**JCL**

* Job control language.
* JCL is used to **submit jobs** to the system.
* A job is one unit of work.
* There are 3 types of JCL statements:
  + **JOB:** specifies the information of the job, JOBCARD.
  + **EXEC:** specifies the program to use, tell what job will do SORT, COPY PS files etc..
  + **DD:** specifies the files to use in the job.
* There are 2 types of JCL parameters:
  + **Positional:** should be coded in specific order.
  + **Keyword:** can appear in any order.

**JCL syntax:**

* JCL must be written in a dataset of LREC 80.
* JCL must be written in uppercase.
* Syntax of a JCL is:
  + **//NAMEFIELD OPERAND PARAMETERS**
  + **Namefield** is maximum of 8 characters.
  + **Operand** can be JOB, EXEC, DD.
  + **Parameters** can be positional or keyword and is separated by commas.
* // is the start of every line called Identifier.
* //\* is used to give comment.
* /\* is the end of JCL.

**STATEMENTS IN JCL**

**JOB statement (Job card):**

* JOB statement syntax

**//job\_name JOB accounting\_info, programmer\_name,**

**// CLASS=B,**

**// PRTY=10,**

**// NOTIFY=user\_id,**

**// MSGCLASS=A OR 1,**

**// MSGLEVEL=(0/1/2,0/1/2),**

**// TYPRUN=SCAN/HOLD,**

**// TIME=(mm,ss),**

**// REGION=nK or nM,**

* Jobname must be 1-8 characters long.
* Jobname must be coded from position 3.
* First character of jobname must be either alphabetic or national character.
* **Positional Parameter:**
  + **Accounting Info:**
    - It refers to person or a group to which the CPU time is owed.
    - It is system specific.
    - It is not mandatory.
    - If absent then it should be replaced with comma.
  + **Programmer Name:**
    - The person or group who owns this JCL.
    - It is not mandatory.
* **Keyword Parameter:**
  + **CLASS:** 
    - Jobs similar to some aspect are grouped in one class.
    - 0 to 9 | A to Z.
  + **PRTY:** 
    - Specifies the priority of the job within a job.
    - 0 to 15 | 15 is max priority.
  + **TIME:** 
    - Maximum time allocated for the job execution.
    - (mm,ss)
  + **MSGCLASS:** 
    - Specifies the output destination for the job messages.
    - 0 to 9 | A to Z.
    - Ex: MSGCLASS=P means print the job output, A means fax the job output, E means email.
  + **MSGLEVEL:** 
    - Specifies the type of messages to be written to the output.
    - MSGLEVEL=(STATEMENT, MSG)
    - STATEMENT = 0 job statement only, 1 JCL along with symbolic parameters expanded, 2 JCL only.
    - MSG = 0 messages written only on abnormal termination, 1 messages written irrespective of normal/abnormal termination.
  + **TYPRUN:** 
    - Specifies a special processing of the job.
    - TYPRUN=HOLD | SCAN
    - HOLD puts the job on hold in the job queue, and can be later executed if required.
    - SCAN checks the syntax errors of the JCL without executing.
  + **NOTIFY:** 
    - Specifies the user to be notified on the completion of the JOB.
  + **REGION:**
    - Specifies the amount of address space to be allocated for job execution.

**EXEC statement:**

* The EXEC statement is used to invoke the program/procedure that you want to execute.
* It tells the system, what program/procedure you want to execute.
* One JCL program can have multiple EXEC statements.
* Syntax of EXEC statement is:
  + **//step\_name EXEC parameters**
  + **Step\_name**  must be 1-8 characters long.
  + **Step\_name** must be coded from position 3.
* **Positional Paramters:**
  + **PGM:** 
    - It refers to the program name to be executed in the job step.
  + **PROC:** 
    - It refers to the procedure name to be executed in the job step.
  + A procedure refers to a set of JCL statements that are in the library and retrieved by its name.
  + A procedure may contain one or more EXEC statements.
* **Keyword Parameters:**
  + **PARM:**
    - Used to pass values to the program being executed in the step.
  + **ACCT:** 
    - Specifies the accounting information of the job step.
    - ACCT=user\_id
  + **TIME:** 
    - Max time allocated for the step execution.
    - TIME=(mm,ss) | ss
  + **REGION:**
    - Specifies the amount of address space to be allocated for job execution.
    - REGION=nK | nM

**//step\_name EXEC PGM=SORT,**

**// ACCT=user\_id,**

**// PARM=’ADD 122 322’,**

**// TIME=(15,20),**

**// REGION=10M**

**DD statement:**

* DD stands for Dataset Definition.
* DD statement defines all the datasets to be used in the JCL.
* Syntax of DD statement is:
  + **//DD\_name DD Parameters**
* DD\_name must be 1-8 characters long.
* DD\_name must be coded from position3.
* **Positional Parameters:**
  + **\*:**
    - The \* begins an in-stream dataset.
  + **DUMMY:**
    - Used for testing of a JCL
    - Tells the system to not use any input/output for this DD statement
  + **DATA:**
    - In-stream dataset containing statements, which begins with //
* **Keyword Parameters:**
  + **DSN:**
    - Dataset name (DSN) is used to give the name of the dataset.
    - **DSN=NUHID.XXX.XXX**
  + **DISP:**
    - Disposition (DISP) describes the status of the dataset, what to do with the dataset on normal/abnormal end of the job.
    - **DISP=(status, normal end-disposition, abnormal end-disposition)**
    - Status can be:
      * **NEW:** 
        + the dataset will be created by this DD statement
      * **OLD:**
        + the dataset already exists, existing data will be overwritten and the jobstep will have exclusive access until completion of jobstep.
      * **SHR:**
        + The dataset already exists, and will be read, multiple jobs can access it at the same time.
      * **MOD:**
        + The dataset already exists, if not then new dataset will be created. Data will be added at the bottom of the existing data.
    - **Normal end disposition:**
      * Disposition can be: **CATLG, UNCATLG, DELETE, KEEP, PASS**
    - **Abnormal end disposition:**
      * Disposition can be: **CATLG, UNCATLG, DELETE, KEEP**
    - **CATLG:**
      * The dataset is retained with an entry in the system catalog.
      * For example, I’m creating a new dataset, it’ll create a new entry **in system catalog.**
      * **System catalog** is a dataset in which we store information about of all datasets of the system.
    - **UNCATLG:**
      * The dataset is retained but the system catalog entry is removed.
    - **KEEP:**
      * The dataset is retained without changing any of the catalog entries.
    - **DELETE:**
      * Dataset is deleted from the user and system catalog.
    - **PASS**
      * Used for only normal end-disposition, this is used when the dataset is to be passed and processed by the next job step in a JCL.
    - Default value of DISP:
      * **DISP=(NEW,CATLG,DELETE)**
  + **DCB:**
    - Data Control Block (DCB) specifies the values for creating a new dataset.
    - It contains various parameters such as:
      * **LRECL:** record length of the dataset
      * **RECFM:** record format of the dataset such as FB, V, VB
      * **BLKSIZE:** blocksize of the dataset
      * **DSORG:** dataset organization
  + **SPACE:**
    - Specifies the space required for the dataset.
    - **SPACE=(space\_units, (primary, secondary, directory-blocks), RLSE)**
    - Space\_unit can be KB, MB, Bytes, Blocks, Tracks, Cylinder
    - RLSE stands for release the empty storage
  + **UNIT:**
    - Specifies what type of storage to use for the dataset.
    - **UNIT=DASD | SYSDA**
    - **DASD (Direct Access Storage Device)** will store the dataset in a Direct Access Storage Device which is like a tape storage, HDD, SSD.
    - **SYSDA (System Direct Access)** will store it in next available disk storage device. Denotes that the location of the dataset is in DASD.
  + **VOL:**
    - Specifies the volume number on the device where the dataset needs to be stored
    - **VOL=SER=(v1,v2)**
    - v1, v2 are volume serial numbers.
  + **SYSOUT:**
    - Specifies the output destination based on the class.
    - **SYSOUT=class**
    - When the class is A then the output is through the printer
    - When the class is \* then the output goes to the same destination as the MSGCLASS

**//dd\_name DD DSN=NUHID.XXX.XXX,**

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

**// UNIT=SYSDA,**

**// VOL=SER=3390,**

**// SPACE=(TRK,(100,50,10),RLSE),**

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

**//SYSOUT DD SYSOUT=\***

**Difference between SYSOUT and SYSPRINT:**

* **If your program is not containing any files, SYSOUT in cobol contains output to your program and SYSOUT is to get messages regarding your program execution.**
* **This is same in JCL as well as COBOL.**
* **SYSOUT prints the program output and SYSPRINT prints the messages generated while program execution.**

**EXECUTION OF JCL PROGRAM**

**SORT JOB:**

**//SORTFILE JOB ANUBHAV,**

**// CLASS=A,**

**// MSGCLASS=A, /\*SPOOL AREA OUTPUT WILL BE PRINTED\*/**

**// MSGLEVEL=(2,1),**

**// NOTIFY=&SYSUID**

**//\***

**//STEP001 EXEC PGM=SORT**

**//\***

**//SORTIN DD DSN=NUHID.XXX.XXX,**

**// DISP=SHR**

**//SORTOUT DD DSN=NUHID.XXX.YYY,**

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

**// DCB=(REFCM=FB,LRECL=80,BLKSIZE=800,DSORG=PS)**

**// SPACE=(TRK,(100,50),RLSE)**

**//SYSOUT DD SYSOUT=\***

**//SYSIN DD \***

**SORT FIELDS=(1,8,CH,A)**

**/\***

**//\***

**Job Compilation Processing:**

**JCL Submission -> Job Entry subsystem (JES)**

**| (Input Data)**

**JCL Interpretation in JES -> Job Queuing -> Job execution and log creation in spool**

**(Output Data to printer/dataset/spool)**

* JCL is **converted into an interpreted text** to be understood by JES and **stored into a dataset**, which is called as **SPOOL**.
* **JES** decides the priority of the job based on the CLASS and PRTY parameters.
* The JCL errors are checked and the job is scheduled into the job queue if there are no errors.
* **JES (Job Entry Subsystem)** is used to:
  + Submit the job to the system.
  + Schedule the processing of the job.
  + Control the output of the jobs.
* **SPOOL (Simultaneous Peripheral Operations Online)** is a dataset which stores the:
  + Interpreted text for the JCL execution.
  + **JESMSGLG**: Messages related to job execution, such as job log, how much time does job took for execution, what return code each step executed etc..
  + **JESJCL**: we can see the JCL of the program
  + **JESYSMSG**: details message of the job step by step
  + **SYSPRINT**: information related to the step execution
  + **SYSOUT:** Output data from the job execution, such as errors.
* SPOOL information can be viewed using **SDSF (System Display Search Facility).**
* JCL is submitted to JES -> JES interprets it in understandable format.

**Return codes:**

* When a job executed a return code is set based on the status of execution.
* Return code can vary from **0 to 4095**.
* The most common RC are:
  + 0 successful execution
  + 4 successful execution with warnings
  + 8 errors
  + 12 severe error
  + 16 major error
* **MAXCC** is the maximum of the return code from all of the job steps.

**SDSF Subsystem:**

* **System Display and Search Facility (SDSF)** is used to review the job output. It’s primary purpose is to display the output in the spool area.
* SDSF can be used to:
  + View and search the system log
  + Hold, release, cancel and purge the jobs
  + Monitor the jobs while they are running/processed
  + Display job output
  + Control the order in which jobs are processed
* SDSF can be used to save thee data to a dataset for backup.
* Commands in SDSF are:
  + Enter **TSO SDSF** in the command line to got to SDSF
  + Then enter **ST** in **SDSF** command line to see the jobs. ST means Status of jobs.
    - **?:** list the datasets
    - **S job\_name:** seach jobname
    - **OWNER user\_id:** show jobs owned by user\_id
    - **OWNER \*:** show all the jobs of all owners
    - **F job\_name:** find and locate to jobname, but all the jobs will also be displayed
    - **PRE job\_name:** prefix of jobname
    - **P:** purge the dataset
    - **C:** cancel the job
    - **XDC:** store the output to a dataset

**Instream Procedures:**

* Creating a procedure:

**//proc\_name PROC /\*Define a procedure\*/**

**//STEP001 EXEC PGM=IEFBR14**

**//DD1 DD DSN=&DNAME, /\*SYMBOLIC PARAMTER\*/**

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

**// SPACE=(TRK,(100,50),RLSE),**

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

**// PEND /\*To end the procedure\*/**

**//\***

**//STEP2 EXEC proc\_name, DNAME=NUHID.XXX**

**//STEP3 EXEC proc\_name, DNAME=NUHID.YYY**

**//STEP4 EXEC proc\_name, DNAME=NUHID.ZZZ**

**/\***

* The procedure and calling step of the procedure is in the same JCL.
* Instream procedure can only be used in same JCL.
* Procedure is a way of reducing our code, it is given to perform a certain function, which is repeated again and again. So, instead repeating the code we create procedure.

**Symbolic Parameters:**

* Symbolic parameters are used to pass value to procedures.
* It starts with ‘&’ and end with ‘.’
* ‘.’ Can be ignored in case there is no other parameter after it.
* Value of symbolic parameter are passed while calling the procedure.
* Reserved keywords are not allowed as symbolic parameter names.
* Examples of symbolic paramters:
  + &DNAME

**Cataloged Procedures:**

* Cataloged procedure is created in another location not in same JCL.
* Cataloged procedure doesn’t require a jobcard, it just needs procedure body.

**//JOBCARD**

**//\***

**//PROCLOC JCLLIB ORDER= NUHID.XXX.XXX**

**//\***

**//STEP002 EXEC proc\_name,MNAME=NEW**

**/\***

**Inside NUHID.XXX.XXX there is a member called proc\_name, inside that procedure is created.**

**//proc\_name PROC**

**//STEP001 EXEC PGM=IEFBR14**

**//DD1 DD DSN=NUHID.&MNAME..PS1,**

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

**// SPACE=(TRK,(100,50),RLSE),**

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

**// PEND**

**LIBRARIES IN JCL**

**JOBLIB:**

* Default library of the system is **SYS1.LINKLIB**, where job will search for the program by default.
* JOBLIB tells the location of the load module.
* Syntax:
  + **//JOBLIB DD DSN=NUHID.XXX.XXX**

**// DD DSN=NUHID.YYY.YYY**

* More than one dataset can be mentioned in JOBLIB location.
* If JOBLIB is provided then the load module is first searched in the PDS mentioned in the JOBLIB, if not found then the load modules is searched in SYS1.LINKLIB.
* If load module is not found in both JOBLIB and default lib then an abend **S806** will occur.
* JOBLIB has to be given just after the JOBCARD/JOB statement, before any EXEC statement.
* JOBLIB is applicable to all steps in the JCL.

**STEPLIB:**

* STEPLIB is used to tell the location of the load module for one particular step
* Syntax:
  + **//STEPLIB DD DSN=NUHID.XXX.XXX**

**// DD DSN=NUHID.YYY.YYY**

* STEPLIB is given exactly after the EXEC statement before any DD statement in the step.
* More than one dataset can be mentioned in STEPLIB location.
* If both JOBLIB and STEPLIB are present then JOBLIB will be ignored.
* If load module is not found in STEPLIB then it is searched in the default library SYS1.LINKLIB.
* STEPLIB makes the searching of the load module faster.

**JCLLIB:**

* It is used to tell the location of JCL code
* Syntax:
  + **//PROCLOC JCLLIB ORDER= (NUHID.XXX.XXX, NUHID.YYY.YYY)**
* JCLLIB is for JCL code like cataloged procedure, but STEPLIB and JOBLIB are for executable code i.e. load modules.
* Load module is an executable code, that means you can run this load module to run your program.
* JCL code is readable code. Load modules is in special characters.

**CONDITIONAL PROCESSING IN JCL**

**COND Parameter:**

* **COND=(RC,OPERATOR,STEPNAME)**
* RC can be from 0 to 4095.
* Operator can be:
  + GT Greater than
  + LT Less than
  + EQ Equal
  + NE Not equal
  + GE Greater than or equal
  + LE Less than or equal
* COND can be given in a JOB statement or an EXEC statement.
* COND code does not work on the first step in a JCL.
* If the COND is true then bypass the step, if COND is false then execute the step.
* COND parameter coded on JOB card gets evaluated first and if it is satisfied, the job gets terminated. Otherwise the step level parameter gets evaluated.
* Syntax:
  + **//step\_name EXEC PGM=SORT, COND=(0,EQ,step\_name2)**
  + **//job\_name JOB job\_parameters,COND=(4,LT)**
* **If you want to never execute a step:**
  + **COND=(0,LE) | (4095,GT)**
* **If you want to execute always execute:**
  + **COND=(4095,LT)**
* **COND=ONLY**, if this condition is there in any step, then that step will only execute when previous to that any step got abended. (Only when abend)
* **COND=EVEN,** if this condition is there in any step, then that means even if any of the previous step has abended execute the step. (Execute even if abend occurred)

**IF-ELSE:**

* IF-ELSE work on first step of the JCL as well.
* Syntax IF-ENDIF:

**//STEP1 EXEC PGM=SORT**

**//…**

**//CHECK1 IF RC=0 THEN**

**//STEP2 EXEC PGM=SORT, COND=(4,LT)**

**//…**

**// ENDIF**

**//STEP3 EXEC PGM=SORT**

**//..**

* Syntax IF-ELSE:

**//STEP1 EXEC PGM=SORT**

**//…**

**//CHECK1 IF STEP1.RC=8 THEN**

**//STEP2 EXEC PGM=SORT**

**//…**

**//ELSE1 ELSE**

**//STEP3 EXEC PGM=SORT**

**//..**

**// ENDIF**

**//STEP4 EXEC PGM=SORT, COND=(4,LT)**

**GENERATION DATA GROUPS**

**GDG:**

* **GDG** is a group of files related to each other and having a common name.
* **NUHID.XXX.YYY.G0001V00 -> NUHID.XXX.YYY.G0002V00 …so on**
* **G0001** stands for generation number, **V00** stands for version number.
* To create a GDG, a dataset called GDG base is first created. All the generations are then modeled from this base.
* Max number of GDG generations can be 255.
* Utility called **IDCAMS** is used to define a GDG.
* GDG Parameters:
  + **LIMIT:** Max number of generations that a GDG can hold.
  + **EMPTY:** Uncatalog all generation once limit is reaches.
  + **NOEMPTY:** Uncatalog only the oldest generation once limit is reached.
  + **SCRATCH:** Delete the generation physically when uncataloged.
  + **NOSCRATCH:** Does not deletes the generation when uncataloged.

**Create a GDG:**

**//JOBCARD**

**//\***

**//STEP001 EXEC PGM=IDCAMS**

**//SYSPRINT DD SYSOUT=\***

**//SYSIN DD \***

**DEFINE GDG(NAME(NUHID.XXX.XXX) -**

**LIMIT(12) -**

**NOEMPTY -**

**SCRATCH)**

**/\***

**//\***

**//STEP002 EXEC PGM=IEFBR14**

**//DD1 DD DSN=NUHID.XXX.XXX(+1),**

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

**// SPACE=(TRK,(100,50),RLSE),**

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

**//SYSPRINT DD SYSOUT=\***

**//\***

**Delete the GDG Generation:**

**//STEP002 EXEC PGM=IEFBR14**

**//DD1 DD DSN=NUHID.XXX.XXX(0 | -1 | -2,..),**

**// DISP=(OLD,DELETE,DELETE),**

**//SYSPRINT DD SYSOUT=\***

**//\***

**Delete the GDG Base:**

**//JOBCARD**

**//\***

**//STEP002 EXEC PGM=IDCAMS**

**//SYSIN DD \***

**DELETE(NUHID.XXX.XXX) GDG PURGE**

**//SYSPRINT DD SYSOUT=\***

**//\***

* **FORCE:** it’ll only delete if the retention period has expired.
* **PURGE:** it’ll delete even if the retention period of the GDG has not expired.

**REXX**

* **REXX** is a scripting language.
* It is mainly used for automation on mainframe.
* Let suppose u have created rexx program inside a pds member, so to execute it give ‘EX’ command infront of pds member and rexx code will be executed.
* Syntax:

**/\* REXX \*/**

**say ‘Hello World’**

**exit**

**VSAM (VIRTUAL STORAGE ACCESS METHOD)**

**What is VSAM:**

* VSAM is a data access method on mainframe.
* **What is an access method?** An access method defines the technique that is used to **store and retrieve data.**
* An access method has:
  + Its own structure to organize data -> Data and Index component. Data is organized in control intervals.
  + System-provided programs (or macros) to define datasets. -> IDCAMS
  + And utility programs to process -> IDCAMS
* There are other access methods on mainframes such as QSAM. PS files are created and stored using **QSAM**.
* Access Method for PS file is **QSAM**.
* Access Method for VSAM file is **CI**.
* **QSAM** stores data as Blocks. A block is a collection of records. In every read whole block is copied into the memory.
* The equivalent of this in VSAM is **Control Interval.**
* **VSAM stores and retrieves data from the disk as CI, when the program read the VSAM file then in every read one CI is read from the disk and put into the memory.**
* **Block in QSAM:**

|  |  |  |
| --- | --- | --- |
| **Record** | **Record** | **Record** |

* **Control Interval (CI) in VSAM:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Record** | **Record** | **Record** | **Free Space** | **RDF (3 Bytes)** | **CIDF (4 Bytes)** |

* + **Free Space:** When we enter record into the VSAM file the already existing record move into the free space.
  + **RDF:** Record Descriptor Field, it describes the records. There is a RDF for each record which tells what is the length of this record.
  + **CIDF:** Control Interval Definition Field, it gives the info about the whole CI.
* VSAM can organize data into 4 types of datasets:
  + **KSDS** 
    - Record is retrieved and inserted using a key value.
    - KSDS has 2 components: Data and Index
  + **RRDS**
    - The record can be accessed using record number.
  + **ESDS**
    - The records are retrieved in the sequence they are added.
  + **LDS**
    - The data is stored in byte-streams, just like notepad.
* IDCAMS utility is used to define VSAM datasets.
* VSAM is also called clusters.

**Control Interval, Control Area and VSAM cluster:**

* VSAM is made up of Control Intervals.
* CI is the smallest unit of transfer b/w disk and operating system. In every read one CI is read from disk and put into the memory.
* CI contains:
  + **Records**
    - Data of VSAM file
  + **Freespace**
    - It is used for inserting a record in VSAM
  + **CI Fields:**
    - **RDF**
      * 3 Bytes long
      * It describes the length of records.
    - **CIDF**
      * 4 Bytes long
      * It contains the information about the CI
      * Default size of CIDF is 4096 bytes, can be up to max 32KB.

**TYPES OF VSAM DATASETS**

**KSDS (KEY SEQUENCED DATASET):**

* Records are stored in ascending sequence by key.
* Records are always sorted by the key.
* Keys are always unique
* No duplicate records are possible in KSDS.
* KSDS has both Data and Index component.
* It can access data by **Direct Access as well as Sequential Access**.
* We give a key to the index and it will find the corresponding data record in the Control Area.
* Parameters while creating a KSDS;
  + **NAME:** Gives the name of the VSAM
  + **KEYS: KEY(length offset) -> KEY(8 0)**
    - Length is length of key and offset is position from which key is starting
    - maximum key length 255
  + **RECSZ: RECSZ(average maximum) -> RECSZ(90 200)**
    - LREC of the VSAM file
  + **FREESPACE: FREESPACE(%CI %CA)**
    - Amount of space which will be kept free in the control interval and control area
  + **INDEXED:**
    - It tells the system that the VSAM is KSDS.
    - **NUMBERED for RRDS**
    - **NONINDEXED for ESDS**
    - **LINEAR for LDS**
  + **CISZ:**
    - Control Interval Size
    - Default is 4096 bytes. Can be maximum of 32KB.
  + **SAPCE:**
    - The amount of total space allocated to VSAM.
    - **UNIT(PRIMARY SECONDARY) -> TRACKS(100 50)**
  + **VOLUME:**
    - The volume serial where VSAM will reside is given here.

**//JOBCARD**

**//\***

**//STEP01 EXEC PGM=IDCAMS**

**//SYSPRINT DD SYSOUT=\***

**//SYSIN DD \***

**DEFINE CLUSTER( -**

**NAME(NUHID.XXX.XXXX) -**

**INDEXED -**

**KEYS(5 0) -**

**RECSZ(90 200) -**

**FREESPACE(10 20) -**

**TRACKS(50 30) -**

**CISZ(8192) -**

**VOLUME(ZASYS1)) -**

**/\***

**ESDS (ENTRY SEQUENCED DATASET):**

* Records are stored in the order they are added, i.e. records are stored and retrieved sequentially.
* New records are added **at the end** of the dataset.
* Records **cannot be deleted** in an ESDS. It’ll be set as inactive if deleted but data we will able to see.
* There is **no index component, so duplicate records** are possible in ESDS.
* Sequential access and Random access is possible. Random access is done using **the RBA (Relative Byte Address).**
* Parameters required while creating a ESDS are:
  + **NAME**
  + **RECSZ**
  + **FREESPACE**
  + **NONINDEXED**
  + **CISZ**
  + **SPACE**
  + **VOLUME**

**//JOBCARD**

**//\***

**//STEP001 EXEC PGM=IDCAMS**

**//SYSPRINT DD SYSOUT=\***

**//SYSIN DD \***

**DEFINE CLUSTER( -**

**NAME(NUHID.XXX.XXX) -**

**NONINDEXED -**

**RECSZ(100 50) -**

**FREESPACE(10 20) -**

**CISZ(8192) -**

**TRACKS(100 500 -**

**VOLUME(\* \*)) -**

**/\***

**RRDS (RELATIVE RECORD DATASET):**

* RRDS has records that re identified by their Relative Record Number (RRN). The first record in the dataset is RRN 1, the second record is RRN 2 and so on.
* Records are stored in record slots. New records are added at an empty slot.
* Records can be deleted in RRDS, deleting creates an empty slot.
* Sequential access and Random Access is possible. Random access is done using the RRN.
* Parameters required to create RRDS are:
  + **NAME**
  + **RECSZ**
  + **FREESPACE**
  + **NUMBERED**
  + **CISZ**
  + **SPACE**
  + **VOLUME**

**//JOBCARD**

**//\***

**//STEP01 EXEC PGM=IDCAMS**

**//SYSPRINT DD SYSOUT=\***

**//SYSOUT DD SYSOUT=\***

**//SYSIN DD \***

**DEFINE CLUSTER( -**

**NAME(NUHID.XXX.XXX) -**

**NUMBERED -**

**RECSZ(90 200) -**

**CISZ(8192) -**

**TRACKS(100 50) -**

**VOLUME(\* \*) -**

**FREESPACE(10 10)) -**

**/\***

**LDS (LINEAR DATASET):**

* It has byte stream data like notepad.
* There are no records, there is only data bytes.
* Parameters required to create LDS are:
  + **NAME**
  + **LINEAR**
    - It tells system that the VSAM is LDS
  + **SPACE**
  + **CISZ**
  + **VOLUME**

**//JOBCARD**

**//\***

**//STEP01 EXEC PGM=IDCAMS**

**//SYSPRINT DD SYSOUT=\***

**//SYSOUT DD SYSOUT=\***

**//SYSIN DD \***

**DEFINE CLUSTER( -**

**NAME(NUHID.XXX.XXX) -**

**LINEAR -**

**CISZ(4096) -**

**TRACKS(100 50) -**

**VOLUME(\* \*))**

**/\***

**PROCESSING VSAM DATASETS**

**DITTO Utility:**

* Used to edit the data inside VSAM.
* In our project we use File manager to edit the data inside VSAM.
* How to use DITTO utility:
  + M -> 7 (DITTO untility) -> 2 (edit the data) / 1 (browse the data) -> 1 (edit the VSAM) -> give VSAM file name

**LISTCAT Command:**

* It is used to get the information of VSAM dataset.
* Give ‘I’ Infront of VSAM file in 3.4 option
* LISTCAT program:

**//JOBCARD**

**//\***

**//STEP01 EXEC PGM=IDCAMS**

**//SYSPRINT DD SYSOUT=\***

**//SYSIN DD \***

**LISTCAT ENTRY(NUHID.XXX.XXX) -**

**ALL**

**/\***

**REPRO Command:**

* It is used to load data into VSAM dataset
* Used to copy data into the VSAM file
* **PS to VSAM | VSAM to VSAM | VSAM to PS**
* **PS to KSDS** -> Records should be in sorted order and keys should unique.
* **PS to VSAM job:**

**//JOBCARD**

**//\***

**//STEP01 EXEC PGM=IDCAMS**

**//PSFILE DD DSN=NUHID.XXX.PS,DISP=SHR**

**//VSAMFL DD DSN=NUHID.XXX.KSDS,DISP=SHR**

**//SYSIN DD \***

**REPRO INFILE(PSFILE) -**

**OUTFILE(VSAMFL)**

**/\***

* **VSAM to VSAM job:**

**//JOBCARD**

**//\***

**//STEP01 EXEC PGM=IDCAMS**

**//VSAM1 DD DSN=NUHID.XXX.KSDS,DISP=SHR**

**//VSAM2 DD DSN=NUHID.YYY.KSDS,DISP=SHR**

**//SYSIN DD \***

**REPRO INFILE(VSAM1) -**

**OUTFILE(VSAM2)**

**/\***

* **VSAM to PS job:**

**//JOBCARD**

**//\***

**//STEP01 EXEC PGM=IDCAMS**

**//VSAM1 DD DSN=NUHID.XXX.KSDS,DISP=SHR**

**//PSFILE DD DSN=NUHID.YYY.PS,DISP=SHR**

**//SYSIN DD \***

**REPRO INFILE(VSAM1) -**

**OUTFILE(PSFILE)**

**/\***

**ALTERNATE INDEX**

**What is Alternate Index (AIX):**

* It is used to access the VSAM dataset apart from primary key.
* Alternate Index creates a another key in the VSAM dataset.
* We can use in ESDS and KSDS.
* Let suppose we have unique id, name and phone number in a dataset, we want to access the dataset using phone number.
* Steps in creating Alternate Index:
  + **Define the AIX**
  + **Define path for AIX**
    - Used for accessing the AIX
  + **Build AIX**
    - Used to put the data inside your AIX
* **UNIQUEKEY**: It means the new key that you are creating having unique values, if its not unique values give **NONUNIQUEKEY**
* **UPGRADE**: It means if we update our base KSDS so will our AIX dataset will be updated. If not updated then give **NONUPGRADE**.
* **AIX** is always browsed using the path.

**//JOBCARD**

**//\***

**//STEP01 EXEC PGM=IDCAMS /\* DEFINING THE AIX \*/**

**//SYSPRINT DD SYSOUT=\***

**//SYSIN DD \***

**DEFINE AIX( -**

**NAME(NUHID.XXX.XXX.AIX) -**

**RELATE(NUHID.XXX.KSDS) -**

**KEYS(10 20) -**

**RECSZ(90 200) -**

**FREESPACE(10 20) -**

**TRACKS(50 30) -**

**CISZ(8192) -**

**VOLUME(ZASYS1) -**

**UNIQUEKEY -**

**UPGRADE**

**/\***

**//STEP02 EXEC PGM=IDCAMS /\* DEFINING PATH FOR AIX \*/**

**//SYSPRINT DD SYSOUT=\***

**//SYSOUT DD SYSOUT=\***

**//SYSIN DD \***

**DEFINE PATH(NAME(NUHID.XXX.XXX.AIX.PATH) -**

**PATHENTRY(NUHID.XXX.XXX.AIX)**

**/\***

**//STEP03 EXEC PGM=IDCAMS /\*BUILD AIX\*/**

**//SYSPRINT DD SYSOUT=\***

**//SYSOUT DD SYSOUT=\***

**//SYSIN DD \***

**BUILDINDEX -**

**INDATASET(NUHID.XXX.XXX.KSDS) -**

**OUTDATASET(NUHID.XXX.XXX.AIX)**

**/\***