Mainframe Technology Training in DXC Technology

Project report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology in

BIOTECHNOLOGY

May 24,2021

Submitted By:

JIGYASA (171821)

Project Work Completed Under the guidance at

DXC Technologies Pvt. Ltd.

JAYPEE UNIVERSITY OF INFORMATION AND TECHNOLOGY

DEPARTMENT OF BIOTECHNOLOGY AND BIOINFORMATICS

WAKNAGHAT, H.P - 173234

Project Report Undertaking

I Ms. Jigyasa Roll No. 171821 Branch Biotechnology is doing my internship with DXC technology from to

As per procedure I have to submit my project report to the university related to my work that I have done during this internship.

I have compiled my project report. But due to COVID-19 situation my project mentor in the company is not able to sign my project report.

So I hereby declare that the project report is fully designed/developed by me and no part of the work is borrowed or purchased from any agency. And I'll produce a certificate/document of my internship completion with the company to TnP Cell whenever COVID-19 situation gets normal.

Signature : [illegible]

Name : Jigyasa

Roll No. : 171821

Date : 24th March 2021

DECLARATION

I hereby declare that the presented report entitled " Mainframe Technology Training in DXC technology" in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Biotechnology submitted in the Department of Biotechnology and Bioinformatics, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from February 2021 to May 2021 under the supervision of Dr. Udayabanu, M., Associate Professor, Jaypee University Of Information Technology. The matter embodied in the report has not been submitted for the award of any other degree or diploma.

[illegible]

(Student Sign)

Jigyasa, 171821

[illegible]

(Supervisor Signature)

Dr. Udayabanu, M,

Associate Professor,

Jaypee University Of Information Technology

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION 8

1.1 INTRODUCTION 8

1.2 ABOUT COMPANY 8

CHAPTER 2: SOFTWARE DEVELOPMENT 10

2.1 INTRODUCTION 10

2.2SDLC Phases 10

CHAPTER 3: MAINFRAME TECHNOLOGY 13

3.1 INTRODUCTION 13

3.2 CHARACTERISTICS 13

3.3 MAINFRAME OPERATING SYSTEM 13

3.4 BASIC FILE MANAGEMENT OPERATIONS 14

CHAPTER 4 : JOB CONTROL LANGUAGE (JCL) 16

4.1 INTRODUCTION 16

4.2 JOB PROCESSING 16

4.3 JCL STRUCTURE 17

4.4 JOB STATEMENT 17

4.5 EXEC STATEMENT 18

4.6 DD STATEMENT 18

4.7 JCL UTILITIES 18

CHAPTER 5 : VIRTUAL STORAGE ACCESS METHOD (VSAM) 21

5.1 INTRODUCTION 21

5.2 CHARATERISTICS 21

5.3 IDCAMS 21

5.4 VSAM DATA SETS 22

5.5 ALTERNATE INDEX (AI) 24

5.6 CATALOG 25

CHAPTER 6 : COBOL LANGUAGE 26

6.1 INTRODUCTION 26

6.2 COBOL HISTORY 26

6.3 IMPORTANCE 26

6.4 FEATURES 27

6.5 COBOL STRUCTURE 27

6.6 DIVISIONS 28

6.7 DATA TYPES 29

CHAPTER 7: DB2/SQL 34

7.1 INTRODUCTION 34

7.2 TABLES 35

7.3 EMBEDDED SQL PROGRAMMING 38

CONCLUSION 45

REFERENCES 46

List of Figures

Fig 1 1 Enterprise Technology Stack 9

Fig 2 1 Software Development Life Cycle (SDLC) 11

Fig 3 1 Time Sharing Option 15
Fig 3 2 Interactive System Productivity Facility 15

Fig 4 1 Job Processing 16
Fig 4 2 Code Using IEFBR14 19
Fig 4 3 Code Using IEBGENER Utility 19
Fig 4 4 Code Using Sort Utility 20

Fig 5 1 IDCAMS Utility to create a Cluster in VSAM 22
Fig 5 2 Types Of Catalog in VSAM 25

Fig 6 1 COBOL Structure 27
Fig 6 2 Data Types in COBOL 29
Fig 6 3 COBOL code using loops and conditional statements 30

Fig 7 1 Database Architecture 34
Fig 7 2 Code to create a table 36
Fig 7 3 Code to create an Index 37
Fig 7 4 Code to insert values 37
Fig 7 5 Code to select 38
Fig 7 6 COBOL/DB2 Code 41

List of Tables

Table 5 1 KSDS Components..23

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION

I got placed at DXC technology and has successfully completed DXC Early Career Professionals program (DECP). The whole program was comprised of two trainings – Foundation and Technical. I have been assigned to Application Development Learning Track during my foundation training where I have learned about the computer science fundamentals such as Software Development Processes, Data Structures, Programming Fundamentals, Database Management System (DBMS), Introduction to Cloud, and Big Data. After the successful completion of Foundation training I have been assigned the Mainframe Application Track in during my technical training. During the technical training I have learned various technologies like understanding Mainframe technology, Z/os concepts, TSO-ISPF, JCL, VSAM, DB2 and COBOL.

1.2 ABOUT COMPANY

DXC Technology, a Fortune 500 company, is an American multinational corporation that provides business-to-business information technology services. When Hewlett Packard Enterprise Company (HPE) broke off its Enterprise Services Business and combined it with Computer Sciences Corporation in 2017, DXC Technology was born. CSC and HPE Enterprise Services were recognised throughout their histories for developing to keep up with the ever-changing world of technology — and for providing clients with a new viewpoint based on a long tradition of innovation and industry-leading services. Mike Salvino was appointed president and CEO of DXC Technology in 2019. DXC Technology (NYSE: DXC) helps worldwide enterprises modernise IT, optimise data infrastructures, and ensure security and scalability across public, private, and hybrid clouds. Since April 3, 2017, DXC has traded on the New York Stock Exchange under the ticker "DXC." Their focus is on on-premises and cloud IT transformation, data-driven operations, and workplace modernisation. With the scope and size of services in the Enterprise Technology Stack, DXC is in a great position to assist clients in managing their IT estate.

In India, DXC Technology is a beehive of innovation and technological talent. Our worldwide operations are the company's largest, with high-caliber technology specialists dedicated to assisting clients with their core issues and market possibilities.

DXC Technology provides services to the various industries.

DXC's Corporate Responsibility programme strives to build a sense of community and togetherness within the company by actively participating in and donating to socially relevant projects. DXC Corporate

Responsibility (CR) is a top focus, and its employees go above and above in a variety of ways to help their local communities. Our workers donate their resources and time to a variety of community development activities both locally and globally.

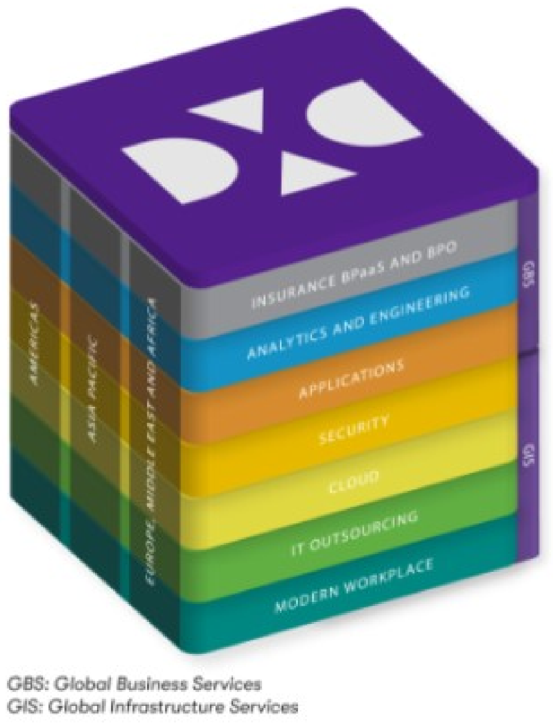

INSURANCE BPaaS AND BPO
ANALYTICS AND ENGINEERING
APPLICATIONS
SECURITY
CLOUD
IT OUTSOURCING
MODERN WORKPLACE
AMERICAS
ASIA PACIFIC
EUROPE, MIDDLE EAST AND AFRICA
GBS
GIS
GBS: Global Business Services
GIS: Global Infrastructure Services

Fig 1 1 Enterprise Technology Stack

CHAPTER 2: SOFTWARE DEVELOPMENT

2.1 INTRODUCTION

SDLC is for Software Development Life Cycle, and it is a method for developing software that ensures its quality and correctness. The SDLC process is designed to develop high-quality software that fulfils customer requirements. The system development should be completed within the schedule and budget constraints. SDLC is a step-by-step process that outlines how to plan, construct, and maintain software. Each phase of the SDLC life cycle includes its own set of processes and deliverables, which feed into the next. The Software Development Life Cycle (SDLC) is also known as the Application Development Life Cycle (ADLC).

The following are some of the most essential reasons why SDLC is crucial while designing a software system.

- It serves as a foundation for project planning, scheduling, and costing.
- Provides a framework of activities and deliverables that are all the same.
- It's a system for keeping track of and controlling projects.
- Increases the visibility of project planning to all development process stakeholders.
- Speed of development has been increased and improved.
- Client relationships have improved.

2.2SDLC Phases

The SDLC process is broken down into the following stages:

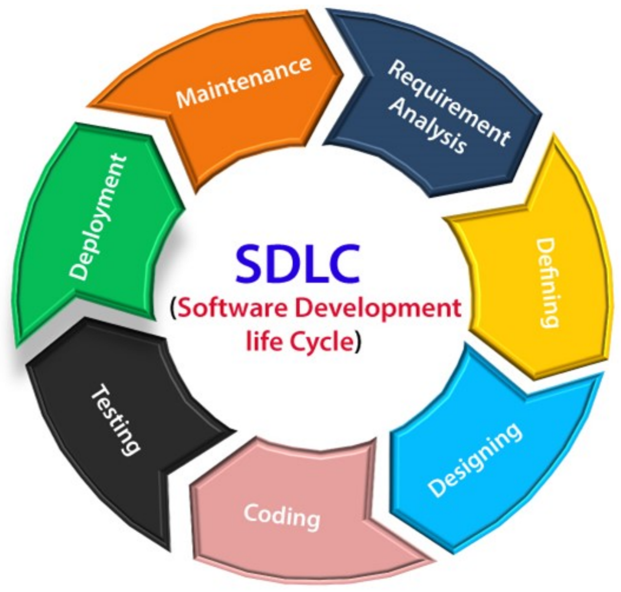


Fig 2 1 Software Development Life Cycle (SDLC)

2.2.1 STAGE – 1 : REQUIREMENT ANALYSIS

The first stage of the SDLC process is the requirement analysis. It is organised by senior team members with input from all industry stakeholders and domain specialists. At this point, the quality assurance requirements are planned for, as well as the hazards that are involved.

This stage provides a clearer understanding of the project's overall scope as well as the anticipated difficulties, opportunities, and instructions that prompted it.

The team must gather specific and accurate criteria throughout the gathering step. This aids businesses in determining the necessary timescale for the system's completion.

2.2.2 STAGE – 2: DEFINING FEASIBILITY

The next stage of the SDLC is to define and record software requirements after the requirement analysis phase is completed. This procedure was carried out using the 'Software Requirement Specification' document, commonly known as the 'SRS' document. During the project life cycle, it comprises everything that has to be created and produced.

There are five different kinds of feasibility checks:

Economic: Can we finish the project on time and on budget?

Legal: Can this project be handled under cyber law and other regulatory frameworks/compliances?

Operation: Are we able to design operations that the client expects?

Technical: Need to see if the software will run on the current computer system.

Schedule: Determine whether or not the project can be finished on time.

2.2.3 STAGE – 3: DESIGNING

The goal of this stage is to gather all of the requirements, analysis, and design information for the software project. This makes it easier to define the overall system architecture. This design phase serves as input to the model's subsequent phase.

2.2.4 STAGE – 4: CODING

Developers begin developing code in the selected programming language to create the full system at this phase. Tasks are separated into units which are further separated into modules and given to different developers throughout the coding phase. It is the most time-consuming step of the Software Development Life Cycle. During this phase, the developer must adhere to a set of predetermined code principles. They must also produce and implement code using programming tools such as compilers, interpreters, and debuggers.

2.2.5 STAGE – 5: TESTING

As testing operations are mainly included in all phases of SDLC in current SDLC models, this stage is frequently a subset of all phases. The testing team begins testing the full system's functioning. This is done to ensure that the entire application functions as expected by the client.

QA and testing teams may discover errorsat this phase, which they report to developers. The development team fixes the issue and sends it back to QA for another round of testing. This procedure is repeated until the programme is bug-free, reliable, and meets the system's business requirements.

2.2.6 STAGE – 6: DEPLOYMENT

When the software testing step is completed and the system is free of errors and faults, the final deployment procedure begins. Product rollout is sometimes done in stages, depending on the company's business plan. The product may then be released as is or with proposed enhancements in the intended market group depending on the feedback. After the programme has been deployed, it must be maintained.

2.2.7 STAGE – 7: MAINTENANCE

When the businesses begins to use the designed systems, genuine challenges and requirements arise that must be addressed on a regular basis. Maintenance is a practice that involves taking care of a created product.

CHAPTER 3: MAINFRAME TECHNOLOGY

3.1 INTRODUCTION

The different end-user application systems that are housed on mainframe computers are known as mainframe application systems. Because of their capacity to process enormous amounts of data more effectively as well as data security measures, mainframe computers and their services are crucial for governments and big companies. As a result, numerous firms use mainframes to host their applications in order to assure data security and smooth data processing. This describes what a mainframe system is and how it may help you.

3.2 CHARACTERISTICS

- Extremely fast
- Support for a huge number of people
- Take up space in the area
- Expensive

3.3 MAINFRAME OPERATING SYSTEM

In its most basic form, an operating system is a collection of programmes that control the internal workings of a computer system, including memory, processors, peripherals, and the file system. Mainframe operating systems are complex solutions with a wide range of features and functions.

3.3.1 FEATURES OF MAINFRAME OPERATING SYSTEM

- Multiprogramming refers to the capacity to run many programmes at the same time.
- When main memory is insufficient to hold huge amounts of data, virtual storage is used.
- Spooling is the process of copying the image of the file content to be printed to a spool file on the hard drive, which is subsequently printed later.
- Ability to accommodate several users at the same time (time sharing).
- Batch processing refers to the capacity to process huge amounts of data in a non-interactive

manner.

3.3.2 DIFFERENT MAINFRAME OPERATING SYSTEMS

z/OS: The z/OS operating system, which is commonly used on mainframes, is meant to provide a reliable, secure, and always-available environment for mainframe applications.

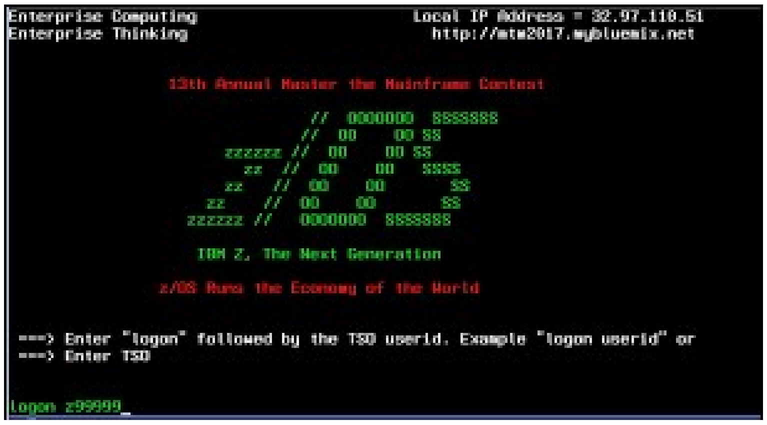
- z/VM: Because it runs other operating systems in the virtual machines it produces, z/Virtual Machine (z/VM) is a hypervisor as a control application.
- z/VSE: Users of smaller mainframe machines choose z/Virtual Storage Extended (z/VSE). When their needs beyond the capabilities of z/VSE, some of these clients transition to z/OS.
- Linux for System z: On a mainframe, a variety of (non-IBM) Linux distributions can be utilised.
- z/TPF: The z/Transaction Processing Facility (z/TPF) operating system is a special-purpose system used by firms that process a large number of transactions, such as credit card firms and airline reservation systems.

3.4 BASIC FILE MANAGEMENT OPERATIONS

3.4.1 TIME SHARING OPTION (TSO)

TSO is an interactive time-sharing environment for IBM mainframe operating systems such as MVS , OS/360 MVT, OS/390, OS/VS2 (SVS), and z/OS. It consists of a collection of commands that the user must put into the interface in order to complete the task.

Time-sharing is a design concept in computing that allows many users to utilise a computer system simultaneously and independently without interfering with one another. [1] Each TSO user is isolated; they believe they are the only ones using the system.

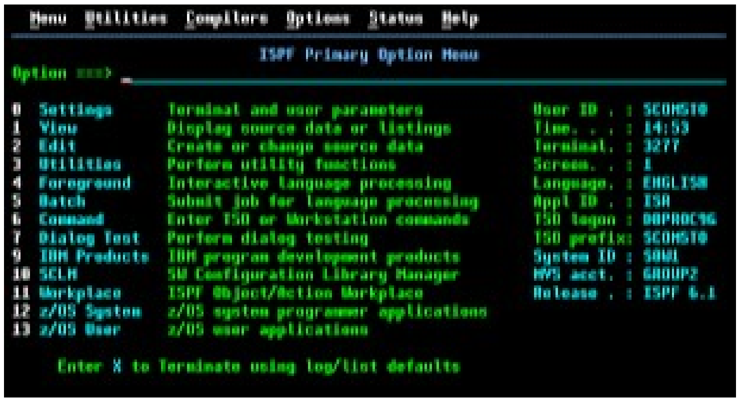


```
Enterprise Computing                          Local IP Address = 32.97.110.51
Enterprise Thinking                             http://mtm2017.mybluemix.net

                  13th Annual Master the Mainframe Contest

                                //  0000000  SSSSSSS
                               //  00    00 SS
                       zzzzzz //  00    00 SS
                          zz //  00    00  SSSS
                        zz  //  00    00      SS
                      zz   //  00    00      SS
                    zzzzzz //  0000000  SSSSSSS

                     IBM Z, The Next Generation

                 z/OS Runs the Economy of the World

 ===> Enter "logon" followed by the TSO userid. Example "logon userid" or
 ===> Enter TSO

logon z99999_
```

Fig 3 1 Time Sharing Option

3.4.2 INTERACTIVE SYSTEM PRODUCTIVITY FACILITY (ISPF)

ISPF is a menu-driven interface that allows the user to do operations by picking from a menu of alternatives. Because most mainframe software providers utilised ISPF functions to build their programmes, their tools look and operate similarly to ISPF. Many installations, likewise, create their own informal tools that make advantage of ISPF capabilities.



```
  Menu  Utilities  Compilers  Options  Status  Help
 ------------------------------------------------------------------------------
                          ISPF Primary Option Menu
 Option ===> _
 
 0  Settings      Terminal and user parameters         User ID . : SCOMS10
 1  View          Display source data or listings      Time. . . : 14:53
 2  Edit          Create or change source data         Terminal. : 3277
 3  Utilities     Perform utility functions            Screen. . : 1
 4  Foreground    Interactive language processing      Language. : ENGLISH
 5  Batch         Submit job for language processing   Appl ID . : ISR
 6  Command       Enter TSO or Workstation commands    TSO logon : DBPROC96
 7  Dialog Test   Perform dialog testing               TSO prefix: SCOMS10
 9  IBM Products  IBM program development products     System ID : S0W1
 10 SCLM          SW Configuration Library Manager     MVS acct. : GROUP2
 11 Workplace     ISPF Object/Action Workplace         Release . : ISPF 6.1
 12 z/OS System   z/OS system programmer applications
 13 z/OS User     z/OS user applications

     Enter X to Terminate using log/list defaults
```

Fig 3 2 Interactive System Productivity Facility

CHAPTER 4 : JOB CONTROL LANGUAGE (JCL)

4.1 INTRODUCTION

In a mainframe context, JCL is used to communicate between a programme and the operating system (for example, COBOL, Assembler, or PL/I). Programs can be run in batch or online mode in a mainframe system. Processing bank transactions using a VSAM (Virtual Storage Access Method) file and applying them to the appropriate accounts is an example of a batch system. A back office screen used by bank employees to establish an account is an example of an online system. Programs are delivered to the operating system as a job using a JCL in batch mode.

4.2 JOB PROCESSING

A job is a unit of work that may consist of several job phases. Each job step is defined by a series of Job Control Statements in a Job Control Language (JCL).

The Job Entry System (JES) is used by the Operating System to receive jobs, schedule them for processing, and regulate the output.

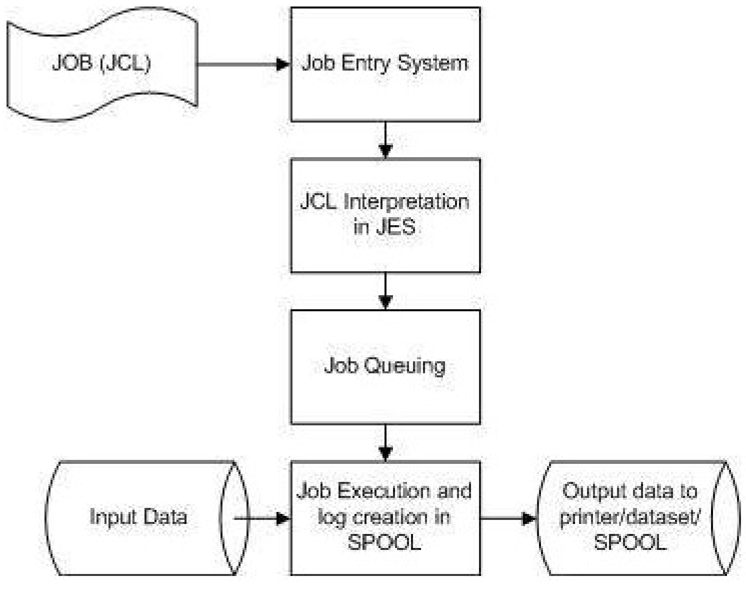
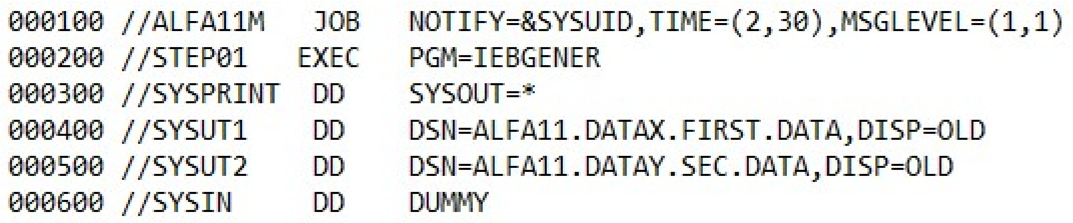

JOB (JCL)
Job Entry System
JCL Interpretation in JES
Job Queuing
Input Data
Job Execution and log creation in SPOOL
Output data to printer/dataset/ SPOOL

Fig 4 1 Job Processing

JOB PROCESSING STEPS

- Job Submission - This is where you send the JCL to JES.
- Job Conversion - The JCL and PROC are translated into an interpreted text that JES can understand and saved in a dataset called SPOOL.
- Job Queuing **-** JES determines the job's priority depending on the CLASS and PRTY criteria in the JOB statement . If there are no mistakes in the JCL code, the job is placed into the job queue.
- Job Execution - When a work achieves its greatest priority, it is selected from the job queue for execution. The JCL file is read from the SPOOL, the programme is run, and the output is redirected to the output destination provided in the JCL file.
- Purging - When the work is finished, the allotted resources are freed, as well as the JES SPOOL space. Before the job log can be released from the SPOOL, it must be copied to another dataset for storage.

4.3 JCL STRUCTURE

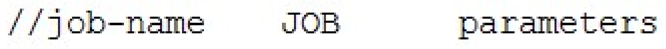


```
//ALFA11M   JOB   NOTIFY=&SYSUID,TIME=(2,30),MSGLEVEL=(1,1)
//STEP01   EXEC   PGM=IEBGENER
//SYSPRINT  DD    SYSOUT=*
//SYSUT1    DD    DSN=ALFA11.DATAX.FIRST.DATA,DISP=OLD
//SYSUT2    DD    DSN=ALFA11.DATAY.SEC.DATA,DISP=OLD
//SYSIN     DD    DUMMY
```

4.4 JOB STATEMENT

In a JCL, the JOB Statement is the initial control statement. This informs the Operating System (OS), the spool, and the scheduler about the job's identity. The JOB statement's arguments assist operating systems in assigning the appropriate scheduler, needed CPU time, and delivering user alerts.

The following is the code for the JOB statement:



```
//job-name    JOB      parameters
```

Let's look at the definitions for the keywords used in the JOB statement syntax above.

4.5 EXEC STATEMENT

The EXEC statement is the statement that carries the job step program/procedure information.

The EXEC statement's purpose is to supply necessary information for the program/procedure that is run in the job phase. If the EXEC statement calls a procedure instead of immediately running a programme, the parameters programmed in this statement can give data to the programme in execution, override specific parameters of the JOB statement, and send parameters to the procedure.

The syntax of a JCL EXEC statement is as follows:

```
//step-name    EXEC      parameters
```

4.6 DD STATEMENT

Datasets are mainframe files that contain records in a specified order. Datasets are fundamental data storage regions that are kept on the mainframe's Direct Access Storage Device (DASD) or Tapes. If these data to be used/created in a batch application, the physical name of the file (i.e., dataset), as well as the file type and organisation, must be coded in a JCL.

The DD statement specifies the definition of each dataset utilised in the JCL. A DD statement must be used to specify the input and output resources required by a task step, including dataset organisation, storage requirements, and record length.

The syntax of a JCL DD statement is as follows:

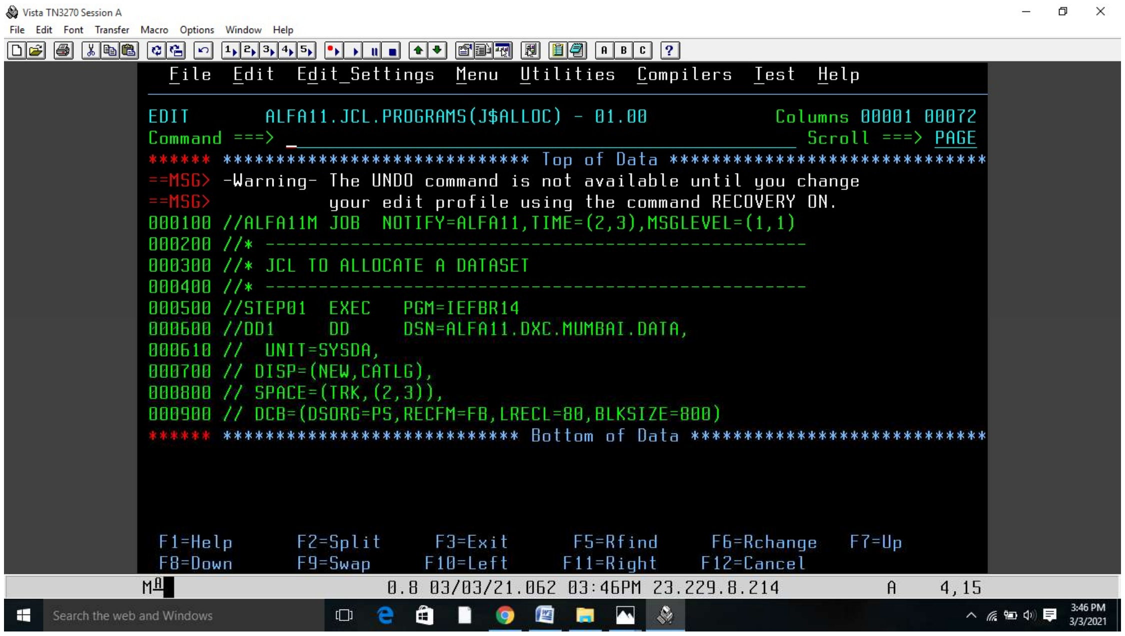
```
//dd-name    DD      parameters
```

4.7 JCL UTILITIES

Utility programmes are pre-written programmes that system programmers and application developers employ on mainframes to meet day-to-day needs such as data organisation and maintenance.

4.7.1 IEFBR14

IEFBR14 is a bogus utility that doesn't perform anything. This application is used to create a dataset and remove one. When this utility is used to allocate or delete a data set, it sends the request to the operating system's functions, which allocate and delete the data set.



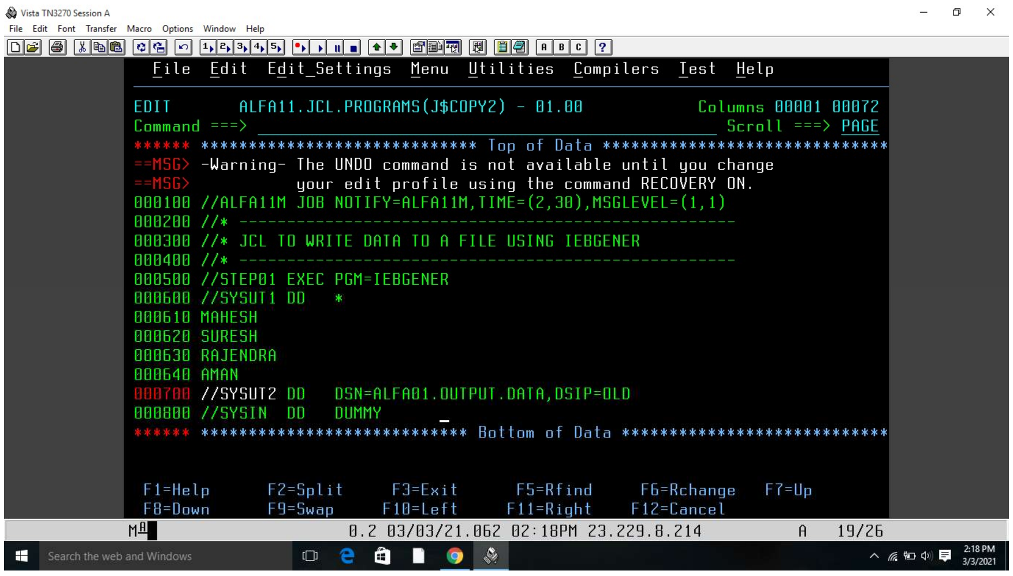
```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       ALFA11.JCL.PROGRAMS(J$ALLOC) - 01.00            Columns 00001 00072
Command ===> _                                                Scroll ===> PAGE
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000100 //ALFA11M JOB  NOTIFY=ALFA11,TIME=(2,3),MSGLEVEL=(1,1)
000200 //* ---------------------------------------------------
000300 //* JCL TO ALLOCATE A DATASET
000400 //* ---------------------------------------------------
000500 //STEP01   EXEC    PGM=IEFBR14
000600 //DD1      DD      DSN=ALFA11.DXC.MUMBAI.DATA,
000610 //  UNIT=SYSDA,
000700 // DISP=(NEW,CATLG),
000800 // SPACE=(TRK,(2,3)),
000900 // DCB=(DSORG=PS,RECFM=FB,LRECL=80,BLKSIZE=800)
****** **************************** Bottom of Data ****************************


 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Fig 4 2 Code Using IEFBR14

4.7.2 IEBGENER

Uses of IEBGENER utility is used to copy non-VSAM data sets with the IEBGENER tool. Non-VSAM data sets' contents can also be printed or shown.

For the IEBGENER tool, the following dd-names must be used: SYSUT1, SYSUT2, SYSIN and SYSPRINT

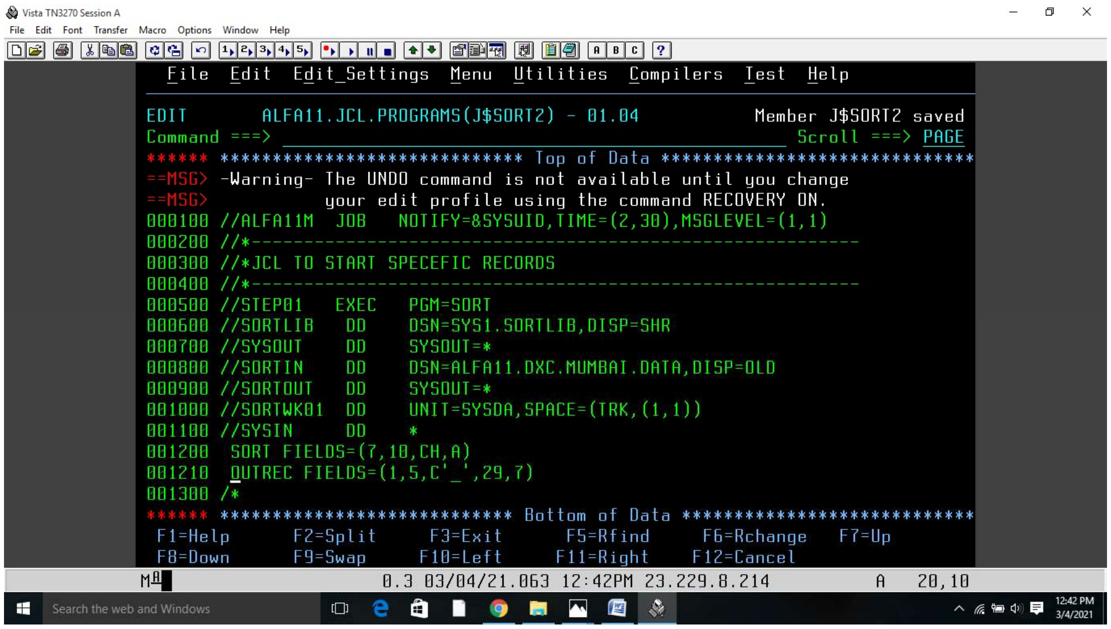


```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       ALFA11.JCL.PROGRAMS(J$COPY2) - 01.00            Columns 00001 00072
Command ===>                                                  Scroll ===> PAGE
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000100 //ALFA11M JOB NOTIFY=ALFA11M,TIME=(2,30),MSGLEVEL=(1,1)
000200 //* ---------------------------------------------------
000300 //* JCL TO WRITE DATA TO A FILE USING IEBGENER
000400 //* ---------------------------------------------------
000500 //STEP01 EXEC PGM=IEBGENER
000600 //SYSUT1 DD   *
000610 MAHESH
000620 SURESH
000630 RAJENDRA
000640 AMAN
000700 //SYSUT2 DD   DSN=ALFA01.OUTPUT.DATA,DSIP=OLD
000800 //SYSIN  DD   DUMMY
****** **************************** Bottom of Data ****************************


 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Fig 4 3 Code Using IEBGENER Utility

4.7.2 SORT

SORT is a sophisticated utility for copying, sorting, and merging information. The input datasets are specified using the SORTIN and SORTIN DD commands. The output data is specified using the SORTOUT and OUTFIL commands. The sort and merge conditions are specified using the SYSIN DD command.



```
EDIT       ALFA11.JCL.PROGRAMS(J$SORT2) - 01.04           Member J$SORT2 saved
Command ===> ________________________________________________ Scroll ===> PAGE
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000100 //ALFA11M  JOB   NOTIFY=&SYSUID,TIME=(2,30),MSGLEVEL=(1,1)
000200 //*-------------------------------------------------------------
000300 //*JCL TO START SPECEFIC RECORDS
000400 //*-------------------------------------------------------------
000500 //STEP01   EXEC   PGM=SORT
000600 //SORTLIB    DD     DSN=SYS1.SORTLIB,DISP=SHR
000700 //SYSOUT     DD     SYSOUT=*
000800 //SORTIN     DD     DSN=ALFA11.DXC.MUMBAI.DATA,DISP=OLD
000900 //SORTOUT    DD     SYSOUT=*
001000 //SORTWK01   DD     UNIT=SYSDA,SPACE=(TRK,(1,1))
001100 //SYSIN      DD     *
001200  SORT FIELDS=(7,10,CH,A)
001210  OUTREC FIELDS=(1,5,C'_',29,7)
001300 /*
****** **************************** Bottom of Data ****************************
 F1=Help       F2=Split      F3=Exit       F5=Rfind      F6=Rchange    F7=Up
 F8=Down       F9=Swap      F10=Left      F11=Right     F12=Cancel
```

Fig 4 4 Code Using Sort Utility

CHAPTER 5 : VIRTUAL STORAGE ACCESS METHOD (VSAM)

5.1 INTRODUCTION

Virtual Storage Access Mechanism (VSAM) is a high-performance access method and data set organisation that uses a catalogue structure to manage and preserve data. It makes use of the virtual storage idea and may password-protect datasets at multiple levels. VSAM, like physical sequential files, may be utilised in COBOL applications. The logical datasets for storing records are called VSAM. In VSAM, files may be read sequentially or randomly. It's a better approach to store data that gets beyond some of the drawbacks of traditional file systems like Sequential Files.

5.2 CHARATERISTICS

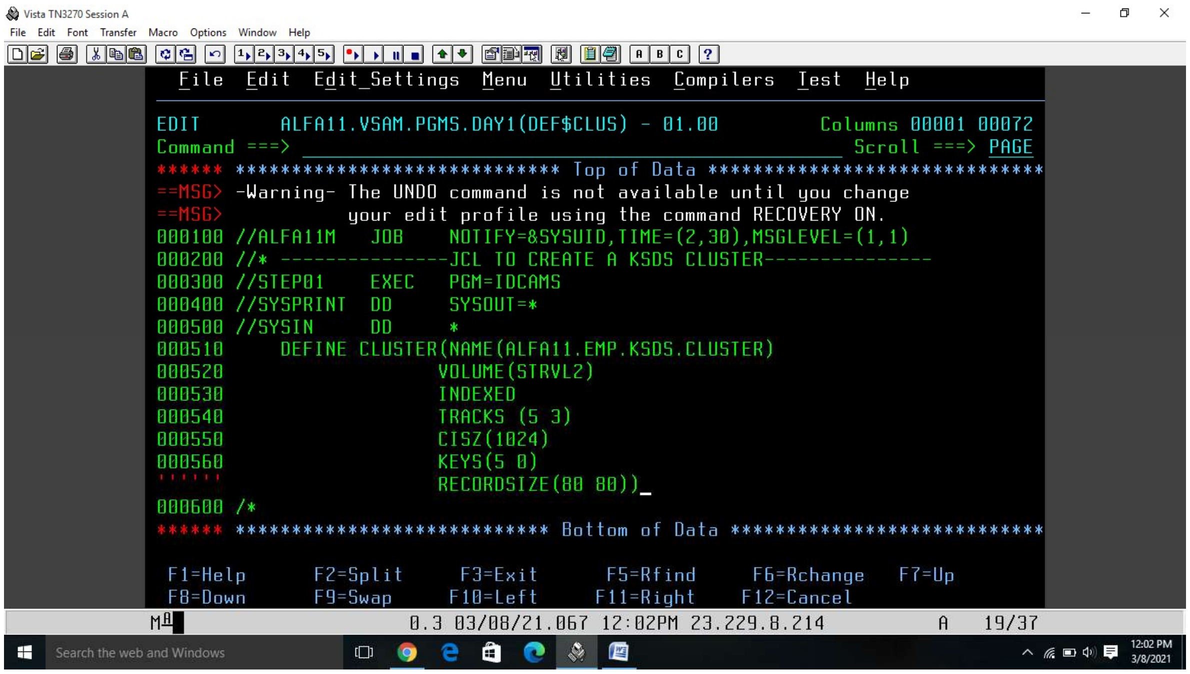
- Passwords are used by VSAM to safeguard data from illegal access.
- VSAM allows users to quickly access data sets.
- VSAM provides performance optimization options.
- In both batch and online environments, VSAM allows data sets to be shared.
- In terms of data storage, VSAM is more organized and orderly.
- In VSAM files, free space is automatically reused.

5.3 IDCAMS

JCL is used to define the VSAM cluster. To construct a cluster, JCL employs the IDCAMS function. IBM created IDCAMS, a tool for access method services. It is mostly used to define VSAM datasets.

Following is the syntax to create a cluster:

```
DEFINE CLUSTER ((NAME(cluster-name)
        INDEXED | NONINDEXED | NUMBERED | LINEAR
                VOLUME(volume-serial-name)
                space-unit(primary, secondary)
                CISZ(size)
                FREESPACE(CI% CA%)
                REUSE | NOREUSE
                KEYS(key-length starting-position)
                RECORDSIZE(average-len maximum-len))
        DATA (NAME(data-component-name))
        INDEX (NAME(index-component-name))
```



```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       ALFA11.VSAM.PGMS.DAY1(DEF$CLUS) - 01.00         Columns 00001 00072
Command ===>                                                  Scroll ===> PAGE
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000100 //ALFA11M   JOB     NOTIFY=&SYSUID,TIME=(2,30),MSGLEVEL=(1,1)
000200 //* ---------------JCL TO CREATE A KSDS CLUSTER---------------
000300 //STEP01    EXEC    PGM=IDCAMS
000400 //SYSPRINT  DD      SYSOUT=*
000500 //SYSIN     DD      *
000510     DEFINE CLUSTER(NAME(ALFA11.EMP.KSDS.CLUSTER)
000520                   VOLUME(STRVL2)
000530                   INDEXED
000540                   TRACKS (5 3)
000550                   CISZ(1024)
000560                   KEYS(5 0)
''''''                   RECORDSIZE(80 80))_
000600 /*
****** **************************** Bottom of Data ****************************

 F1=Help       F2=Split      F3=Exit       F5=Rfind      F6=Rchange    F7=Up
 F8=Down       F9=Swap      F10=Left      F11=Right     F12=Cancel
```

Fig 5 1 IDCAMS Utility to create a Cluster in VSAM

5.4 VSAM DATA SETS

5.4.1 ENTRY SEQUENCED DATA SET (ESDS)

Entry Sequenced Data Set (ESDS) stands for Entry Sequenced Data Set. An entry-sequenced data set works similarly to a sequential file system, but with a few extra features. We have direct access to the records and may also employ passwords for security. For ESDS datasets, we must specify NONINDEXED in the DEFINE CLUSTER command.

The major characteristics of ESDS are as follows:

- Records are kept in the ESDS cluster in the order that they were introduced into the dataset.
- Physical addresses, also known as Relative Byte Addresses, are used to reference records (RBA). If we have 80 byte records in an ESDS dataset, the RBA for the first record will be 0, the RBA for the second record will be 80, the RBA for the third record will be 160, and so on.
- RBA may access records in a sequential order, which is known as addressed access.
- Records are kept in the sequence in which they were input into the system. At the conclusion, new records are added.
- In the ESDS dataset, records cannot be deleted. They can, however, be tagged as inactive.
- The records in the ESDS dataset can be of any length.

5.4.2 KEY SEQUENCED DATA SET (KSDS)

The Key Sequenced Data Set (KSDS) is an acronym for Key Sequenced Data Set. A key-sequenced data set (KSDS) is more complicated than an ESDS or an RRDS, but it is also more valuable and adaptable. For KSDS datasets, we must include INDEXED in the DEFINE CLUSTER command. The following two components make up the KSDS cluster:

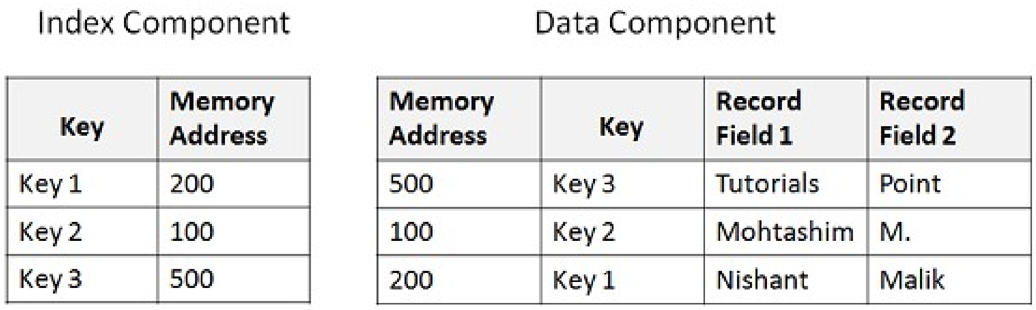
The index component of the KSDS cluster comprises a list of key values for the cluster's records, together with pointers to the data component's associated records. The physical address of a KSDS record is referred to via the index component. The key of each record is linked to the record's position in the data collection. This index is updated whenever a record is added or removed.

The real data is stored in the data component of the KSDS cluster. Each record in a KSDS cluster's data component has the same number of characters and appears in the same relative position in each record.

The primary aspects of KSDS are as follows:

- Within the KSDS data collection, records are always ordered by key-field. By key, records are kept in ascending, collating order.
- Records can be retrieved in any order, and there is also the option of direct access.
- A key is used to identify records. Each record's key is a field in a predetermined location inside the record. In the KSDS dataset, each key must be unique. As a result, record duplication is not feasible.
- When new records are added, the logical order of the records is determined by the key field's collating sequence.
- The length of records in the KSDS dataset might be constant or variable.
- KSDS, like any other file, may be utilised in COBOL applications. The file name will be specified in JCL, and the KSDS file will be used for processing within the application.

Table 5 1 KSDS Components



Index Component

Key	Memory Address
Key 1	200
Key 2	100
Key 3	500

Data Component

Memory Address	Key	Record Field 1	Record Field 2
500	Key 3	Tutorials	Point
100	Key 2	Mohtashim	M.
200	Key 1	Nishant	Malik

5.4.3 LINEAR DATA SET

Linear Data Set (LDS) stands for Linear Data Set. The sole type of byte-stream dataset utilised in traditional operating system files is the linear dataset. Linear datasets are used seldom. The essential aspects of LDS are as follows:

- Because linear datasets do not have any control information inherent in their CI, they do not include RDFs or CIDFs.
- In Linear datasets, data that may be retrieved as byte-addressable strings in virtual storage.
- The control interval size in linear datasets is 4KBytes.
- LDS is a non-vsam file that has some VSAM features such as IDCAMS and VSAM-specific information in the catalogue.
- Linear Data Sets are now used the most by DB2.
- IDCAMS is a programme that is used to define

5.5 ALTERNATE INDEX (AI)

Alternate indexes are indexes that are built in addition to the primary index for KSDS/ESDS datasets. An alternative index allows you to access records by utilising several keys. The alternate index key might be a duplicated key or a non-unique key.

The steps to create an Alternate index are as follow :

1. Defining AI
2. Defining Path
3. Build Index

5.6 CATALOG

The unit and volume where the dataset is stored are maintained by the catalogue. Datasets are retrieved via the catalogue. Non-VSAM datasets use the Disposition Parameter in JCL to build a catalogue item. VSAM datasets have their own catalogue, which is kept in the form of a KSDS cluster.

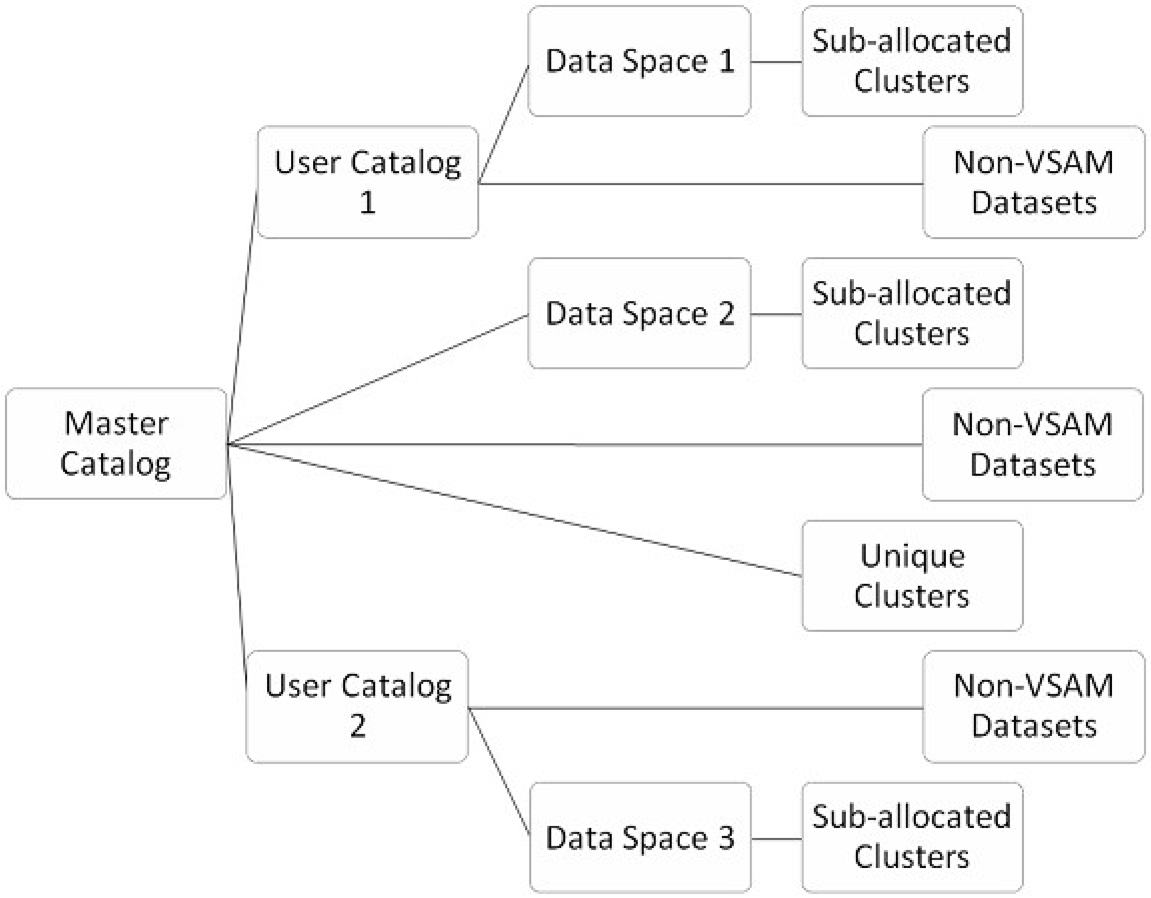


Fig 5 2 Types Of Catalog in VSAM

CHAPTER 6 : COBOL LANGUAGE

6.1 INTRODUCTION

COBOL is a very high-level programming language. It is necessary to comprehend COBOL's operation. Machine code is a binary sequence of 0s and 1s that computers can interpret. A compiler is required to transform COBOL code to machine code. Compile the source code of the application. The compiler checks for syntax mistakes before converting the code to machine language. The load module is the output file created by the compiler. The 0s and 1s in this output file represent executable code.

6.2 COBOL HISTORY

When enterprises in the western world grew in the 1950s, there was a need to automate numerous procedures for ease of operation, which led to the development of a high-level programming language for commercial data processing.

- COBOL was created by CODASYL in 1959. (Conference on Data Systems Language).
- COBOL-61, the following version, was published in 1961 with several changes.
- COBOL was accepted as a standard language for commercial usage by ANSI in 1968. (COBOL-68).
- COBOL-74 and COBOL-85 were developed when it was rewritten again in 1974 and 1985, respectively.
- Object-Oriented COBOL was released in 2002.

6.3 IMPORTANCE

- The first widely used high-level programming language was COBOL. It's an easy-to-understand English-like language. All of the instructions may be written in plain English.
- COBOL is also a self-documenting programming language.
- COBOL is capable of handling large amounts of data.
- COBOL is backwards compatible with earlier versions.

- COBOL offers good error messages, which makes bug resolution easier.

6.4 FEATURES

- Standard English : COBOL is a standard language that may be compiled and run on IBM AS/400s, personal computers, and other platforms.
- Dedicated to the business world : COBOL was created for business-oriented applications in the finance, defence, and other fields. Because of its extensive file processing capabilities, it can manage large amounts of data.
- Language that is robust : COBOL is a strong language with a wide range of debugging and testing facilities for practically all computer platforms.
- Language That Is Organized: COBOL has logical control structures, which makes it easier to comprehend and alter. COBOL is divided into sections, making it simple to debug.

6.5 COBOL STRUCTURE

As demonstrated in the accompanying CHART, a COBOL programme structure is made up of divisions.

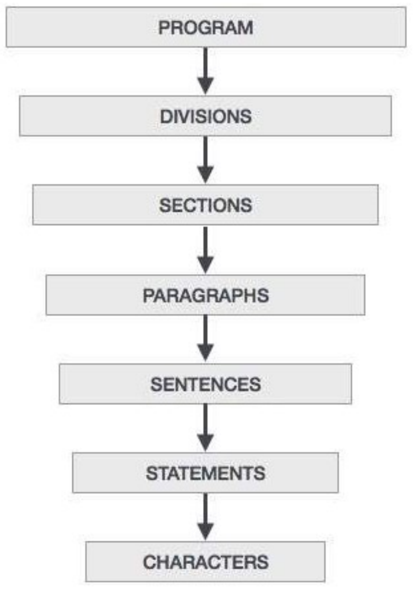


Fig 6 1 COBOL Structure

The following is a basic overview of these divisions:

- The logical subdivision of programme logic is called sections. A section is made up of several paragraphs.
- A section or division is subdivided into paragraphs. It comprises of zero or more sentences/entries and has a user-defined or preset name followed by a period.
- A sentence is made up of one or more statements. Only the Procedure division has sentences. A period must be used to conclude a sentence.
- COBOL statements that do processing are known as statements.
- Characters are at the bottom of the chain and cannot be divided.

6.6 DIVISIONS

There are four divisions in a COBOL programme.

Identification: It is the first and only division in any COBOL programme that must be completed. This division is used by the programmer and the compiler to identify the programme. PROGRAM-ID is the sole necessary paragraph in this division. PROGRAM-ID gives the programme name, which can be between one and thirty characters long.

Environment: The environment division is used to specify the program's input and output files. It is divided into sections.

The system on which the application is written and executed is described in the configuration section. It is made up of two paragraphs.

- The system that was used to compile the application is known as the source computer.
- The application is executed using an object computer system.

The Input-Output section contains details about the files that will be utilised in the application. It is made up of two paragraphs.

- Information about external data sets utilised in the application is provided via the file control.
- I/O control gives you information about the files you're using in the programme.

Data: The variables in the programme are defined via data division. It is divided into four pieces.

The file section is used to specify the file's record structure.

- The Working-Storage section is where you specify the program's temporary variables and file structures.

– The section Local-Storage is comparable to the section Working-Storage. The main distinction is that variables are allocated and initialised each time a programme is run.

– The data names obtained from an external programme are described in the Linkage section.

Procedure: The logic of the programme is included in the procedure division. It is made up of statements that may be executed utilising variables declared in the data division. Paragraph and section titles are user-defined in this division.

In the method division, there must be at least one statement. STOP RUN, which is used in calling programmes, or EXIT PROGRAM, which is used in called programmes, is the last statement to halt execution in this division.

6.7 DATA TYPES

The variables used in a programme are defined via Data Division. To explain data in COBOL, one needs be familiar with the terminology given in the picture.

```
01           TOTAL-STUDENTS           PIC9(5)          VALUE '125'.
|                  |                     |                   |
|                  |                     |                   |
|                  |                     |                   |
Level Number     Data Name           Picture Clause      Value Clause
```

Fig 6 2 Data Types in COBOL

Data Name: Before utilising data names in the Procedure Division, they must first be specified in the Data Division. Reserved words cannot be used; they must have a user-defined name. The memory regions where real data is stored are referred to by data names. They might be simple or multi-leveled.

Level Number: The level of data in a record is indicated by a number. They're utilised to tell the difference between basic and group products. To make group items, elementary objects may be grouped together.

Picture Clause: The following things are defined using the picture clause:

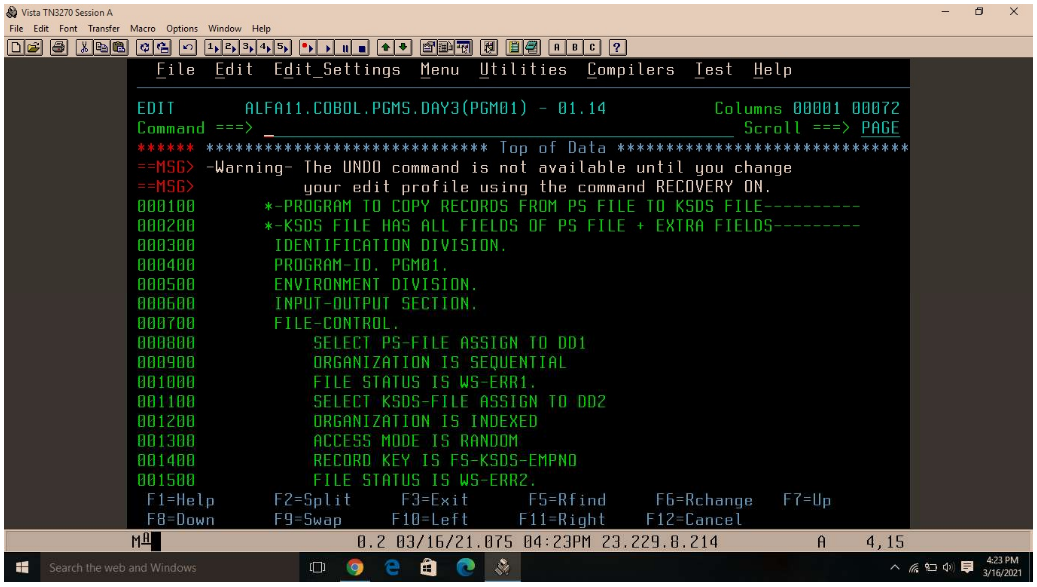
Numeric, alphabetic, or alphanumeric data types are all possible. Only the numbers 0 to 9 make up the numeric type. The letters A through Z, as well as spaces, make up alphabetic type. Digits, letters, and special characters make up the alphanumeric type.

– With numeric data, sign can be employed. It can be either a plus or a minus sign.

– With numeric data, the decimal point location might be utilised. The position of the decimal point is assumed and is not included in the data.

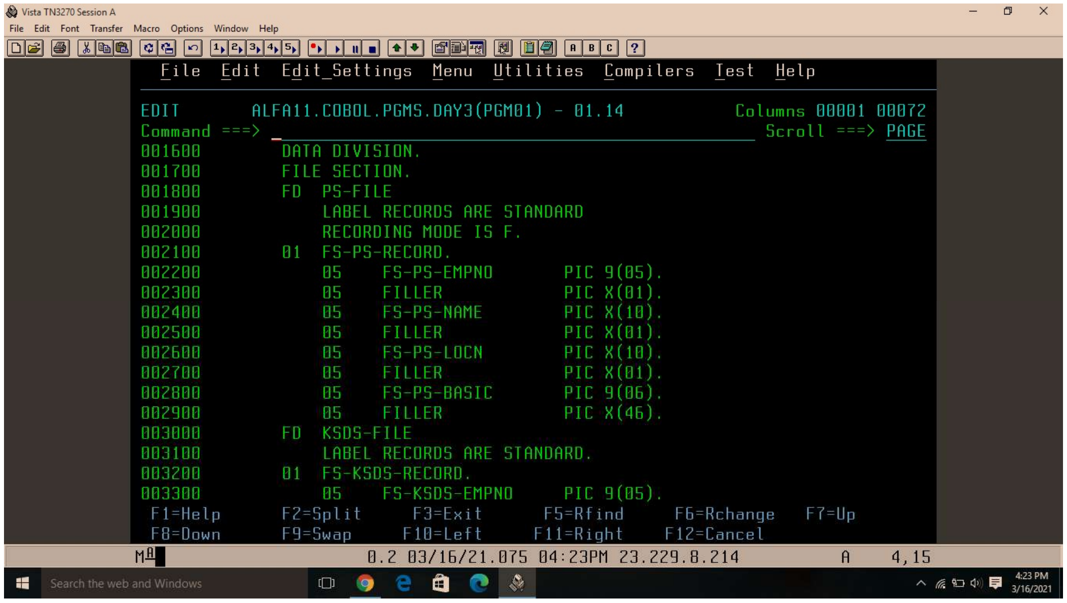
– The number of bytes consumed by the data item is defined by its length.

Value Clause: The value clause is an optional clause that is used to set the data elements' initial values. Numeric literals, alphanumeric literals, and metaphorical constants can all be used. It may be used to both group and basic objects.



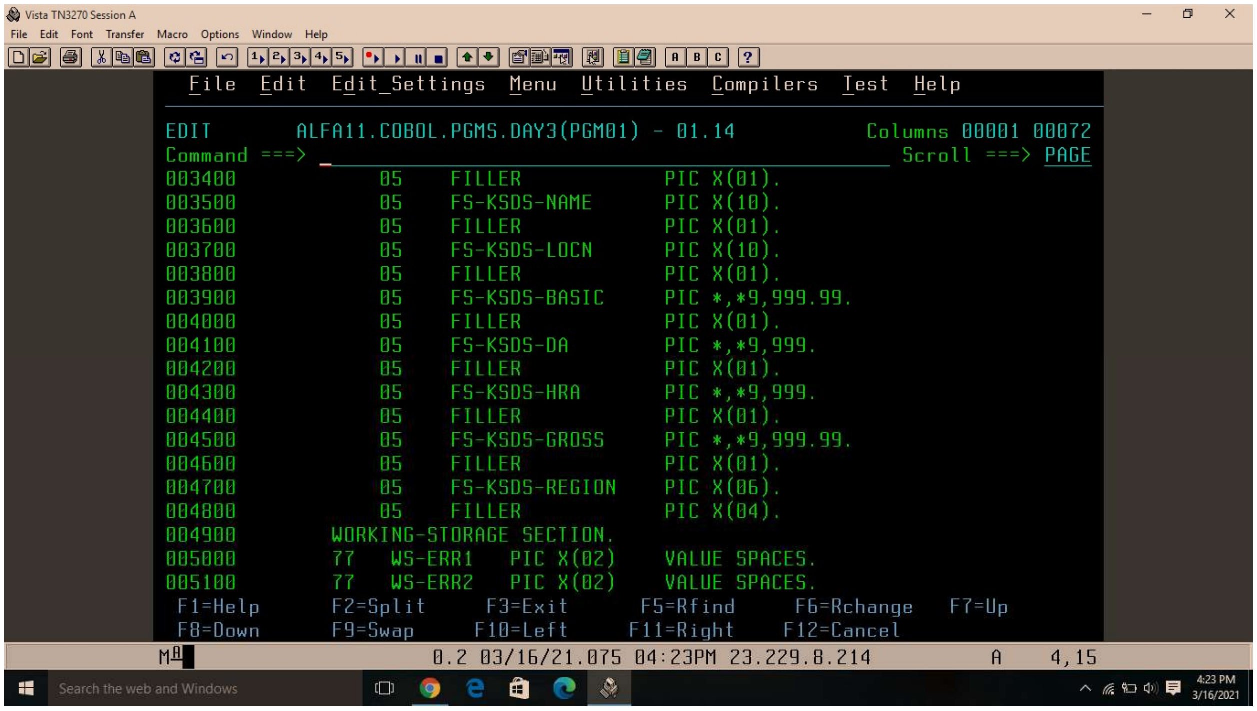
```
EDIT       ALFA11.COBOL.PGMS.DAY3(PGM01) - 01.14           Columns 00001 00072
Command ===> _                                                 Scroll ===> PAGE
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
      *-PROGRAM TO COPY RECORDS FROM PS FILE TO KSDS FILE----------
      *-KSDS FILE HAS ALL FIELDS OF PS FILE + EXTRA FIELDS---------
       IDENTIFICATION DIVISION.
       PROGRAM-ID. PGM01.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT PS-FILE ASSIGN TO DD1
           ORGANIZATION IS SEQUENTIAL
           FILE STATUS IS WS-ERR1.
           SELECT KSDS-FILE ASSIGN TO DD2
           ORGANIZATION IS INDEXED
           ACCESS MODE IS RANDOM
           RECORD KEY IS FS-KSDS-EMPNO
           FILE STATUS IS WS-ERR2.
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap      F10=Left     F11=Right    F12=Cancel
```

Fig 6 3 (a) COBOL code using loops and conditional statements



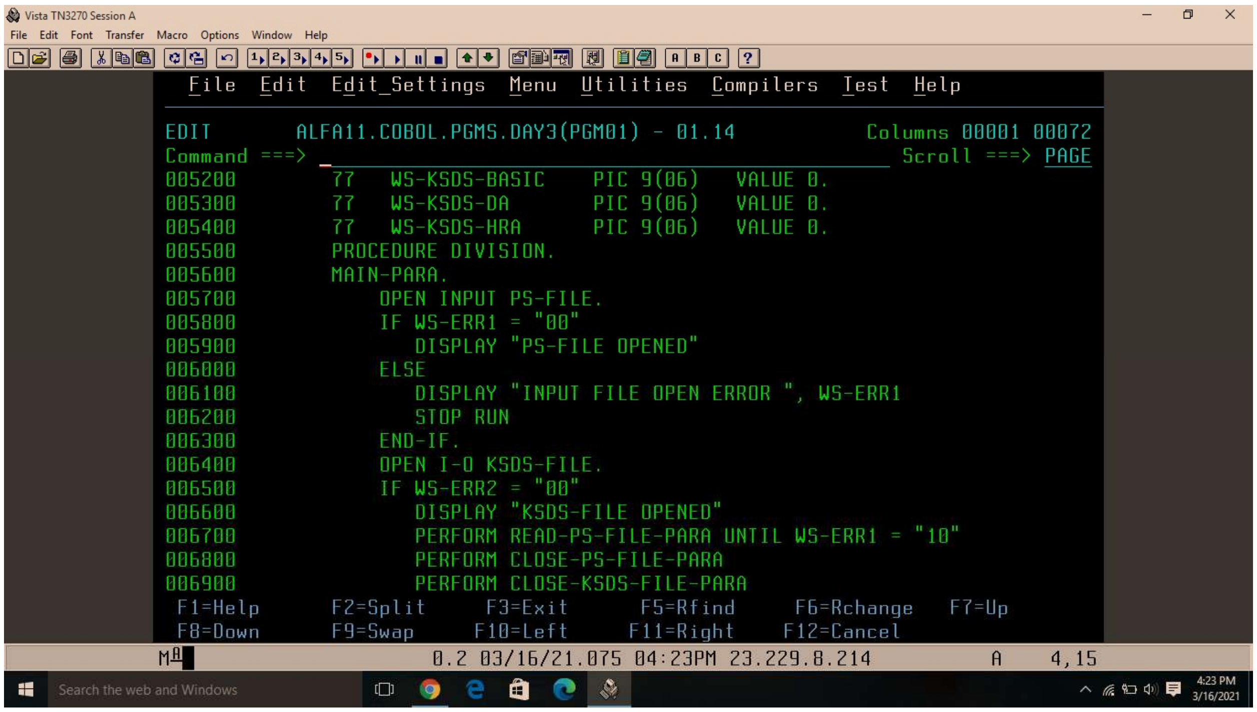
```
EDIT       ALFA11.COBOL.PGMS.DAY3(PGM01) - 01.14           Columns 00001 00072
Command ===> _                                                 Scroll ===> PAGE
       DATA DIVISION.
       FILE SECTION.
       FD  PS-FILE
           LABEL RECORDS ARE STANDARD
           RECORDING MODE IS F.
       01  FS-PS-RECORD.
           05     FS-PS-EMPNO         PIC 9(05).
           05     FILLER              PIC X(01).
           05     FS-PS-NAME          PIC X(10).
           05     FILLER              PIC X(01).
           05     FS-PS-LOCN          PIC X(10).
           05     FILLER              PIC X(01).
           05     FS-PS-BASIC         PIC 9(06).
           05     FILLER              PIC X(46).
       FD  KSDS-FILE
           LABEL RECORDS ARE STANDARD.
       01  FS-KSDS-RECORD.
           05     FS-KSDS-EMPNO       PIC 9(05).
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap      F10=Left     F11=Right    F12=Cancel
```

Fig 6 3(b) COBOL code using loops and conditional statements


Vista TN3270 Session A
File Edit Font Transfer Macro Options Window Help
File Edit Edit_Settings Menu Utilities Compilers Test Help
F1=Help F2=Split F3=Exit F5=Rfind F6=Rchange F7=Up
F8=Down F9=Swap F10=Left F11=Right F12=Cancel
0.2 03/16/21.075 04:23PM 23.229.8.214 A 4,15
Search the web and Windows
4:23 PM 3/16/2021

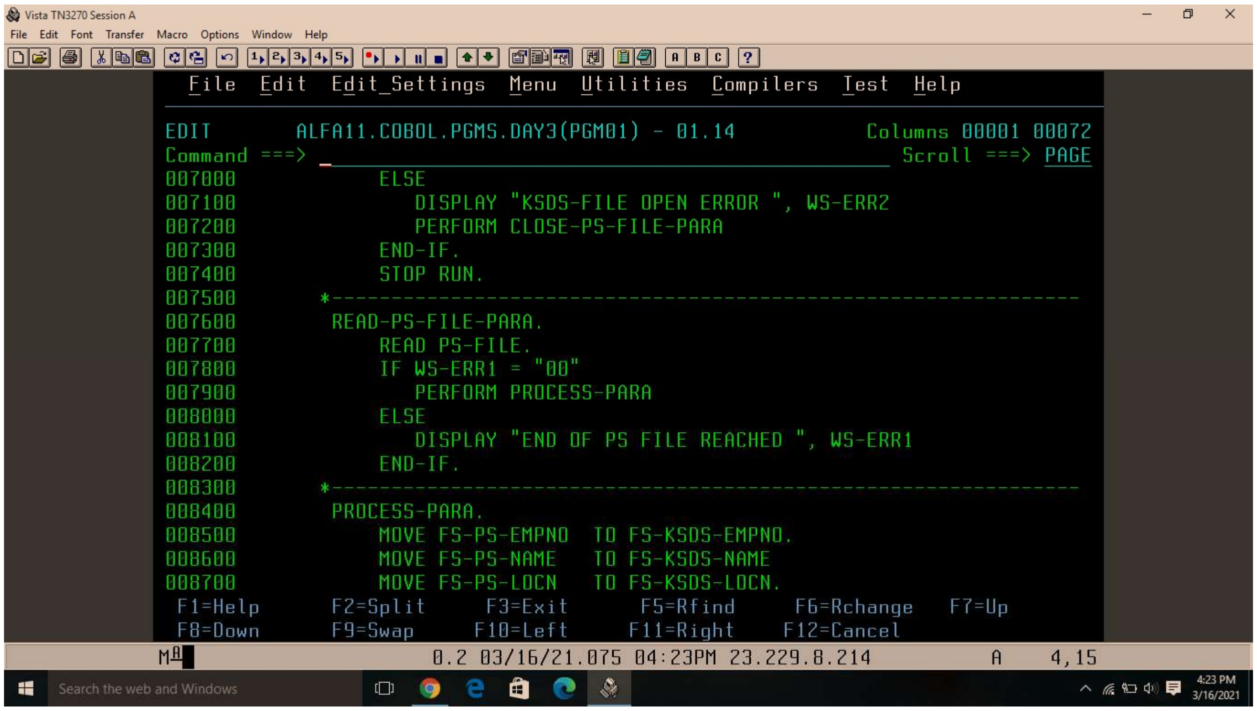
```
EDIT       ALFA11.COBOL.PGMS.DAY3(PGM01) - 01.14           Columns 00001 00072
Command ===> _                                                 Scroll ===> PAGE
           05    FILLER            PIC X(01).
           05    FS-KSDS-NAME      PIC X(10).
           05    FILLER            PIC X(01).
           05    FS-KSDS-LOCN      PIC X(10).
           05    FILLER            PIC X(01).
           05    FS-KSDS-BASIC     PIC *,*9,999.99.
           05    FILLER            PIC X(01).
           05    FS-KSDS-DA        PIC *,*9,999.
           05    FILLER            PIC X(01).
           05    FS-KSDS-HRA       PIC *,*9,999.
           05    FILLER            PIC X(01).
           05    FS-KSDS-GROSS     PIC *,*9,999.99.
           05    FILLER            PIC X(01).
           05    FS-KSDS-REGION    PIC X(06).
           05    FILLER            PIC X(04).
       WORKING-STORAGE SECTION.
       77   WS-ERR1   PIC X(02)    VALUE SPACES.
       77   WS-ERR2   PIC X(02)    VALUE SPACES.
```

Fig 6 3(c) COBOL code using loops and conditional statements


Vista TN3270 Session A
File Edit Font Transfer Macro Options Window Help
File Edit Edit_Settings Menu Utilities Compilers Test Help
F1=Help F2=Split F3=Exit F5=Rfind F6=Rchange F7=Up
F8=Down F9=Swap F10=Left F11=Right F12=Cancel
0.2 03/16/21.075 04:23PM 23.229.8.214 A 4,15
Search the web and Windows
4:23 PM 3/16/2021

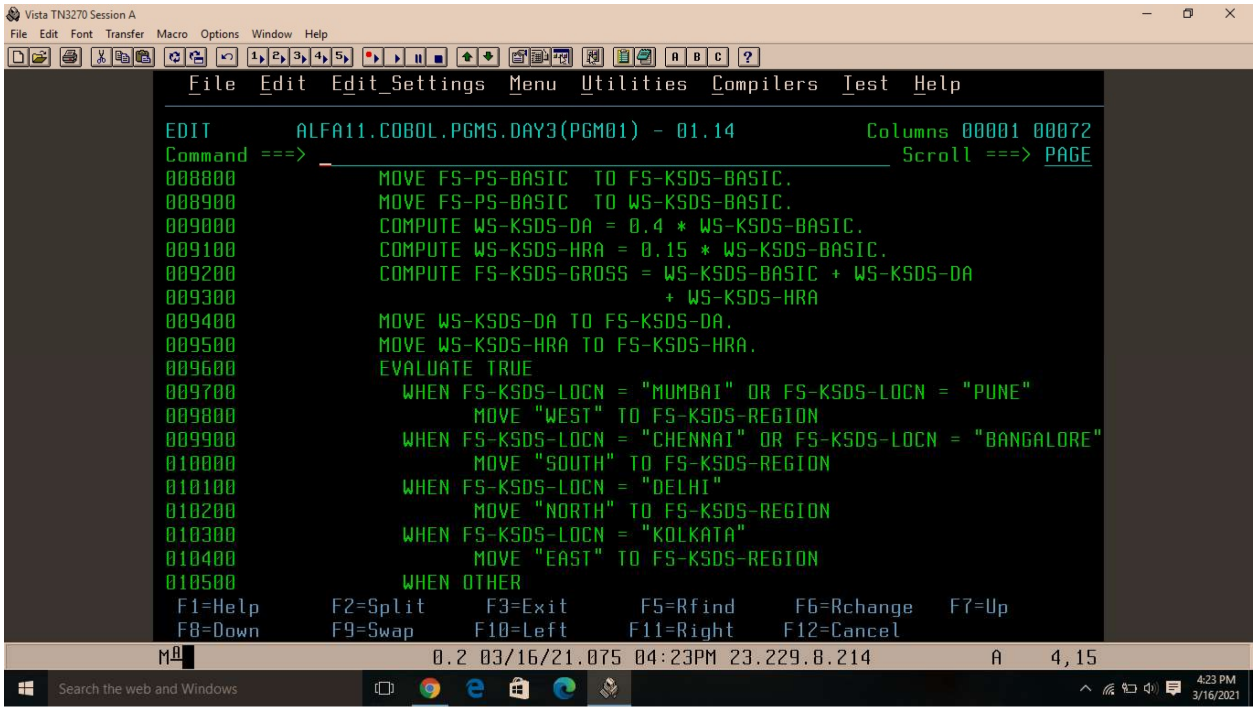
```
EDIT       ALFA11.COBOL.PGMS.DAY3(PGM01) - 01.14           Columns 00001 00072
Command ===> _                                                 Scroll ===> PAGE
       77   WS-KSDS-BASIC    PIC 9(06)   VALUE 0.
       77   WS-KSDS-DA       PIC 9(06)   VALUE 0.
       77   WS-KSDS-HRA      PIC 9(06)   VALUE 0.
       PROCEDURE DIVISION.
       MAIN-PARA.
           OPEN INPUT PS-FILE.
           IF WS-ERR1 = "00"
              DISPLAY "PS-FILE OPENED"
           ELSE
              DISPLAY "INPUT FILE OPEN ERROR ", WS-ERR1
              STOP RUN
           END-IF.
           OPEN I-O KSDS-FILE.
           IF WS-ERR2 = "00"
              DISPLAY "KSDS-FILE OPENED"
              PERFORM READ-PS-FILE-PARA UNTIL WS-ERR1 = "10"
              PERFORM CLOSE-PS-FILE-PARA
              PERFORM CLOSE-KSDS-FILE-PARA
```

Fig 6 3(d) COBOL code using loops and conditional statements


Vista TN3270 Session A
File Edit Font Transfer Macro Options Window Help
File Edit Edit_Settings Menu Utilities Compilers Test Help
F1=Help F2=Split F3=Exit F5=Rfind F6=Rchange F7=Up
F8=Down F9=Swap F10=Left F11=Right F12=Cancel
Search the web and Windows

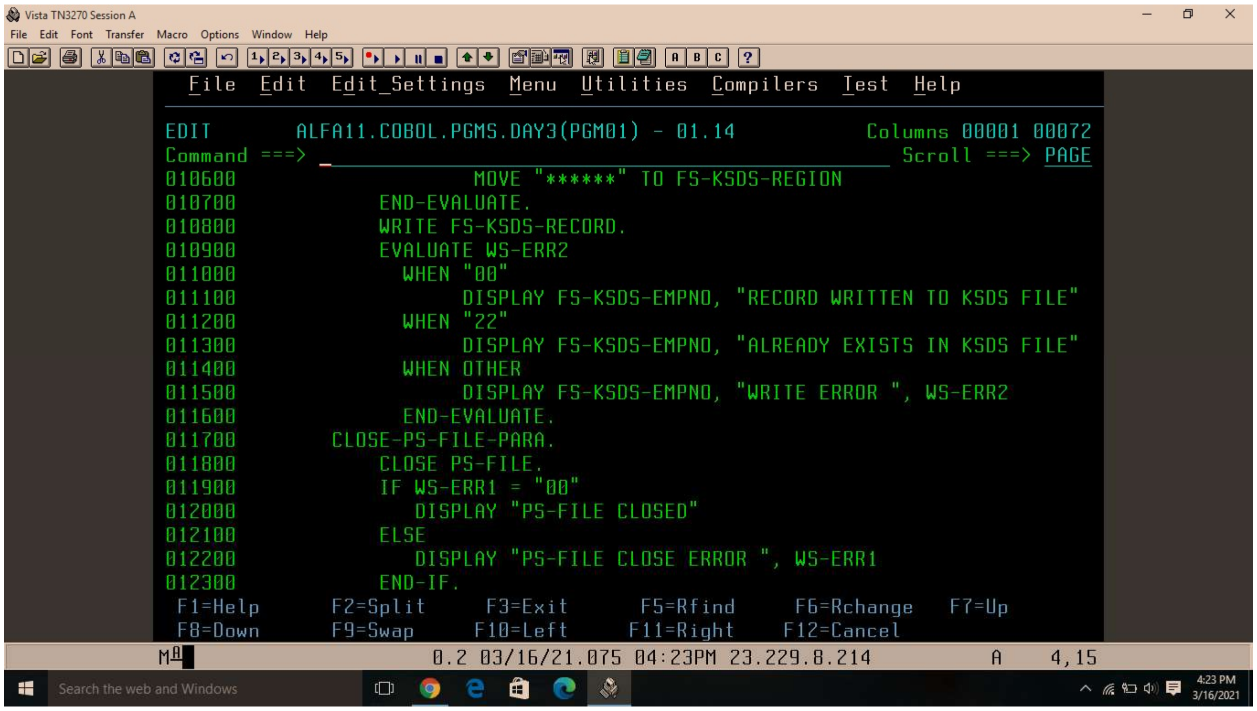
```
EDIT       ALFA11.COBOL.PGMS.DAY3(PGM01) - 01.14          Columns 00001 00072
Command ===> _                                                Scroll ===> PAGE
           ELSE
              DISPLAY "KSDS-FILE OPEN ERROR ", WS-ERR2
              PERFORM CLOSE-PS-FILE-PARA
           END-IF.
           STOP RUN.
      *---------------------------------------------------------------
       READ-PS-FILE-PARA.
           READ PS-FILE.
           IF WS-ERR1 = "00"
              PERFORM PROCESS-PARA
           ELSE
              DISPLAY "END OF PS FILE REACHED ", WS-ERR1
           END-IF.
      *---------------------------------------------------------------
       PROCESS-PARA.
           MOVE FS-PS-EMPNO  TO FS-KSDS-EMPNO.
           MOVE FS-PS-NAME   TO FS-KSDS-NAME
           MOVE FS-PS-LOCN   TO FS-KSDS-LOCN.
```

Fig 6 3(e) COBOL code using loops and conditional statements


Vista TN3270 Session A
File Edit Font Transfer Macro Options Window Help
File Edit Edit_Settings Menu Utilities Compilers Test Help
F1=Help F2=Split F3=Exit F5=Rfind F6=Rchange F7=Up
F8=Down F9=Swap F10=Left F11=Right F12=Cancel
Search the web and Windows

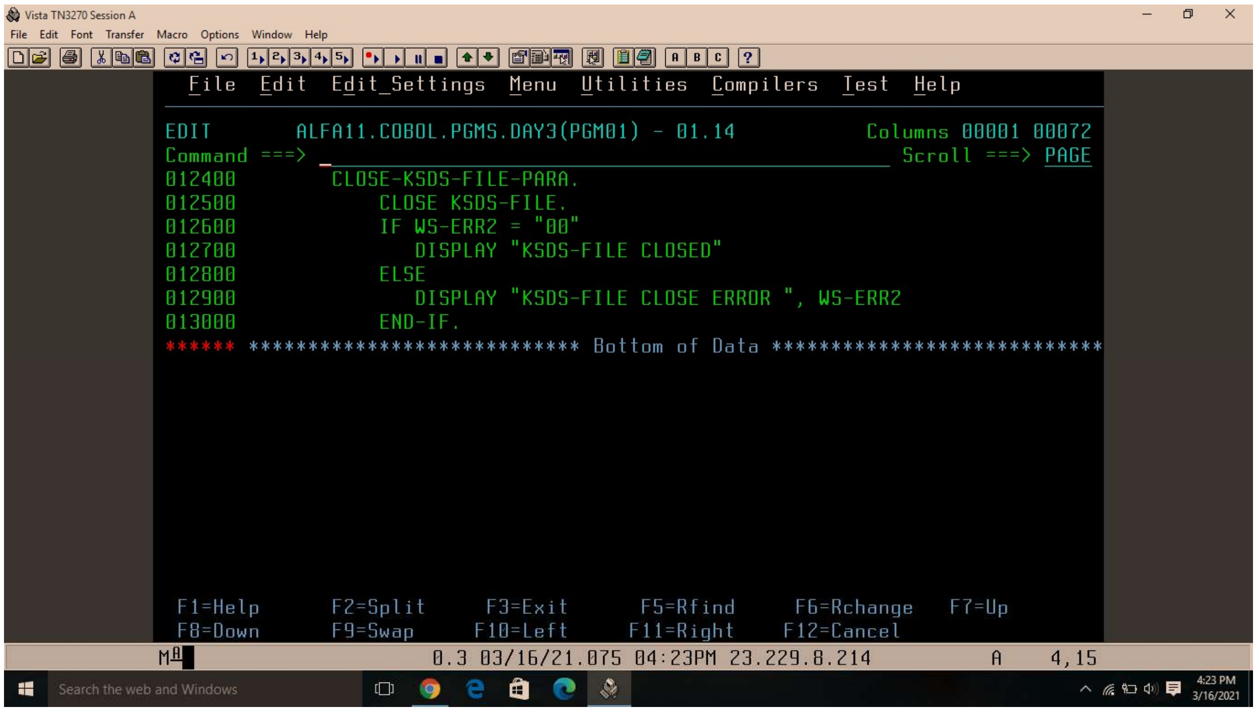
```
EDIT       ALFA11.COBOL.PGMS.DAY3(PGM01) - 01.14          Columns 00001 00072
Command ===> _                                                Scroll ===> PAGE
           MOVE FS-PS-BASIC  TO FS-KSDS-BASIC.
           MOVE FS-PS-BASIC  TO WS-KSDS-BASIC.
           COMPUTE WS-KSDS-DA = 0.4 * WS-KSDS-BASIC.
           COMPUTE WS-KSDS-HRA = 0.15 * WS-KSDS-BASIC.
           COMPUTE FS-KSDS-GROSS = WS-KSDS-BASIC + WS-KSDS-DA
                                   + WS-KSDS-HRA
           MOVE WS-KSDS-DA TO FS-KSDS-DA.
           MOVE WS-KSDS-HRA TO FS-KSDS-HRA.
           EVALUATE TRUE
             WHEN FS-KSDS-LOCN = "MUMBAI" OR FS-KSDS-LOCN = "PUNE"
                   MOVE "WEST" TO FS-KSDS-REGION
             WHEN FS-KSDS-LOCN = "CHENNAI" OR FS-KSDS-LOCN = "BANGALORE"
                   MOVE "SOUTH" TO FS-KSDS-REGION
             WHEN FS-KSDS-LOCN = "DELHI"
                   MOVE "NORTH" TO FS-KSDS-REGION
             WHEN FS-KSDS-LOCN = "KOLKATA"
                   MOVE "EAST" TO FS-KSDS-REGION
             WHEN OTHER
```

Fig 6 3(f) COBOL code using loops and conditional statements


Vista TN3270 Session A
File Edit Font Transfer Macro Options Window Help
File Edit Edit_Settings Menu Utilities Compilers Test Help

```
EDIT       ALFA11.COBOL.PGMS.DAY3(PGM01) - 01.14          Columns 00001 00072
Command ===> _                                              Scroll ===> PAGE
010600                MOVE "******" TO FS-KSDS-REGION
010700        END-EVALUATE.
010800        WRITE FS-KSDS-RECORD.
010900        EVALUATE WS-ERR2
011000          WHEN "00"
011100               DISPLAY FS-KSDS-EMPNO, "RECORD WRITTEN TO KSDS FILE"
011200          WHEN "22"
011300               DISPLAY FS-KSDS-EMPNO, "ALREADY EXISTS IN KSDS FILE"
011400          WHEN OTHER
011500               DISPLAY FS-KSDS-EMPNO, "WRITE ERROR ", WS-ERR2
011600          END-EVALUATE.
011700    CLOSE-PS-FILE-PARA.
011800        CLOSE PS-FILE.
011900        IF WS-ERR1 = "00"
012000           DISPLAY "PS-FILE CLOSED"
012100        ELSE
012200           DISPLAY "PS-FILE CLOSE ERROR ", WS-ERR1
012300        END-IF.
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
             0.2 03/16/21.075 04:23PM 23.229.8.214          A     4,15
```

Fig 6 3(g) COBOL code using loops and conditional statements


Vista TN3270 Session A
File Edit Font Transfer Macro Options Window Help
File Edit Edit_Settings Menu Utilities Compilers Test Help

```
EDIT       ALFA11.COBOL.PGMS.DAY3(PGM01) - 01.14          Columns 00001 00072
Command ===> _                                              Scroll ===> PAGE
012400    CLOSE-KSDS-FILE-PARA.
012500        CLOSE KSDS-FILE.
012600        IF WS-ERR2 = "00"
012700           DISPLAY "KSDS-FILE CLOSED"
012800        ELSE
012900           DISPLAY "KSDS-FILE CLOSE ERROR ", WS-ERR2
013000        END-IF.
****** ***************************** Bottom of Data ****************************
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
             0.3 03/16/21.075 04:23PM 23.229.8.214          A     4,15
```

Fig 6 3(h) COBOL code using loops and conditional statements

CHAPTER 7: DB2/SQL

7.1 INTRODUCTION

IBM's DB2 is a database product. It's a database management system generally known as Relational Database Management System (RDBMS). DB2 is designed to effectively store, analyse, and retrieve data. With the addition of XML, the DB2 software now supports Object-Oriented features and non-relational structures.

SQL is a programming language that is used to manipulate data in a relational database. SQL is a database management system (RDBMS) standard language. It contains a large collection of instructions that allow a programmer to generate, alter, and remove data from a relational database management system. Each relational database management system (RDBMS) has its own set of rules. As a result, while using SQL for an RDBMS, be sure the SQL commands you use meet the RDBMS's standards.

DB2 was originally built by IBM for their own platform. It has been working on a Universal Database (UDB) DB2 Server since 1990, which may run on any authoritative operating system, including Windows, UNIX, and Linux.

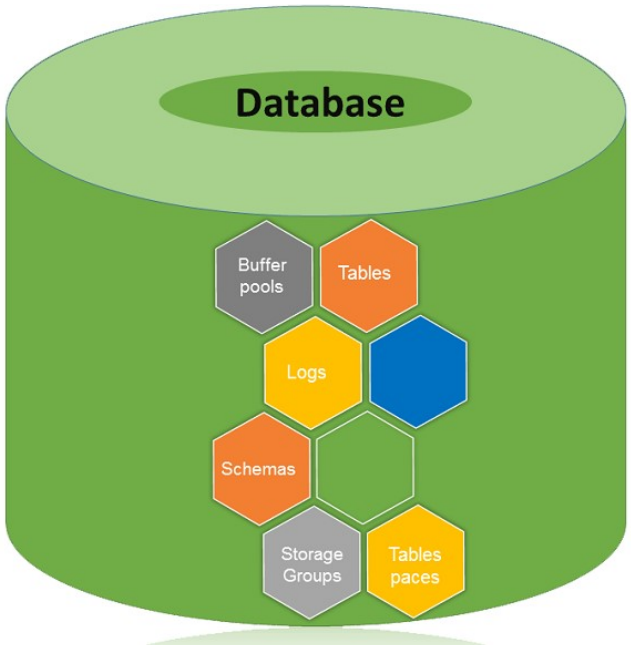
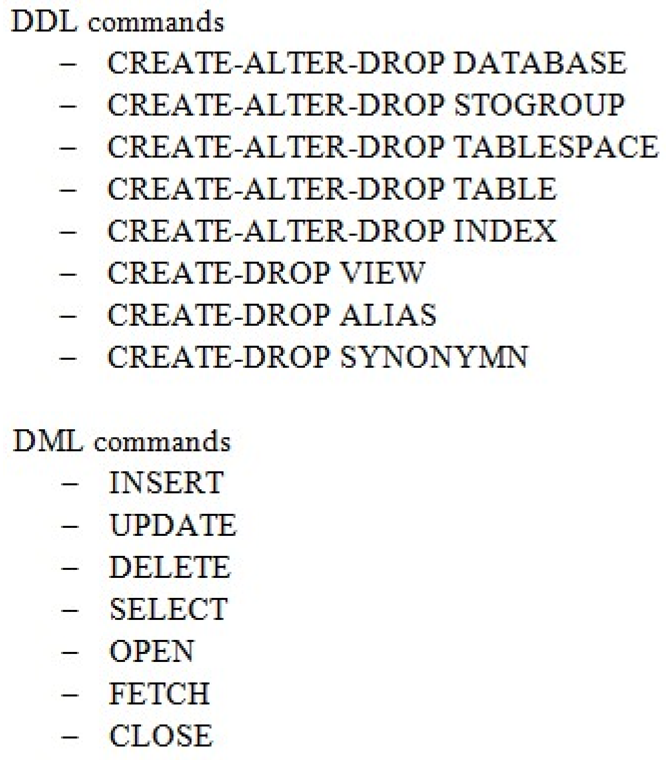


Fig 7 1 Database Architecture

SQL commands may be classified into two groups:

1. DDL - Data Definition Language, which contains commands for database managers to employ.

2. DML - Data Manipulation Language, which contains instructions for programmers to employ.



DDL commands

- CREATE-ALTER-DROP DATABASE
- CREATE-ALTER-DROP STOGROUP
- CREATE-ALTER-DROP TABLESPACE
- CREATE-ALTER-DROP TABLE
- CREATE-ALTER-DROP INDEX
- CREATE-DROP VIEW
- CREATE-DROP ALIAS
- CREATE-DROP SYNONYMN

DML commands

- INSERT
- UPDATE
- DELETE
- SELECT
- OPEN
- FETCH
- CLOSE

7.2 TABLES

Tables are logical structures that Database Manager maintains. Each vertical element in a table is referred to as a column (Tuple), and each horizontal block is referred to as a row (Entity). A table is a collection of data kept in the form of columns and rows. Each column in a table has a separate data type. Tables are used to keep track of data across time.

7.2.1 TABLE TYPES

Base tables are used to store data that is persistent across time. Base tables come in a variety of types, including: Regular tables, Multidimentional Clustering Table, Insert Time Clustering Table, Range-

Clustered tables Table, Partitioned Table and Temporal Table.

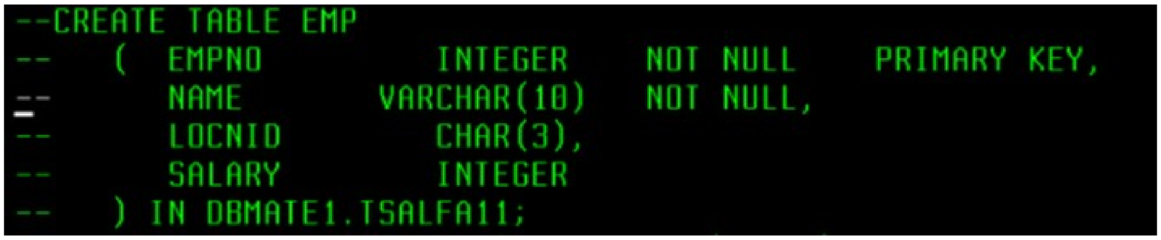
Temporary tables are used for temporary work in various database procedures. The produced temporary tables (DGTTs) do not exist in the system catalogue, and XML columns cannot be utilised in them.

MQT stands for Materialized Query Tables, and it may be used to increase query performance. A query, which is used to determine the data in the tables, defines these sorts of tables.

7.2.2 CREATING A TABLE

The following syntax is used to create a table:

```
CREATE TABLE table
     (     column-1     data-type     NOT NULL PRIMARY KEY,
           column-2     data-type,
           column-3     data-type,
           column-4     data-type
     ) IN database-name.tablespace-name;
```



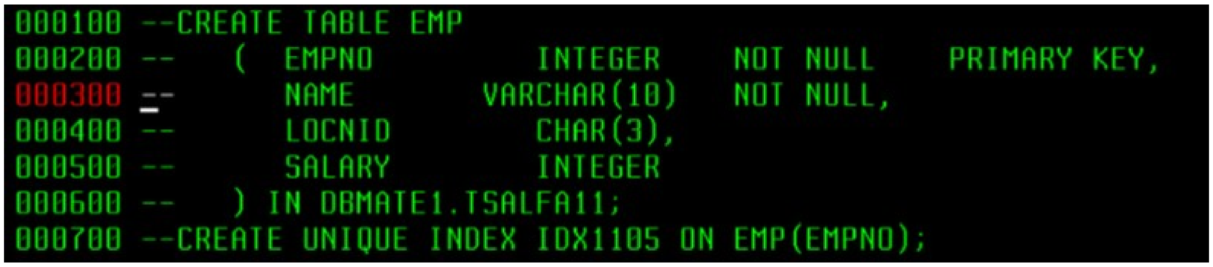
```
--CREATE TABLE EMP
--    (  EMPNO          INTEGER    NOT NULL    PRIMARY KEY,
--       NAME        VARCHAR(10)   NOT NULL,
--       LOCNID         CHAR(3),
--       SALARY         INTEGER
--    ) IN DBMATE1.TSALFA11;
```

Fig 7 2 Code to create a table

7.2.4 CREATING INDEX

The following syntax is used to create a index:

CREATE UNIQUE INDEX   IDX1   ON TEAMS(TCODE);



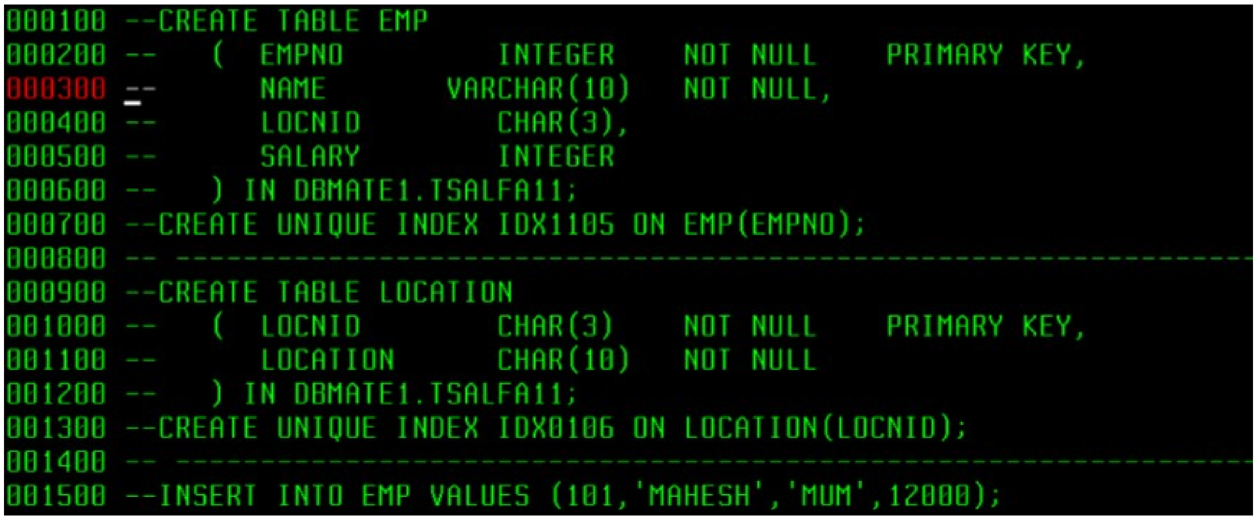
```
--CREATE TABLE EMP
--    (  EMPNO          INTEGER    NOT NULL    PRIMARY KEY,
--       NAME        VARCHAR(10)   NOT NULL,
--       LOCNID         CHAR(3),
--       SALARY         INTEGER
--    ) IN DBMATE1.TSALFA11;
--CREATE UNIQUE INDEX IDX1105 ON EMP(EMPNO);
```

Fig 7 3 Code to create an Index

7.2.5 INSERT

The following syntax is used to insert values into a table:

INSERT INTO table VALUES(value1,value2,value3,…);



```
--CREATE TABLE EMP
--    (  EMPNO          INTEGER    NOT NULL    PRIMARY KEY,
--       NAME        VARCHAR(10)   NOT NULL,
--       LOCNID         CHAR(3),
--       SALARY         INTEGER
--    ) IN DBMATE1.TSALFA11;
--CREATE UNIQUE INDEX IDX1105 ON EMP(EMPNO);
-- ---------------------------------------------------------------
--CREATE TABLE LOCATION
--    (  LOCNID         CHAR(3)    NOT NULL    PRIMARY KEY,
--       LOCATION       CHAR(10)   NOT NULL
--    ) IN DBMATE1.TSALFA11;
--CREATE UNIQUE INDEX IDX0106 ON LOCATION(LOCNID);
-- ---------------------------------------------------------------
--INSERT INTO EMP VALUES (101,'MAHESH','MUM',12000);
```

Fig 7 4 Code to insert values

7.2.6 UPDATE AND DELETE A TABLE

The following syntax is used to update values in a table:

Syntax:

UPDATE table SET column = value WHERE condition;

Example:

UPDATE TEAMS SET CAPTAIN = 'KOHLI' WHERE TCODE = 'IND';

The following syntax is used to delete a table:

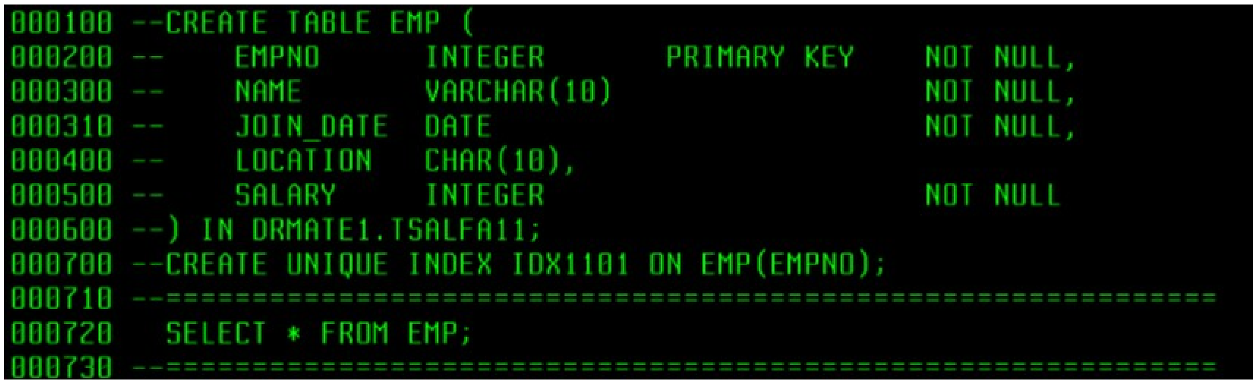
Syntax:
DELETE FROM table WHERE condition;

Example:
DELETE FROM PLAYERS WHERE PCODE = 'P34';

7.2.7 SELECT COMMAND

It is used to retrieve values. The following syntax is used for the select command:

SELECT {DISTINCT} column-1, column-2, … FROM table;



```
--CREATE TABLE EMP (
--    EMPNO       INTEGER        PRIMARY KEY    NOT NULL,
--    NAME        VARCHAR(10)                   NOT NULL,
--    JOIN_DATE   DATE                          NOT NULL,
--    LOCATION    CHAR(10),
--    SALARY      INTEGER                       NOT NULL
--) IN DRMATE1.TSALFA11;
--CREATE UNIQUE INDEX IDX1101 ON EMP(EMPNO);
--===================================================================
  SELECT * FROM EMP;
--===================================================================
```

Fig 7 5 Code to select

7.3 EMBEDDED SQL PROGRAMMING

Embedded SQL is a programming approach that involves embedding SQL instructions into a COBOL programme. A COBOL/DB2 programme is an example of this type of software. In a COBOL/DB2 application, all SQL instructions must be written between EXEC SQL and END-EXEC. Only one SQL command can be included in an EXEC SQL block. The SQL command must not terminate with ; in the EXEC SQL block. The EXEC SQL block must be written in Area B at all times.

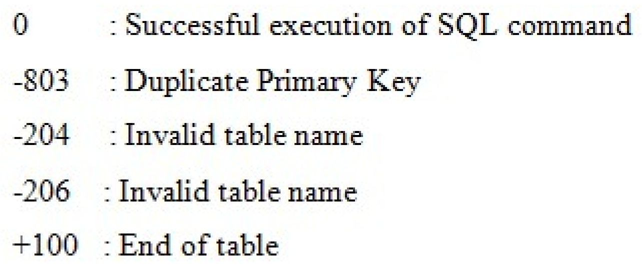
- The INCLUDE command will insert the member's code into the location where the INCLUDE command is supplied.
- In COBOL, the INCLUDE command is similar to the COPY command. This command will be run throughout the compilation process.

7.3.1 HOST VARIABLES

- Variables in the working-storage section can only be utilised outside of an EXEC SQL block.
- Column names are only allowed to be used within an EXEC SQL block.
- You can utilise host variables both inside and outside of an EXEC SQL block.
- Host variables are defined in the DCLGEN member, which is generated using the ISPF option 8 DCLGEN option.
- By default, the name of the host variable is the same as the name of the column. The DCLGEN option allows you to add a prefix.
- When using a host variable in an EXEC SQL block, type : before the host variable (:HV-EMPNO).
- : must not be written before the host variables when they are utilised outside of an EXEC SQL block (HV-EMPNO)

7.3.2 SQLCODE VARIABLE

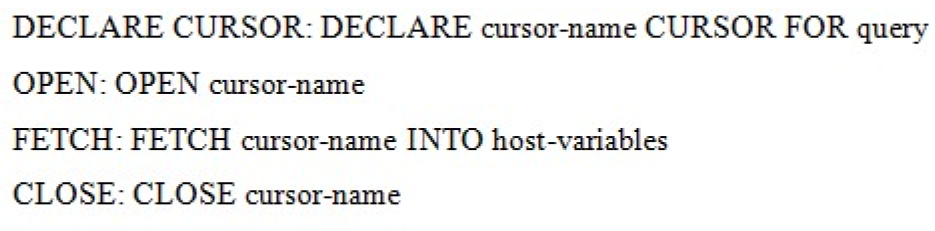
- SQLCODE is a built-in DB2 variable that contains the SQL command's execution status. If the SQL command is successful, it will return a result of 0 or positive. If the SQL statement fails to run, it will return a negative result. Checking the SQLCODE after each EXEC SQL block is a good idea.
- SQLCODE is defined in the SQLCA member and has the S9(09) COMP PIC clause.
- CONTAIN To utilise SQLCODE in the application, you must use the SQLCA command.
- SQL codes that are commonly used:



0	: Successful execution of SQL command
-803	: Duplicate Primary Key
-204	: Invalid table name
-206	: Invalid table name
+100	: End of table

7.3.3 CURSORS

- If you have a SELECT in a programme that returns numerous rows, you'll need to employ a cursor.
- The cursor is related with the following four commands:



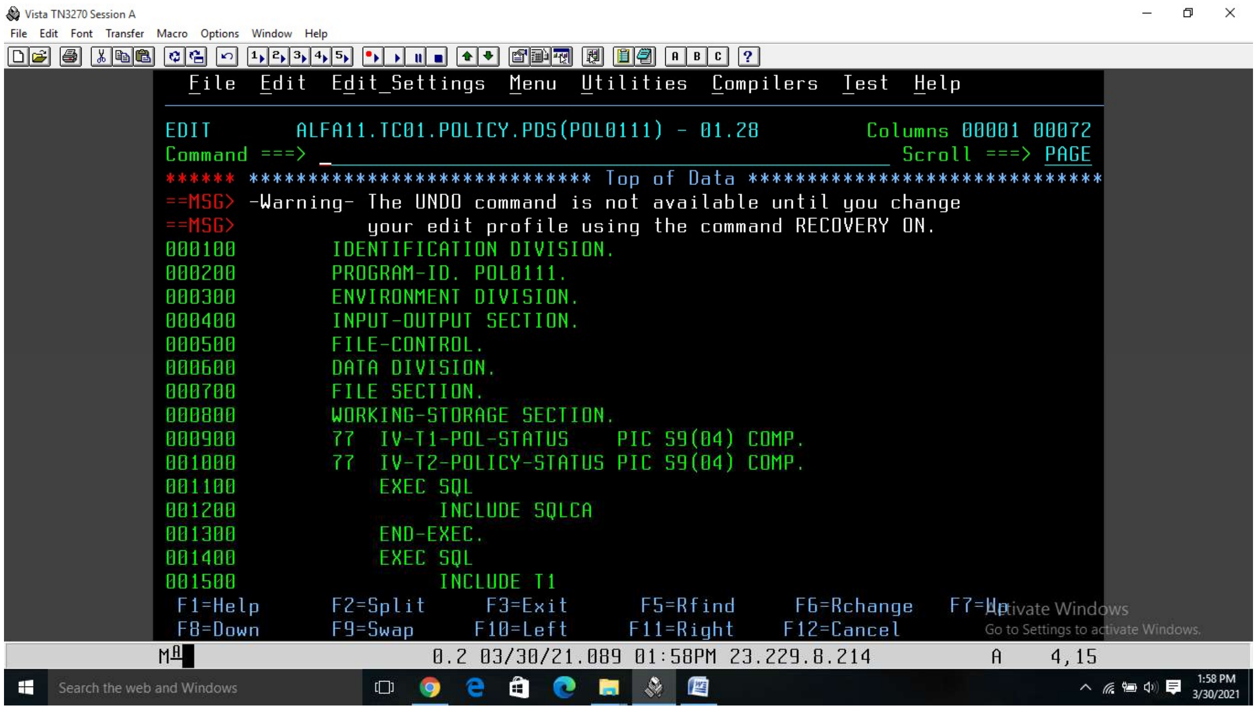
DECLARE CURSOR: DECLARE cursor-name CURSOR FOR query

OPEN: OPEN cursor-name

FETCH: FETCH cursor-name INTO host-variables

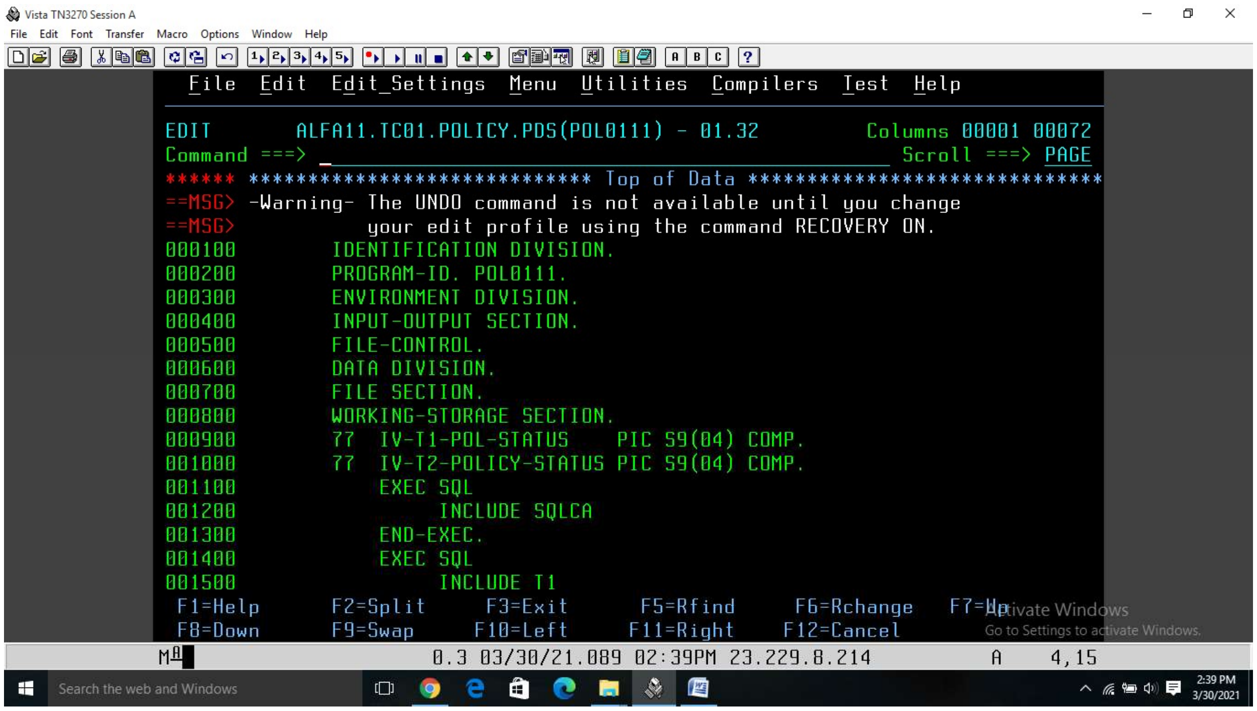
CLOSE: CLOSE cursor-name

- In the WORKING-STORAGE SECTION, type the DECLARE CURSOR command to set the cursor. We mention the cursor's name and correlate it with a SELECT query. DECLARE CURSOR creates a record in the DB2 system table with the cursor name and the SELECT query. The DECLARE CURSOR command does not run the SELECT query.
- The OPEN command will search the DB2 system table for the cursor-name specified in the command. If the cursor-name is found in the table, it will run the SELECT query associated with it and produce a result table that the cursor will refer to.
- The FETCH command retrieves the current row of the cursor table and assigns the data to the host variables specified.
- The CLOSE command is used to close the cursor.


Vista TN3270 Session A
File Edit Font Transfer Macro Options Window Help
Search the web and Windows
1:58 PM
3/30/2021

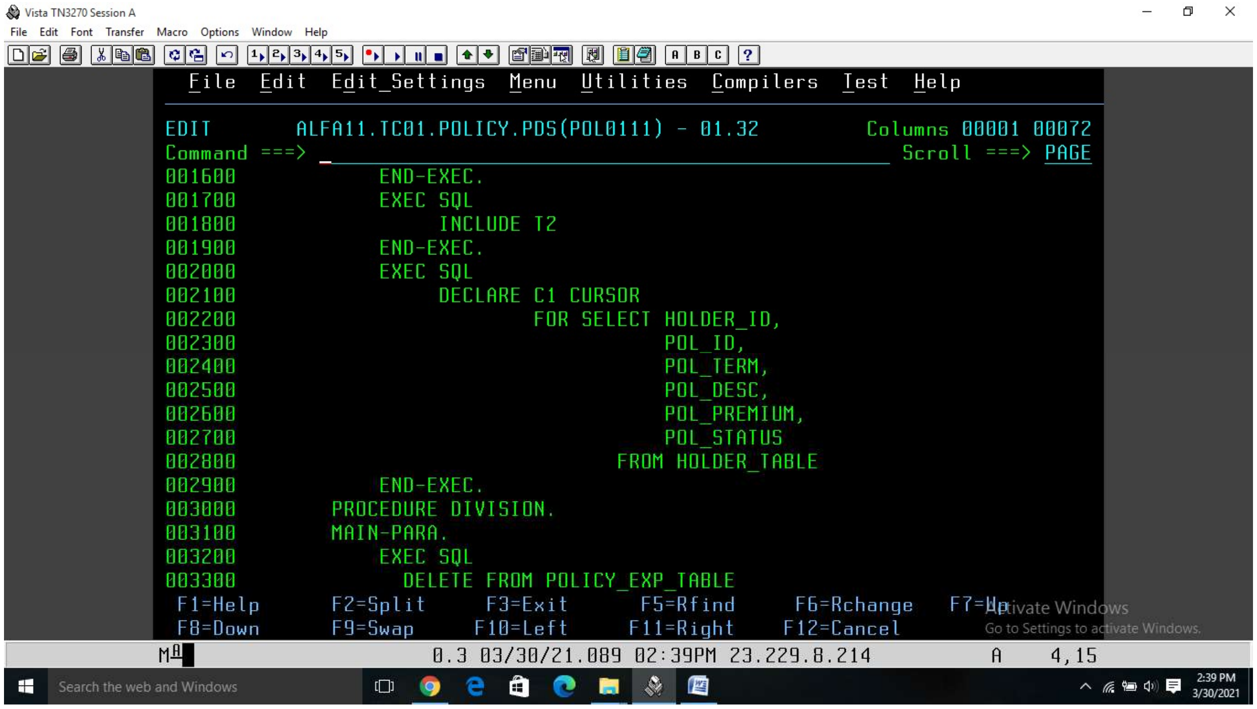
```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       ALFA11.TC01.POLICY.PDS(POL0111) - 01.28         Columns 00001 00072
Command ===> _                                                 Scroll ===> PAGE
****** ****************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000100        IDENTIFICATION DIVISION.
000200        PROGRAM-ID. POL0111.
000300        ENVIRONMENT DIVISION.
000400        INPUT-OUTPUT SECTION.
000500        FILE-CONTROL.
000600        DATA DIVISION.
000700        FILE SECTION.
000800        WORKING-STORAGE SECTION.
000900        77  IV-T1-POL-STATUS     PIC S9(04) COMP.
001000        77  IV-T2-POLICY-STATUS PIC S9(04) COMP.
001100            EXEC SQL
001200                 INCLUDE SQLCA
001300            END-EXEC.
001400            EXEC SQL
001500                 INCLUDE T1
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
    MA              0.2 03/30/21.089 01:58PM 23.229.8.214          A     4,15
```

Fig 7 6 COBOL/DB2 Code


Vista TN3270 Session A
File Edit Font Transfer Macro Options Window Help
Search the web and Windows
2:39 PM
3/30/2021

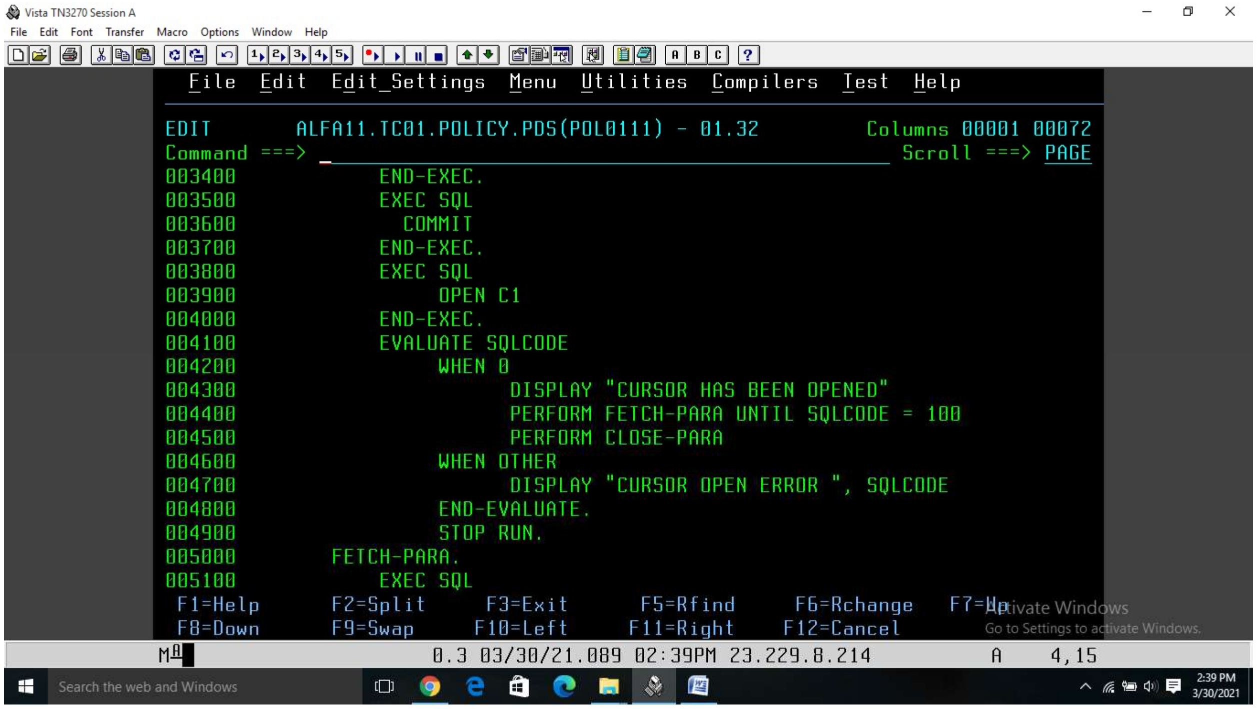
```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       ALFA11.TC01.POLICY.PDS(POL0111) - 01.32         Columns 00001 00072
Command ===> _                                                 Scroll ===> PAGE
****** ****************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000100        IDENTIFICATION DIVISION.
000200        PROGRAM-ID. POL0111.
000300        ENVIRONMENT DIVISION.
000400        INPUT-OUTPUT SECTION.
000500        FILE-CONTROL.
000600        DATA DIVISION.
000700        FILE SECTION.
000800        WORKING-STORAGE SECTION.
000900        77  IV-T1-POL-STATUS     PIC S9(04) COMP.
001000        77  IV-T2-POLICY-STATUS PIC S9(04) COMP.
001100            EXEC SQL
001200                 INCLUDE SQLCA
001300            END-EXEC.
001400            EXEC SQL
001500                 INCLUDE T1
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
    MA              0.3 03/30/21.089 02:39PM 23.229.8.214          A     4,15
```

Fig 7 7 (a)COBOL/DB2 Code



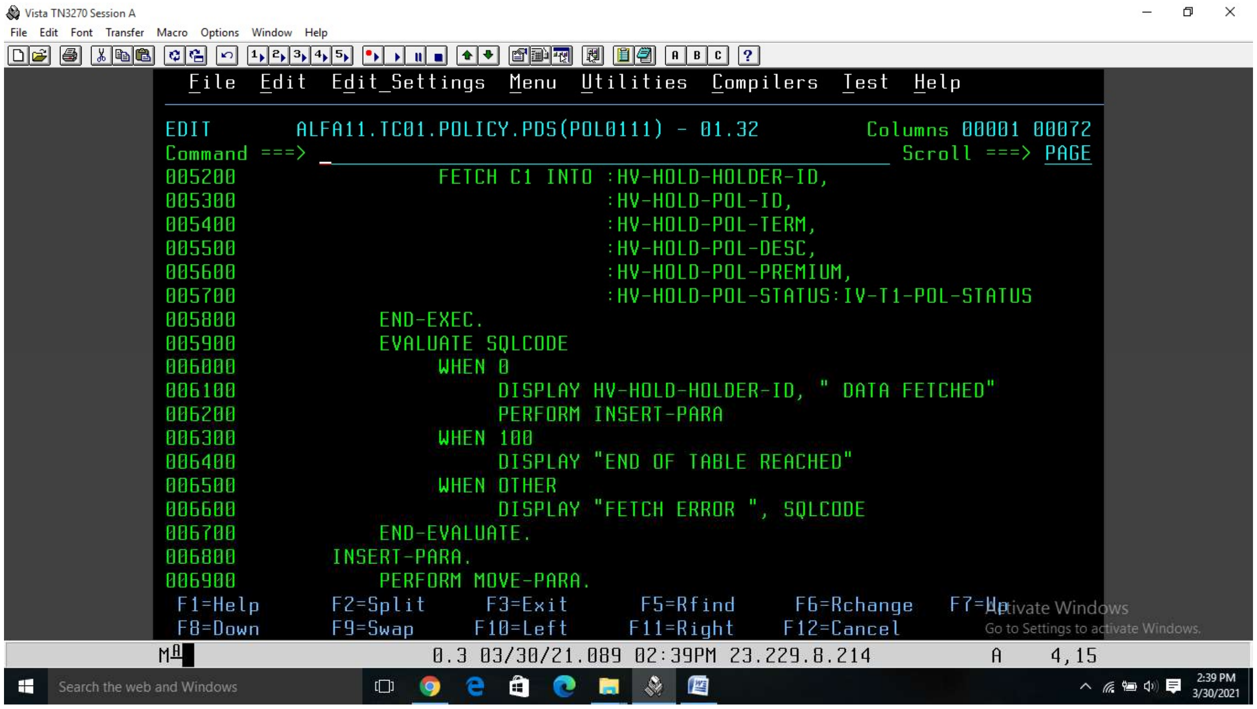
```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       ALFA11.TC01.POLICY.PDS(POL0111) - 01.32          Columns 00001 00072
Command ===> _                                               Scroll ===> PAGE
                END-EXEC.
                EXEC SQL
                     INCLUDE T2
                END-EXEC.
                EXEC SQL
                     DECLARE C1 CURSOR
                             FOR SELECT HOLDER_ID,
                                        POL_ID,
                                        POL_TERM,
                                        POL_DESC,
                                        POL_PREMIUM,
                                        POL_STATUS
                                    FROM HOLDER_TABLE
                END-EXEC.
            PROCEDURE DIVISION.
            MAIN-PARA.
                EXEC SQL
                  DELETE FROM POLICY_EXP_TABLE
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
MA                     0.3 03/30/21.089 02:39PM 23.229.8.214          A     4,15
```

Fig 7 8 (b)COBOL/DB2 Code



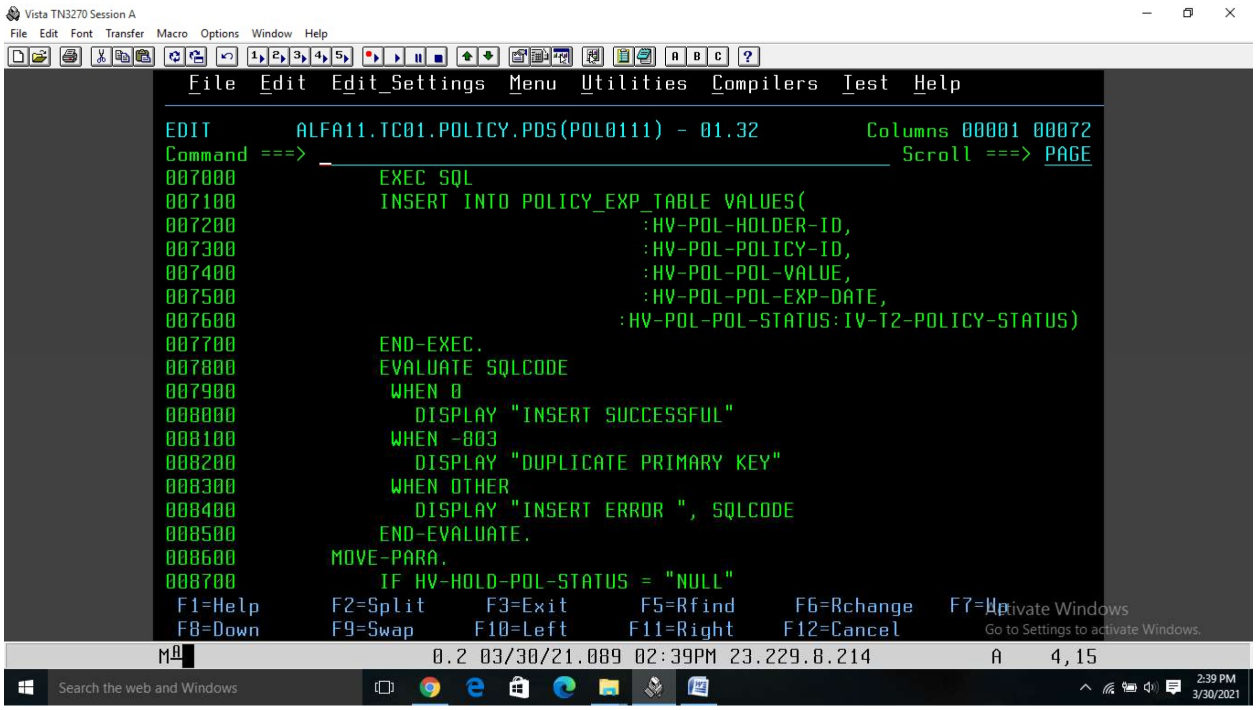
```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       ALFA11.TC01.POLICY.PDS(POL0111) - 01.32          Columns 00001 00072
Command ===> _                                               Scroll ===> PAGE
                END-EXEC.
                EXEC SQL
                  COMMIT
                END-EXEC.
                EXEC SQL
                     OPEN C1
                END-EXEC.
                EVALUATE SQLCODE
                     WHEN 0
                           DISPLAY "CURSOR HAS BEEN OPENED"
                           PERFORM FETCH-PARA UNTIL SQLCODE = 100
                           PERFORM CLOSE-PARA
                     WHEN OTHER
                           DISPLAY "CURSOR OPEN ERROR ", SQLCODE
                     END-EVALUATE.
                     STOP RUN.
            FETCH-PARA.
                EXEC SQL
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
MA                     0.3 03/30/21.089 02:39PM 23.229.8.214          A     4,15
```

Fig 7 9 (c)COBOL/DB2 Code



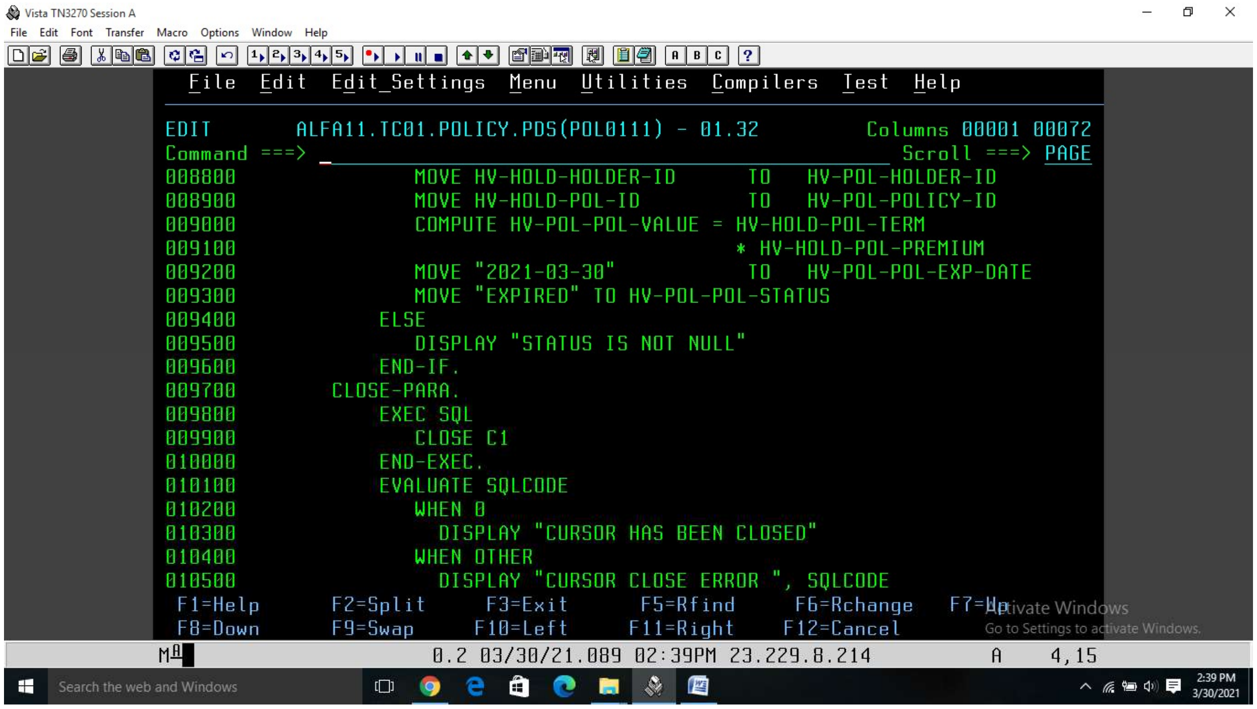
```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       ALFA11.TC01.POLICY.PDS(POL0111) - 01.32         Columns 00001 00072
Command ===> _                                                Scroll ===> PAGE
               FETCH C1 INTO :HV-HOLD-HOLDER-ID,
                             :HV-HOLD-POL-ID,
                             :HV-HOLD-POL-TERM,
                             :HV-HOLD-POL-DESC,
                             :HV-HOLD-POL-PREMIUM,
                             :HV-HOLD-POL-STATUS:IV-T1-POL-STATUS
          END-EXEC.
          EVALUATE SQLCODE
               WHEN 0
                    DISPLAY HV-HOLD-HOLDER-ID, " DATA FETCHED"
                    PERFORM INSERT-PARA
               WHEN 100
                    DISPLAY "END OF TABLE REACHED"
               WHEN OTHER
                    DISPLAY "FETCH ERROR ", SQLCODE
          END-EVALUATE.
      INSERT-PARA.
          PERFORM MOVE-PARA.
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Fig 7 10 (d)COBOL/DB2 Code



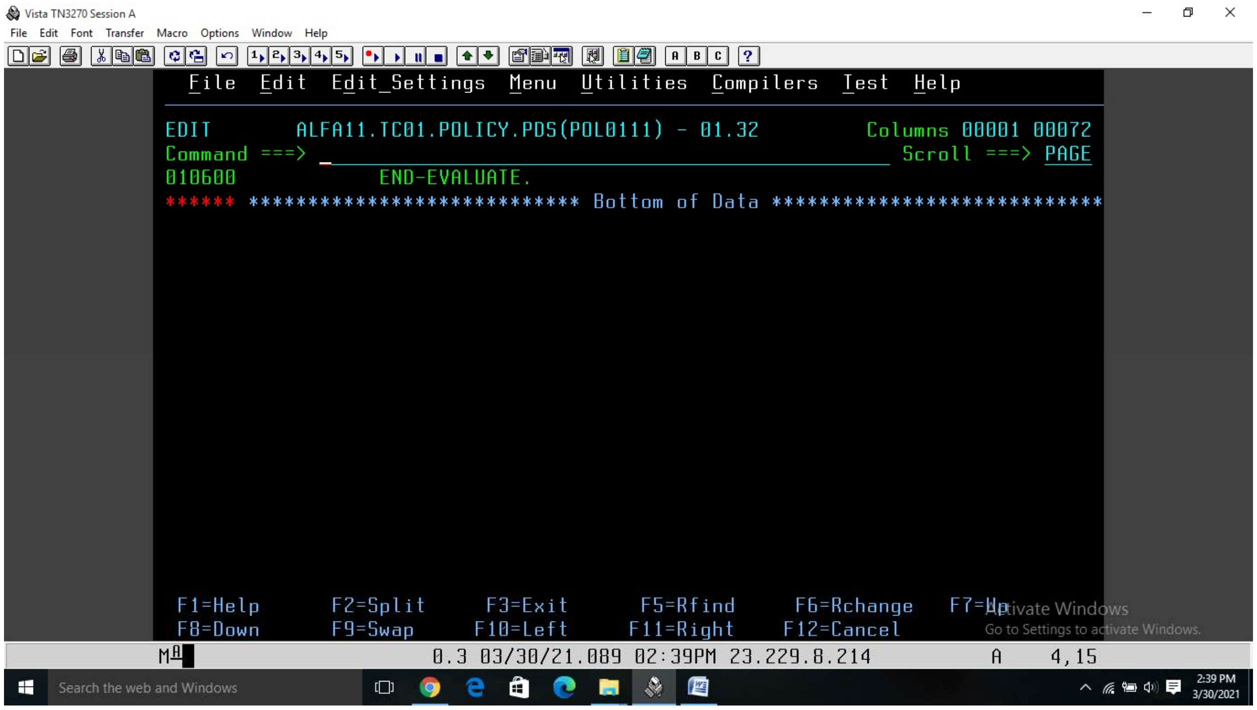
```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       ALFA11.TC01.POLICY.PDS(POL0111) - 01.32         Columns 00001 00072
Command ===> _                                                Scroll ===> PAGE
          EXEC SQL
          INSERT INTO POLICY_EXP_TABLE VALUES(
                                :HV-POL-HOLDER-ID,
                                :HV-POL-POLICY-ID,
                                :HV-POL-POL-VALUE,
                                :HV-POL-POL-EXP-DATE,
                              :HV-POL-POL-STATUS:IV-T2-POLICY-STATUS)
          END-EXEC.
          EVALUATE SQLCODE
           WHEN 0
             DISPLAY "INSERT SUCCESSFUL"
           WHEN -803
             DISPLAY "DUPLICATE PRIMARY KEY"
           WHEN OTHER
             DISPLAY "INSERT ERROR ", SQLCODE
          END-EVALUATE.
      MOVE-PARA.
           IF HV-HOLD-POL-STATUS = "NULL"
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Fig 7 11 (e)COBOL/DB2 Code



```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       ALFA11.TC01.POLICY.PDS(POL0111) - 01.32         Columns 00001 00072
Command ===> _______________________________________________ Scroll ===> PAGE
008800              MOVE HV-HOLD-HOLDER-ID      TO   HV-POL-HOLDER-ID
008900              MOVE HV-HOLD-POL-ID         TO   HV-POL-POLICY-ID
009000              COMPUTE HV-POL-POL-VALUE = HV-HOLD-POL-TERM
009100                                         * HV-HOLD-POL-PREMIUM
009200              MOVE "2021-03-30"           TO   HV-POL-POL-EXP-DATE
009300              MOVE "EXPIRED" TO HV-POL-POL-STATUS
009400           ELSE
009500              DISPLAY "STATUS IS NOT NULL"
009600           END-IF.
009700       CLOSE-PARA.
009800           EXEC SQL
009900              CLOSE C1
010000           END-EXEC.
010100           EVALUATE SQLCODE
010200              WHEN 0
010300                DISPLAY "CURSOR HAS BEEN CLOSED"
010400              WHEN OTHER
010500                DISPLAY "CURSOR CLOSE ERROR ", SQLCODE
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Fig 7 12 (f)COBOL/DB2 Code



```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       ALFA11.TC01.POLICY.PDS(POL0111) - 01.32         Columns 00001 00072
Command ===> _______________________________________________ Scroll ===> PAGE
010600           END-EVALUATE.
****** **************************** Bottom of Data ****************************
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Fig 7 13 (g)COBOL/DB2 Code

CONCLUSION

I have successfully completed my training at DXC Technology via vendors like IIHT and NIIT. I have learned about basic fundamentals of Computer Science. My specialization during training was Mainframe Technology. In Mainframe I have learned about its different types of operating systems. I have also learned about entering commands in the operating system using environment like TSO and ISPF. Further I have learned about Job Control Language which is used for submitting COBOL codes to the mainframe computer operating system. I have learned about various datasets in VSAM data management. I have learned about COBOL which is a business oriented language and also a very simple English like language. Hence COBOL codes are easy to understand. Later I have learned about Embedded SQL Commands in COBOL/DB2 programs. I have done case studies on COBOL and DB2.

Mainframe computers are now an integral part of most of the world's major organizations' everyday operations. Despite the fact that other types of computers are widely employed in business in various capacities, the mainframe continues to have a valued position in today's e-business environment. Data at the nation level is handled by mainframes business in industries like banking, investment, health care, insurance, utilities, and administration.

REFERENCES

1. DXC Technology [Online] Available : *https://www.dxc.technology/* (Visited on May 21, 2020)
2. Tuteja, Maneela, and Gaurav Dubey. "A research study on importance of testing and quality assurance in software development life cycle (SDLC) models." *International Journal of Soft Computing and Engineering (IJSCE)* 2, no. 3 (2012): 251-257.
3. Ebbers, Mike, Wolfgang Bosch, Hans Joachim Ebert, Helmut Hellner, Jerry Johnston, Marco Kroll, Wilhelm Mild et al. *Introduction to the New Mainframe: IBM Z/VSE Basics*. IBM Redbooks, 2016.
4. Barron, David W., and I. R. Jackson. "The evolution of job control languages." *Software: Practice and Experience* 2, no. 2 (1972): 143-164.
5. Sammet, Jean E. "The early history of COBOL." In *History of Programming Languages*, pp. 199-243. 1978.
6. Java Point [Online] Available : *https://www.javatpoint.com/ - Software Development Life* (Visited on May 22, 2020)
7. Tutorials Point Java Point [Online] Available : *https://www.tutorialspoint.com/ - JCL Documentation and DB2 Documentation* (Visited on May 22, 2020)