package and then into the plan.

“ the output of bind is PACKAGE OR PLAN.

* Package is DBRM and best access path which is non-executable.
* PLAN is a collection of packages which is executable .

3.EXECUTUION:-

RUN time superviser overseas the executuion of load module and plan by matching the “timestamp” tokens in there.

If they match it allows execution and .If do not matche throws an abbend with an abbend -818.

* Advantage of binding into a package instead of binding into a plan.
* If the DBRMS directly bound into plan then in case if any program his modified we need to bind the entire plan which will cause the other programs also to bound.
* \*Instead if you bind DBRMS into package first and then into plan, any program mod by just binding that package is sufficient and plane takes this latest package.

RUN STAT:-

It holds the statistics of the data base objects ,tables, columns,primarykeys,foreignkeys,indexs etc.

VIEWS:-

* View is a database object which acts like a virtual window to the base table data.

🡪 These are used for providing data security.

* View is similar to resultant table but resultant gets destroyed once program execution is over where as view is not destroyed .
* View doesn’t hold any data.

CREATE VIEW V-CC-TBL

AS

SELECT CC-NUM,

🡪 When select query is executed on the view then the data is retrieved from the base table into the view

There are two types of views

1.Updatable view

2.Non-updatable view(read only view)

1.Updatable view

A view is said to be updatable if it satisfies all the below conditions.

1. It must be created on a single table
2. No arithmetic operation or aggregate function should be used.
3. Ex :- SELECT SAL+100
4. No group be , having ,distinct must be used
5. No sub queries must be used
6. Base table columns other than view column must be nullable.

2.Non-updatable view(read only view)

A view is said to be read only view . If any one of the above specified condition are true.

**CURSORS:-**

Cursors is a pointer to a row in a resultant table .If we want to retrieve more than one row at a time , we go for cursors.

* Why we need CURSORS? (or) when do we use cursor .

NOTE: If a select queiry used in application ‘prog’ retrives more than one row at a time . To resolve this we use ‘CURSORS’.

* Life cycle of the cursors

1. Declaring the cursor
2. Opening the cursor
3. Fetching the cursor
4. Closing the cursor

1.Declaring the cursor :-

EXEC SQL

DECLATE POL CUR CURSOR

FOR

SELECT POL-AMT,PREM.AMT

FROM POL-TBL

WHERE POL-TYPE=’JB’

END-EXEC.

🡪 Declare cursor statement just creates the cursor declaration can be given ‘working storage selection’ or ‘ procedure division ‘.

🡪 To keep the cursor open even ofter ‘commit’ is issued.

EXEC SQL

DECLARE POLCUR CURSOR WITH HOLD

FOR

SELECT POL-AMT,PREM-AMT FROM POL-JB

WHERE TOL-TYPE=’JB’

END-EXEC.

NOTE:-

It I want to “commit” the updations ,(we are doing for the table) and again I want to update next 10 records, at that time the cursor will automatically closed because of “COMMIT”.If we want to hold the cursor until all the records are updated. Then cursor should declare with “WITHHOLD” option.

🡪**To update the table using cursor :**

1.Declaring the cursor:

EXEC SQL

DECLARE POLCUR CURSOR WITH HOLD

FOR

SELECT POL-AMT,PREM-AMT FROM POL-JB

WHERE TOL-TYPE=’JB’

FOR UPDATE OF PREM-AMT

END-EXEC.

2.OPENING THE CURSOR:

PROCEFURE DIVISION

EXEC SQL POLCUR

OPEN

END-EXEC

🡪 When open cursor statement is executed that quiery associated with that cursor is executed and the resultant table is created

The cursor is then pointed to the first row in the resultant table.

3. FETCHING THE CURSOR:

EXEC SQL

FETCH POLCUR

INTO : HV-POL-AMT,

: HV-PREM-AMT.

END-EXEC.

🡪 When fetch cursor statement executes it retrieves the row to which the cursor is currently pointing to that

4. CLOSING THE CURSOR:-

EXEC SQL

CLOSE POLCUR

END-EXEC

🡪 When close cursor statement executes it destroys cursor as well as the resultant table

CODING:

IDENTIFICATION DIVISION.

PROGRAM-ID PROG1.

DATA DIVISION.

WORKING-STORAGE SECTION.

EXEC SQL

INCLUDE SQLCA

END-EXEC

EXEC SQL

INCLUDE DCLPOL

END-EXEC.

EXEC SQL

DECLATE POLCUR CURSOR WITH HOLD

FOR

SELECT PREM-AMT FORM POL-TBL

WHERE POL-TYPE-‘JB’.

FOR UPDATE OF PRE,-AMT.

END-EXEC.

PROCEDURE DIVISION.

0000-MAIN-PARA.

PERFORM 1000-INITIALIZE-PARA

PERFORM 2000-FETCH-PARA

PERFORM 3000-PROCESS-PARA UNTIL SQLCODE

PERFORM 4000-CLOSE-PARA

STOP RUN.

1000-INITIALIZE-PARA

EXEC SQL

OPEN POLCUR

END-EXEC.

2000-FETCH-PARA

EXEC SQL

FETCH POLCUR

INTO : HV-PREM-AMT(3000)

END-EXEC.

3000-PROCESS-PARA.

EVALUATE SQLCODE

WHEN 0

COMPUTE HV-PREM-AMT = HV-PREM.AMT. \* 90/100

EXEC SQL

UPDATE POL0TBL

SET PREM-AMT= : HV-PREM-AMT

WHERE CUUENT OF POLCUR

ADD+1 TO WE-UPD-CNT

END-EXEC

IF WS-UPD-CNT = 10

EXEC SQL

COMMIT

END-EXEC.

MOVE 0 TO WS-UPD=CNT

END-IF

WHEN 100

CONTINUE

END-EVALUATE

PERFORM 2000-FETCH-PARA.

9000-CLOSE-PARA

EXEC SQL

CLOSE POLCUR

END-EXEC.

**BIND PACKAGE JCL:**

//JOB1 JOB

//STEP1 EXEC JPGM=IKJEFT01

//SYSTSIN DD \*

//BIND PACKAGE (PKG1)-

MEMBER(DBRM1-

OWNER(ALLST)-

EXPLAIN(YES)-

ISOLATION(CS/RR/UR)-

VALIDATION(BIND/RUN-

ACTION(ADD/REPLACE)-

ACQUIRE(USE/ALLOCATE)-

RELEASE(COMMIT/ROLLBACK).

/\*

1. **PACKAGE**: It specifies the package name into which the DBRM and best access path use bound into
2. **MEMBER**: It speciteds the DBRM name
3. **EXPLAIN:** It is a DB2 tool use to generate statistic of access path into the EXPLAIN table

Based on the information in explain tables DBA can analyse whether the information with the RUNSTAT is the updated one or not.

RUNSTAT:

JAN 1ST 2009

----------------

NOT UPDATED.

-------------------

4. **ACTION:**

1)ADD:- If it is new package ADD will include the package into “SYSIBM.SYSPACKAGES”.

2)REPLACE:- It well override the existing package version with the new one.

5**. ACQUIRE:** It acquires the locks(access) on the table spaces.

1)USE:- locks are aquited at the bind time (made available at the bind time)

2)allocate:- locks are acquitted at run time.(made available at the run time)

6**. RELEASE:-** It releases the locks on the table spaces

1)COMMIT:- It saves the changes made to the data in database permanently.

2)ROLLBACK:- It UNDO S (does not save) the changes made to the datebase.

NOTE:-upon successful execution of program and auto-commit is issued.

Upon unsuccessful execution of program an **out –rollback** is issued.

**VALIDATION**:- It checks the authorization of the user for binding or for executing of the queries.

**LOCKING MODES**:-

1. **SHARE LOCK(S**-LOCK):- it allows users to only read the data . More than one user can issue a “S-LOCK” on the same data.
2. **UPDATE LOCK(U-LOCK):-** It allows user to lock the data for updating only one user can issue S-LOCK.
3. **EXCLUSIVE LOCK(X-**LOCK) :- It allows to lock on to data to be updated only one user can issue X-LOCK and no other user can issue any other lock.

**ISOLATION :-** It specifies the level of locking

1. **CURSOR STEBILITY(CS)**:- It is row level locking. An exclusive lock (X-Lock) is acquired on the row to be updated. The lock is released on the next updatable row.

ROW:- LOCK,UPDATE,RELEASE

1. **REPEATABLE READ(RR**):- It is page level locking , An X-Lock is acquired on an entire page on which the row to be modified is present. The lock is released on the page only when the control moves to the next updatable page.
2. **UNCOMMITED READ(UR):-** It allows users to access the data which is modified but not yet committed. Here U-Lock is issued.

**OWNER**:- It specifies the cueator(owner) of the table. Owner-id will be same in both development and production.

**QUALIFIER**:- Qualifier is used to unequally identify a test when it has replication of tables.

🡪 The owner-id and qualifier are same in production when as in development they are different.

🡪 Multiple qualifiers are required in development to allow developers to test their program simultaneously on the same structure.

**BIND PLAN**

//JOB1 JOB ----

//STEP1 EXEC PGM=IKJEFT01

//SYSIN DD \*

BIND PLAN(PLN1)

PKLIST(PKG1,PKG2,----)

/\*

**RUN JCL**

//JOB2 JOB

//STEP1 EXEC PGM=IKJEFT01

//SYSIN DD \*

RUN PROGRAM(PGM1)

PLAN(PLN1)

LOADLIB(‘OZAO47.KEERTHI.LOADLIB’)

/\*

PLAN is a database object created by DBA.

**SUBQUERIES**

* A query with in a query is called sub queries .
* We can all here maximum of 15 sub queries.

**SYNTAX**  :

**SELECT** OUTER QUERY /MAIN QUERY WHERE (SELECT-------) SUB QUERIES / INNER QUERY

**SUB QUERIES** are of two types.

1. Non-corelated sub queries
2. Correlated sub queries
3. **Non- correlated sub queries**:- First the inner query executed and based on its results the outer query executed.

**Ex**:- third max salary

SELECT MAX(SAL) FROM EMP0TABLE

WHERE SAL<(SELECT MAX(Sal) FROM EMP-TABLE

WHERE SAL<(SELECT MAX(SAL) FROM EMP-TABLE)

EX:- To retrieve second min salary

SELECT MIN(SAL) FROM EMP-TBL

WHERE SAL>(SELECT MIN(SAL) FROM EMP-TBL)

1. **CORRELATED SUB QUERUES**:-First the outer query executes and for each row of outer query the inner query executes

**NOTE:**- ALIAS names are created at the time of executing the query and it will be deleted after the execution.

E1.SAL E2.SAL

20000 20000

15000 15000

25000 25000

NOTE:- To fine “n’th” max salary we will mention “(n-1)”

**To fine the second min salary**.

SELECT E1.SAL FROM EMP-TBL E1

WHERE 1- (SELECT COUNT(DISTINCT E2.SAL)

**TO find n max salaries**

SELECT E1.SAL FROM EMP-TBL E1

WHERE N>(SELECT CON(DISTINCT E2.SAL FROM EMP-TBL E2 WHERE E2.SAL>E.SAL)

**CONSTRAINTS** :-

Constraints is a mechanism to control data in the table columns.

1. NULL CONSTRAINT:
2. NULL INDICATOR VARIABLES:-

EMP-NAME CHAR(30),HV-EMP-NAME X(30) HV.EMP-NAME-IND S9(4)

If any null indicators are there in the tables then we have to mention YES as beside . Then it will creates the host variables as the above

SELECT EMP-NAME INTO :HV-EMP-NAME :HV-EMP-IND FROM EMP-TBL------------------------

IF HV-EMP-NAME-IND<0

MOVE SPACES TO HV-EMP-NAME

ELSE IF(>0)

DISPLAY DATA TRUNCATION

END-IF

END-IF

* NULL INDICATOR VARIABLE are used to know, if any NULL values are retived from DB2 into COBOL variables.
* If indicator value is <0 then it indicates NULL value is retrived and since COBOL default values based on the DATA TYPE.
* If indicator value is 0 then value is properly retrived.
* If value is >0 the the data retrived is truncated.

**Referential Integrity:-**

It controls the data in the parent and child tables.

FOREIGN KEY:

* It is used to build relationship between tables.
* It must be a primary key of a table and when used in other tables become foreign key.

NOTE:

* The table in which it is primary key is called parent table.
* The table in which it is foreign key is called child table.

1. INSERT RULE: Insert rule says before inserting a row in a child table insert into parent table first.
2. UPDATE RULE: It says before updating the child table update if in the parent table.
3. DELETE RULE:
4. ONDELETE CASCADE:-When a parent table row is deleted, the correspond row in the child tables also get deleted.
5. ON DELETE RESTRICT:\_ When a parent table row is to be deleted a restriction is applied when there are corresponding rows in the child tables.

So, first we need to delete the child table rows and then we delete the parent table row.

c).ON DELETE SET NULL:- When a parent table row is deleted the foreign key values in the child table are set to NULL.

**CICS(customer information control system)**

* It is the online system in the mainframe.
* CICS itself acts as an operating system under Z/OS handling the online programs.

|  |  |
| --- | --- |
| **Batch**   1. No user interaction 2. If no CICS is used it is batch 3. Compile using JCL 4. Executed using JCL 5. No instant result   Job1  Job5  Job3  Job2  Job6   1. Payroll applications, reporting applications 2. Backend processing 3. Data processing is very fast due to no user interaction. | **Online**   1. There will be user interaction 2. If CICS is used then it is online 3. Compile using JCL 4. Executed using transaction ID 5. We can see instant results   Screen2  Screen1  Screen3  Screen4  Like ISPF it will relates one screen to another.   1. ATM, reservation systems 2. Frontend processing 3. Data processing is slow because there will be user interaction. |

Z/OS:

CICS

P2

P1

P4

P3

P2

P1

* CICS supports multitasking and multicasting.

1. Multitasking: Executing multiple programs simultaneously.
2. Multithreading: Executed multiple programs simultaneously and executing same couples of load module programs executing simultaneously.

Note: Multitasking is the subject of multithreading.

Task: It is an instance of a transaction.

Transaction: It is a set of tasks.

Ex: (ATM operation)

Task1: prompts the user to enter the pin no

Pin:---------

Task2: once the user enter the PIN and presses enter, the card & PIN are validated.

If validate,

Through an options screen

ELSE

ERROR message “please enter correct PIN”

END\_IF

Task3: user selects withdraw option, then throws next screen prompting to enter the amount.

After amount is given and account is updated then the transaction is completed,

Note: All these tasks together are called as transaction.

WAYS to initiate a program:

1. By entering the “transaction-ID” in the left top most corner of the screen.

Transaction-ID:

(Programing)

1. By using pseudo-conventional technique i.e. using return command (RETURN)
2. By using “XCTL” and LINK commands.
3. By using “START” command
4. By using “ATI”(automatic transaction initiation)

|  |  |
| --- | --- |
| TRANS-ID  (internal transaction process)   |  | | --- | | PCT PPT  Prog.names load library  PROG1 0ZA0SUP.CICS.LOADLIB  PROG2 0ZA0SUP.CICS.LOADLIB  --------- -----------------------------  Tn. ID Prog. Names  TR01 PROG1  TR02 PROG2  TR05 PROG5  FCT RCT  Program names plan names  PROG1 PL1  PROG2 PL2  --------- ----------  ACCS.FILE 0ZA046.ACCTS.FILE  POL.FILE 0ZA046.POL.FILE  Logical filenames | |

Program control table (PCT): Trans

* It contains the transaction-ID and corresp.prog

Processing programing table (PPT):

* It contains program names and its corresponding load libraries.

File control table (FCT):

* It contains the logical & physical file names.

Note: Unlike batch programs we can have only one logical file for a physical file.

Resource control table (RCT):

* It contains the program names and corresponding plan names

“How a transaction is process initially?”

* Once a transaction-ID is entered it is first checked in PCT and if it exist it will CICS the corresponding program name.
* Then CICS checks the program name in PPT and copies the load module from the specified load library into the buffer.
* Then the program execution begins
* If any files are used then the logical file name is checked in FCT and records are accessed from the corresponding physical file.

STEPS INVOLVED IN DEVELOPING AN ONLINE PROGRAM:

1. DESIGNING THE MAP
2. CODE THE BMS MACROS FOR THE MAP
3. DESIGN THE PROGRAM FOR THE CORRESPONDING MAP
4. CODE THE PROGRAM
5. COMPILE THE BMS MACROS AND THE PROGRAM TO GENERATE LOAD MODULES
6. MOVE THE LOAD MODULES FROM BATCH TO ONLINE USING CEMT
7. EXECUTE THE PROGRAM USING TRANSACTION\_ID

BMS (basic mapping support):

* It is a set of assembly level MACROS used to design the map.

MAP:

* It is one screen format in CICS

MAPSET:

* It is a group of MAPS

There are 3 important BMS maons to define a map set, a map and a field.

1. DFHMSD (defined field hierarchy map set definition)
2. DFHMD1 (defined field hierarchy MAP definition interpreter)
3. DFHMDF (defined field hierarchy MAP definition field)

ICIMST1 DFHMSD TYPE=&SYSPARM/MAP/SECT,

MODE=INPUT/IN/OUT,

T10APEX=YES,

LANG=COBOL/ASM/PL,

STORAGE=AUTO,

CTRL= (FREEKB,FRSET)

ICIMP1 DFHMD1 SIZE= (24,80),LINE=1,COLUMN=1

DFHMDF INITIAL=‘ICICI’,POS=(1,38),LENGTH=05,

ATTRB=ASKIP

DFHMDF INITIAL=’------------’,POS=(2,38),LENGTH=05,

ATTRB=ASKIP

DFHMDF INITIAL=’USERID:’,POS=(10,31),LENGTH=10,

ATTRB=ASKIP

UID DFHMDF POS=(10,42),LENGTH=10,ATTRB=(UNPORT,IC,FSE),

DFHMDF POS=(10,53),LENGTH=1,ATTRB=ASKIP

DFHMDF INITIAL=’PASSWORD:’,POS=(11,31),LENGTH=10,

ATTRB=ASKIP

PWD DFHMDF POS=(11,42),LENGTH=10,ATTRB=(ONPROT,FSET,DRK)

DFHMDF POS=(10,53),LENGTH=01,ATTRB=PROT

DFHMDF INITIAL=’ENTER:LOGIN FS:REFRESH’,

POS=(20,21),LENGTH=50,ATTRB=ASKIP

ERRMSG DFHMDF POS=(22,06),LENGTH=50,ATTRB=(ASKIP,BRT)

ICICI

USERID:

PASSWORD:

ENTER:LOGIN FS:REFRESH

IC1MST1 DFHMSD TYPE=FINAL

END

1. TYPE: If specifies the type of the map to be generated.
2. &SYSPROM: It will generates both physical and symbolic map
3. MAP: It generates only the physical map
4. DSET: It will generate only symbolic map

* Physical map is the load module of the BMS MACROS
* Symbolic map is the copy book generated for the fields with field name.

1. MODE: It specifies the type of use of map.

INPUT: It enables the map to be used for both Input and output pupose.

IN: -----------

OUT: ------------

TIOAPFX (terminal input output area prefix): when TFOAPFX=YES then system generates a 12 byte filler in which the control information pressed by the user.

The 12 byte memory is TIOA

LANG: It specifies the programing language with which we can use the map.

STORAGE: When STORAGE=AUTO, is specified system allocates a separate memory for the map set.

FREEKB: It unlocks the keyboard before the map is displayed.

SIZE: It determines the size of the map to be generated.

LINE & COLUMN: Determines the position of the map on the screen.

INITIAL: It is similar to value clause in assigning the data statistically to a field.

POS: It specifies the position of the field in the map.

LENGTH: It specifies the length of the field.

ATTRB: It specifies the characteristics of the field.

1. ASKIP: It skips the control over a field.
2. UNPORT: It stops the cursor at the fixed and allows the user to enter the data.
3. PROT: It stops the curser at the field but doesn’t allow the user to enter the data.
4. DRK: It darkens the field making the field invisible to the user.
5. BRT: It displays the field brightly.
6. IC: Insert curser, it determines the position of the cursor.

Note: 1. If NO IC is specified then cursor will be positioned at the first byte in the map.

2. If IC is specified for more than one field then cursor will be positioned at the last unprotected and IC specified field.

IC --------------------------------

IC --------------------------------

IC --------------------------------

IC --------------------------------

SKIPPER TECHNIQUE: To skip the control after the user enters the maximum characters in a field of length ‘n’ the ‘n+1th’ byte is specified as ASKIP.

STOPPER TECHNIQUE: To stop the control after maximum characters are entered the ‘n+1th’ byte is specified as PROT.

Write the BMS macro for the below screen?

1. Enter new policy details.
2. View policies by status.
3. View history policies.
4. View CLAIMS INFO

Option:

F1:VIEW F3:EXIT F5:REFRESH

Symbolic map:

01 ICI MAP1I

02 FILLER PIC X(12)

02 UIDL PIC SQ(4)COMP

02 UIDF PIC X(1)

02 FILLER REDEFINESUIDF

03 UIDA PIC X(1)

02 UIDI PIC X(10)

02 PWDL PIC X(10)

02 PWDF PIC X(1)

02 FILLER REDEFINE1 PWDF

03 PWDA PIC X(1)

02 PWD1 PIC X(10)

02 ERRMSGL PLC S9(4) COMP

02 ERRMSGF PIC X(1)

02 FILLER REDEFINES ERRMSGF

03 ERRMSGF PIC X(1)

02 ERRMSGA PIC X(50)

01 ICIMAO REDEFINES ICIMP1I

02 FILLER PIC X(12)

02 FILLER PIC X(03)

ADD LINE1 ------------------------------

2 ------------------------------

3 ------------------------------

02 UIDO PIC X(10) FSET

02 FILLER PIC X(03) FSET

02 PWDO PIC X(10) FSET

02 FILLER PIC X(03)

02 ERRMSGO PIC X(50)

FIELD+L It is the length field in which the count of the characters entered on the screen is stored.

FIELD+A It stores the characteristics of the field like protect, unprotect, bright, dark.

FIELD+I Use to receive data from screen to program

FIELD+O Use to pass data from program to screen.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | UNPROT(0)  PROT(1) | ASKIP(1) | BRT(0)  DRK(1) | IC(1) |  | MDT IS OFF(0)  MDT IS ON(1) |  |

FLAG BIT

MDT: modified data tag

If the user does not enter any data into a field then its MDT will be ‘0’ i.e. OFF. It turns ON (or set to 1) only when user enters some data.

FRSET: FRSET will reset all the fields, MDT to ‘0’, FSET will set the MDT to 1 for a field, FSET over rides FRSET.

EX: when there are 3 lines for address field line b1’s MDT should be set to ‘0’ using FRSET and LINE2 and 3must be set ‘1’ using FSET. This is because user may or may not enter data in LINE2 and LINE3 and still those fields must be record into the program.

COMM AREA:

* It is the communication area between the two programs or two tasks.
* COMM area is declared in the COBOL program as follows.

LINKAGE SECTION

01 DFHCOMMAREA PIC X(100)

* In every CICS program DFHCOMMAREA is declared and the maximum data length can be 32767(32k)
* EIBCALEN(EXEC interface block communication area length)
* It is used to know whether he user has entered data in the screen or not
* It stores the count of characters entered on the screen
* If EIBCALEN is ‘0’ then it indicates user has not entered any data. ELSE User has entered data.

Note: If NO DFHCOMMAREA declared then system takes a one byte (1B) DFHCOMMAREA.

Pseudo conversation:

This programming technique is used to release the CPU resources once a task is executed. This will avoid unnecessary utilization of CPU resources till the user response. Pseudo conversation is handled in the App program using RETURN command.

EXEC CICS

RETURN

END EXEC

RETURN: It terminates the CICS program and passing control to the most immediate higher level. To execute the same program again when user responds with the return command we need to specify the transaction id. This will cause CICS to still maintain a transaction id even after program is terminated.

EXEC CICS

RETURN

TRANSID (‘ICI1’)

END EXEC

**SAMPLE PROGRAM:**

ICICI

USERID:------------------

PASSWORD:------------------

ENTER:LOGIN FS:REFRESH

IDENTIFICATION DIVISION

PROGRAM\_ID PROG1

DATA DIVISION

WORKING-STORAGE SECTION

COPY ICIMP1

COPY DEHAID

EXEC SQL

INCLUDE SQLCA

END EXEC

LINKAGE SECTION

01 DFHCOMMAREA PIC X(100)

PROCEDURE DIVISION

0000\_MAIN\_PARA

IF E1BCALEN=0

PERFORM 1500\_SEND\_ICIMP1

ELSE

PERFORM 3000\_PROCESS\_PARA

END\_IF

EXEC CICS

RETURN

TRANS ID (‘ICI1’)

END\_EXEC

1500\_SEND. ICIMP1

EXEC CICS

SEND MAP (‘ICIMP1’)

MAPSET (‘ICICMST1’)

2000\_RECEIVE\_ICIMP2

EXEC CICS

RECEIVE MAP(‘ICIMP1’)

MAPSET (‘ICIMST1’)

INTO (ICIMP1I)

END EXEC

3000\_PROCESS\_PARA

EVALUATE E1RA1D

WHEN DFHENTER

PERFORM 2000\_RECEIVE\_ICICMP1

PERFORM 3000\_VALIDATE\_PARA

WHEN DFHPF5

MOVE SPACES TO VI00,PWDO,ERRMSGO

PERFORM 1500\_SEND\_ICIMP1

END\_EVALUATE

3500\_VALIDATE\_PARA

MOVE UIDT TO HV\_USER\_ID

MOVE PWDI TO HV\_PWD

EXESQL

SELECT USE\_ID\_INTO\_:HV\_USER\_ID

FROM AVTH\_TBL

WHERE USER\_ID=:HV\_USER\_ID

AND PWD=:HV\_PWD

END\_EXEC.

EVALUATE SQLCODE

WHEN 0

EXEC CICS

XCTL PROGRAM(‘PROG2’)

END EXEC

WHEN 100

MOVE ‘INVALID USERID/PWD’ TO ERRMSGO

PERFORM 1500\_SEND\_ICIMP1

END EVALUATE

**Important:**

IF EIBCALEN=0

MOVE DFHBMUNP TO ERRMSGO,UIDO,PWDO

MOVE 80 TO WS\_LENGTH

PERFORM 1500\_SEND\_MAP1

ELSE

PER

END\_IF

EVALUATE EIBRSP

WHEN DFHRESP(NORMAL)

IF PWDI=PWD

-------------------

-------------------

ELSE

MOVE ‘PLS ENTER CORRECT PASSWORD’ TO ERRMSG

MOVE -1 TO PWDL

PERFORM

**XCTL:**

It is used to pass control from one program to another but doesn’t expect the control to return back.

EXEC CICS

XCTL PROGRAM(PROGRAM NAME)

END EXEC

Ex: PROG1 CICS

---------- PROG2

---------- --------------

EXEC CICS ---------------

XCTL PROGRAM (‘PROG2’) EX

END\_EXEC

---------------

--------------

EXPC CICS

RETURN

EXID

* When XCTL is used the program which pass the control, the program which are set to be at some level.

LINK: It is used to pass control from one prog. to another but the control is expected to return back.

The programs are set to be at different levels.

LINK: CICS

PROG1

----------

---------- PROG2

EXEC CICS ------------

LINK ------------

PROGRAM(‘PROG2’) EXEC CICS

ENDEXEC RETURN

----------- END\_EXEC

-----------

EXEC CICS

RETURN

END\_EXEC

🡪 Continuation of prog1 with prog2 of previous class

ICICI

UserID:

Password:

1. View A/C details

2. View LAST 5 transactions

3.

4.

Options:

F1: view F3: exit F5:refresh

P3

P4

P5

IDENTIFICATION DIVISION

PROG.ID. PROG2

DATA DIVISION

WORKING\_STORAGE SECTION

COPY ICIMP2 UNKAGE SECTION

COPY DFHAID 01 DFHCOMMAREA PIC x (100)

COPY DFHBMSCA

PROCEDURE DIVISION

0000\_MAIN\_PARA

IF FIBCACEN=0

PERFORM 1500\_SEND\_ICIMP2

ELSE

PERFORM 1500\_SEND\_ICIMP2

END\_IF

EXEC CICS

RETURN

TRANSID (‘ICI2’)

END \_EXEC

1500\_SEND\_ICIMP2

EXEC CICS

SEND MAP (‘ICIMP2’)

MAPSET (‘ICIMST1’)

END EXEC

1600\_SEND\_ICIMP2

EXEC CICS

SEND MAP ( )

MAPSET( )

DATA ONLY

END\_EXEC

2000\_RECEIVE\_ICIMP2

EXEC CICS

RECEIVE MAP(‘ICIMP2’)

MAPSET(‘ICIMST1’)

INTO(‘ICIMP2’)

END\_EXEC

-------------

-------------

3000\_PROCESS\_PARA

EVALUATE EIBAID

WHEN DFHPF1 PERFORM 2000\_RECEIVE\_ICIMP2

WHEN DGHPF5

MOVE SPACES TO OPTION0,ERRMSG0

PERFORM 1600\_ICIMP2\_DATAONLY

WHEN OTHER

MOVE ‘INVALID KEY PRESSED’ TO ERRMSF10

PERFORM 1600\_SEND\_ICIMP2\_DATAONLY

END\_EVALUATE

3200\_OPTIONS\_PARA

EVALUATE OPTIONI

WHEN ‘1’

EXEC CICS

XSTL PROGRAM(‘PROG3’)

WHEN ‘2’

EXEC CICS

XCTL PROHGRAM(‘PROG4’)

END EXEC

WHEN ‘3’

EXEC CICS

XCTL PROGRAM(‘PROG5’)

END EXEC

WHEN ‘4’

EXEC CICS

WHEN OTHER

MOVE ‘INVALID OPTION SELECTED’ TO ERRMSE10

PERFORM 1600\_SEND\_ICIMP2\_DATAONLY

END\_EVALUATE

**DFHALD:** It is a system defined copy book with all the function keys information.

01 DFHAID

02 DFHENTER PIC X(5) VALUE ‘;’

02 BFHPF1 PIC X(5) VALUE

02 DFHPF2 PIC X(5)

02 DFHPF3 PIC X(5)

02 DFHPF4 PIC X(5)

02 DFHPF5 PIC X(5)

* Sending a map without a program

In the CICS specify the below command

CECI SEND MAP(‘ICIMP2’) MAPSET(‘ICIMST1’)

* Sending a map using an application program

Send command:

EXEC CICS

SEND MAP(‘MAP1’)

MAPSET(‘MAPST1’)

END EXEC

* This will send both physical and symbolic map

EXEC CICS

SEND MAP (‘MAP1’)

MAPSET(‘MAPST1’)

DATAONLY

END\_EXEC

* ‘DATAONLY’ will send only the symbolic map

EXEC CICS

SEND MAP(‘MAP1’)

MAPSET(‘MAPST1’)

MAP ONLY

END\_EXEC

* “MAPONLY” will send only the physical map.

“RECEIVE” without app prog

It is used to receive the data entered by the user on the screen into input variables.

EXEC CICS

RECEIVE MAP(‘MAP1’)

MAPSET(‘MAPST1’)

INTO(MAP1I)

END\_EXEC

Note: If any of the fields MDT=0 then the receive command fails and with map fail.

**CURSOR POSITIONING TECHNIQUE**

1. STATIC CURSOR POSITIONING
2. DYNAMIC CURSOR POSITIONING
3. RELATIVE CURSOR POSITIONING

**1.STATIC CURSOR POSITIONING:** USING IC

**2.DYNAMIC CURSOR POSITIONING:**

Dynamically to place the cursor into a desired field we need to move ‘-1’ to its length.

MOVE ‘PLEASE ENTER CORRECT PASSWORD’ TO ERRMSG0

MOVE -1 TO PWDL

PERFORM 1600\_SEND.

**3.RELATIVE CURSOR POSITIONING:**

Relatively the cursor is positioned at a desired field by specifying the relative byte position compared to the first byte.

EXEC CICS

SEND MAP(’MAP1’)

MAP(‘MAPST1’)

CURSOR(842)

END\_EXEC.

**PROG3:**

OPTION:1

F1:

IDENTIFICATION DIVISION

PROGRAM\_ID PROG3

DATA DIVISION

ENTER A/C NO: A123

F1:VIEW DETAILS F3:EXIT

F5:REFRESH F12:LOGOUT

WORKING-STORAGE SECTION

COPY ICIMP3

COPY DFHAID

COPY DFHBMSCA

ACCT DRT

ACC NO:

NAME:

BALANCE:

F3:EXIT F12:LOGOUT

01 WS-COMMAREA

03 FILLER PIC X(100)

03 WS\_COMM\_PGMID PIC X(10)

03 WS\_COMM\_ACCT\_NUM PIC X(30)

03 WS\_COMM\_ERRMSG PIC X(50)

LINKAGE SECTION

01 DFHCOMMAREA

PROCEDURE DIVISION

0000\_MAIN\_DARA

IF EIBCALEN=0

PERFORM 1500\_SEND\_ICIMP3

ELSE MOVE DFCOMMAREA TO WS\_COMMAREA

MOVE SPACES IF WS\_COMM\_PGMID=’7’

TO WS\_COMM\_PGMID MOVE\_COMM\_ERRMSG TO ERRMSG0

PERFORM 1500\_SEND\_ICIMP3

ELSE

PERFORM 3000\_PROCESS\_PARA

END\_IF

EXEC CICS

RETURN

TRANSID(‘ICI3’)

COMM AREA(WS\_COMMAREA)

END\_EXEC

1500\_SEND\_ICIMP3

EXEC CICS

RECEIVE MAP (‘ICIMP3’)

MAPSET(‘ICIMST1’)

INTO (ICIMP3I)

END\_EXEC

3000\_PROCESS\_PARA

EVALUATE EIBAID

WHEN DFHPE1

MOVE ACCNUMI TO WS\_COMM\_ACCT\_NUM

MOVE ‘3’ TO WS\_COMM\_PGMID

EXEC\_CICS

XCTL

PROGRAM(‘PROG7’)

COMMAREA(WS\_COMMAREA)

END\_EXEC

WHEN DFHPFS

EXEC CICS

XCTL PROGRAM(‘PROG2’)

END\_EXEC

WHEN DFHP5

MOVE SPACES TO ACCNO,ERRMSG0

PREFORM 1500\_SEND\_ICIMP3

WHEN DFHPF12

EXEC CICS

XCTL PROGRAM(‘PROG1’)

END\_EXEC

WHEN OTHER

MOVE ‘INVALID KEY PRESSER’ TO ERRHSG0

MOVE DFHBRT TO ERRHSGA

PERFORM 1500\_SEND\_ICIMP3

END\_EVALUATE.

(PROG7): CHECKING THE ACCOUNT NO ENTERED IN THE PROG-3. WHEATER IT IS EXISTED IN PROG FILE OR NO

CODE:

IDENTIFICATION DEVISION.

PROGRAM-ID. PROG7.

DATADIVISION.

WORKING-STORAGE SECTION.

COPY ICIMP7

COPY CPYACLT

COPY DFHAID

COPY DFHBMSCA

01 WS-COMMAREA

03 FILLER PIC X(100)

03 WS-COMM-PGMID PIC X(10)

03 WS-COMM-ACT-NUM PIC X(30)

03 WS-COMM-ERRMSG PIC X(50)

LINKAGE SECTION

01 DFHCOMMAREA PIC X(190)

PROCEDURE DIVISION

0000-MAIN-PARA

MOVE DFHCOMMAREA TO WS-COMMAREA

IF WS-COMM-PGMID=’3’

MOVE SPACES TO WS-COMM

MOVE WS-COMM-ACCT-NUM TO ACC-NUM

PERFORM 3000-PROCESS-PARA

ELSE

PERFORM 3000-PROCESS-PARA

END-IF

ELSE CICS

RETURN

TRANSID(‘ICI1’)

COMMAREA(WS-COMMAREA)

END-EXEC

1500-SEND-ICIMP7

EXEC CICS

SEND MAP(‘ICIMP7’)

MAPSET(‘ICIMPT1’)

END-EXEC

2500-RETRIEVE-ACCT-DET

EXEC CICS

READ FILE(‘ACCIFL’)

RIDFLD(‘ACC-NUM’)

IN0(ACCT-REC)

END-EXEC

EVALUATE EIBESP

WHEN DFHRESP(NORMAL)

MOVE ACCT-NUM TO ACCNUM0

MOVE ACCT-HLDR-NAME TO HLDRNM0

MOVE ACCT-BAL TO ACCTALO

-----------------------------

-----------------------------

PERFORM 1500-SEND-ICIMP7

WHEN DFHRESP(NOT END)

MOV ‘7’ TO WS-COMM-PGMID

MOV ‘INVALID ACCNT NUMBEN’ TO WS-COMM-ERR

EXEC CICS

XCTL

PROGRAM(‘PROG3’)

COMMAREA(WS-COMMAREA)

END-EXEC

END-EVALUATE

3000-PROCESS-PARA

EVALUATE EIBAID

WHEN DFHPF3

EXEC CICS

XCTL PROGRAM(‘

WHEN DFHPF12

**(ACCESSING FILES IN A CICS PROGRAM)**

* In CICS applications we use VSAM(KSDS) files to make the access faster.
* In case of VSAM(KSDS) files if some records are deleted physically their memory is released into the free space.
* And also is rearranges all the records so that the amount of time taken to scan the data will reduce.
* In case of flat files (sequential files) we cannot delete a record physically.
* Unlike flat files VSAM(KSDS) files support random and dynamic access also.

Writing records into a VSAM file:

EXEC CICS

WRITE FILE(‘POL-FILE’)

FROM(POL-REC)

END-EXEC

Reading the records in VSAM randomly:

EXEC CICS

READ FILE(‘POL-FILE’)

RIDFLD(POL-NUM)

LENGTH(80)

INTO(POL-REC)

Note: we cannot use the different file names when we are using flat files. Where as in VSAM files we can use any file name to access the data into.

RIDFLD (record identification field): It specifies the key field based on which the data in KSDS file is sufficient.

LENGTH: It specifies the length of a record retrieved.

INTO: It specifies the record name into which the record data is retrieved.

GENERIC: It allows only read access to the file data(if we want to display it will just display)

Reading records sequentially:

Low-values(all the bit set to 0’s) MOVE LOW-VALUES TO POL-NUM

----------------- EXEC CICS

----------------- STARTBR FILE(‘POL-FILE’)

0-9 RIDFLD(POL-NUM)

Alphabets KEYLEN(0)

Special symbols END EXEC

PERFORM UNTIL WS-EOF=’Y’

EXEC CICS

HANDLE CONDITION

ENDFILE(9999-PARA)

LENDER(9000-PARA)

END-EXEC

EXEC CICS

(READNEXT) FILE (‘POL-FILE’)

READ PREV< RIDFLD(POL-NUM)

INTO(POL-REC)

END-EXEC

END-PERFORM

EXEC CICS

ENDBR FILE(‘POL-FILE’)

END-EXEC

9999-PARA

MOV ‘Y’ TO WS-EOF.

STARTBR (START BROWSE):

* It is used to set the cursor position at a desired record.
* Key length must be always ‘0’.
* To set the cursor position at the first record in the file move “low values” to the “key field”.
* To set the cursor position at the last record move ”high values” to the “key field”.

READ NEXT: It is used to retrieve the records sequentially in the forward direction.

READ PREV: It retrieves the records sequentially in reverse direction.

ENDBR: When start browse is use it is mandatory to use END BROWSWSE to end the sequential access on the file.

Reading the records dynamically:

MOVE ‘P121’ TO POL-NUM

EXEC CICS

STARTBR FILE(‘POL-FILE’)

RIDFLD(POL-NUM)

KEYLEN(0)

END-EXEC

PERFORM UNTIL WS-EOE=’Y’

EXEC CICS

HANDE CONDITION

ENDFILE(9999-PARA)

LENERR(9000-PARA)

END-EXEC

EXEC CICS

READNEXT FILE(‘POL-FILE’)

RIDFLD(POL-NUM)

How do you handle exceptions in CICS?

1. USING HANDLE CONDITION
2. USING RESP

1.USING HANDLE CONDITION: All the exceptions that may encounter after a CICS command is executed must be specified in the handle condition before the CICS command  
🡪 modifying an existing record in the field:

Before this MOVE 2500 TO PREM-AMT

We have to EXEC CICS

Read the file RWWRITE FILE(‘POL-FILE’)

RIDFLD(POL-NUM)

FROM(POL-REC)

MOVE ‘P128’ TO POL-NUM

EXEC CICS

READ FILE(‘POL-FILE’)

RIDFLD(POL-NUM)

INTO(POL-REC)

LENGTH(110) ADD 100 TO POL-AMT

UPDATE EXEC CICS

END-EXEC

“REWRITE”: Is used to save the modified record in the file.

UPDATE: It must be specified in order to use REWRITE command.

RESP: It generates a response code after a CICS command is executed. Response code can be executed in the program using ‘EIBRESP’.

EXEC CICS

READ FILE(‘ACCT-FL’)

RIDFLD(ACCT-NUM)

If we want to handle exceptions the we have to mention the RESP in the EXEC CICS and END-EXEC

INTO(ACCT-REC)

RESP

END EXEC

EVALUATE EIBRESP

WHEN DFHRESP(NORMAL)

WHEN DFRESSP(NOT END)

MOVE ‘INVALID ACCOUNT NUMBER’ TO ERRMSG

PERFORM 1500-SEND-MAP1

WHEN DFHRESP(LENERR)

WHEN DFHRESP(DUPREC)

Usually it terminates if we want to handle we have to mention like above

----------------------------

----------------------------

EXCEPTIONS

NORMAL NOTOPEN NOTFND PGMIDERR LENERR INVR50 MAPFAIL QIPERR TRSIDERR

EIBRESP

CICS Transactions:

1. CEDA (CICS EXECUTION DYNAMIC ALLOCATION)

To define & install a map set:

CEDA DEF PROGRAM (MAPSET NAME) GROUP(AXIS)

CEDA INS PROGRAM (MAPSET NAME) GROUP (AXIS)

To define & install a program:

CEDA DEF PROGRAM (PGM NAME) GROUP(AXIS)

CEDA INS PROGRAM (PGM NAME) GROUP (AXIS)

To define & install a transaction id:

CEDA DEF TRANS (‘AX01’) PROGRAM (PROG1) G(AXIS)

CEDA INS TRANS (‘AX01’) PROGRAM (PROG1) G(AXIS)

1. CEMT(CICS EXECUTION MASTER TERMINAL)

To bring a load module from batch to online

CEMT SET PROGRAM (PROG1) NEWCOPY

MAPS NAME

“NEW COPY” will replace (override) the already existing load module version with the new version from batch.

To know the transaction-ID for a program:

CEMT INQUIRE TRAN PROG (PROG1)

To know the program name associated with a transaction-ID:

CEMT I FILE (‘ACCT-FL’)

‘0ZAO46.ACCTS.FILE’ OPEN ENABLED SHR

CLOSED DISABLED

* To open a file in a batch we need to close and disable the file in online.
* Similarly to use it online it must not be used in batch not even in browse mode
* CEDF (CICS EXECUTION DIAGNOSTIC FECILITY)

It is used to debug CICS commands in an online program.

USER ID:

PASSWD:

UPD DEL

RETURN

TRANS (‘AX01’)

SEND MAP (‘MAP1’)

MAPSET ( )

AX01

EDF mode ON

CEDF

CICS EXECUTION BROWSE (CEBR)

Q1D001

A123 100000

A124 200000

----- --------

----- --------

It is used to browse the data stored in queue.

CEBR QTD001

CESN (CICS EXECUTION SIGN ON):

It will prompt the user to enter his user id and password to log on CICS.

CESE (CICS EXECUTION SIGN OFF):

It log off the user from CICS region.

CECI (CICS EXECUTION COMMAND INTERPRETOR):

To test a map without a program.

CECI SEND MAP (‘MAP1’) MAPSET (MAPSET1)

**QUEUES**

* It is a storage area similar to files storing data in the form of records.
* A queue is identified by a queue-id which can be maximum 8 characters.
* CICS command CEBR is used to browse the data in queue.
* Data can be shared between programs using COMM AREA and queues. Using COMM AREA we can pass max 32 char data whereas using queues we can pass more than 32k.

Queues are of 2 types:

1. TSQ (temporary storage queue)
2. TDQ (transient data queue)
3. TSQ: Queue records are written based on item numbers. TSQ records can be accessed sequentially, dynamically or randomly.
4. Writing records into TSQ:

MOVE 1 TO WS-TSQ-ITEM

EXEC CICS

WRITEQTS

QUEUE (‘QID0001’)

FROM (WS-TSQ-DATA)

ITEM (WS-TSQ-ITEM)

1 A123 1000

2 A124 2000

END-EXEC.

NOTE: TSQ is created once the write queue command is executed for the first time.

Reading records from TSQ:

MOVE 1 TO WS-TSQ-ITEM

EXEC CICS

READQ TS

QUEUE (‘QID0001’)

ITEM (WS-TSQ-ITEM)

INTO (WS-TSQ-DATA)

END-EXEC.

Deleting the TSQ: It completely deletes the queue.

EXEC CICS

DELETEQ TS

QUEUE (‘QID001’)

END-EXEC.

TDQ: records are written without item number. Records can be real only sequentially. A TDQ record once read is deleted automatically.

Write records into TDQ’s:

EXEC CICS

WRITEQ TD

QUEUE (‘QID001’)

FROM (WS-TDQ-DATA)

END-EXEC.

Reading records from TDQ:

EXEC CICS

READQ JD

QUEUE (‘QID001’)

INTO (WS-TDQ-DATA)

END-EXEC.

|  |  |
| --- | --- |
| TSQ   1. Records are written based on item numbers. 2. A record can be read multiple times. 3. Records can be accessed sequentially and dynamically. 4. TSQ is required in TST (temporary storage table). | TDQ   1. Records are written without item number. 2. Only once. 3. Records can be read only sequentially. 4. Is required in DCT (destination control table). |

**ATI (AUTOMATIC TRANSACTION INITIATION)**

* It is automatically triggered or initiate the transaction once the trigger level is reached i.e. once the no. of records reach us the count specified in TRGLEV (trigger level) the transaction is automatically initiated.

TYPE-INTRA/EXTRA

No. of records to be reached then this program will executes.

QUEUE= ‘QID001’

TRANSID= IC05,

TRGLEV= 30

TDQs are of two types.

1. INTRA PARTITIONED
2. EXTRA PARTITIONED

INTRA PARTITIONED: This type of queue can be used with in a single CICS region.

EXTRA PARTITIONED: This type of queue can be used in more than one CICS region and outside CICS region (batch program).

Pseudo code for page up and page down:

1. Read the file sequentially and load the desired data into TSQ.
2. MOVE the initial 10 records from TSQ TO 10 Lines in the MAP send the map.
3. User presses F7 key:
4. When the initial 10 records are displayed on map, throw an error message ‘ALREADY AT THE BEGINING’.
5. Suppose, when records (21-30) are displayed on map, then subtract 19 from the last TSQ item and start moving the data from TSQ (11th item) to map.
6. User presses F8 key:
7. When the last 10 records are displayed on map, throw an error message ‘ALREADY AT1 THE BOTTOM’.
8. Suppose, when records (21-30) are displayed on map, then add1 to the task TSQ-ITEM (30) and start moving the dat from TSQ (31st item) to map.

Compilation of DB2:

1. Translation.
2. Compilation.
3. LINK\_EDIT.
4. Execution.

Compilation of COBOL-DB2-CICS

1. Pre
2. Translation.
3. Bind.
4. Compilation.
5. LINK EDIT.