VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

A Mini Project Report on

Human Resource Record using Indexing

Submitted in partial fulfillment of the requirements as a part of the File Structures Lab of VI semester for the award of degree of ***Bachelor of Engineering*** *in* ***Information Science and Engineering****, Visvesvaraya Technological University, Belagavi*

Submitted by

SURAJ CHAUDHARY
1RN19IS162

SUMIT KUSHWAHA
1RN19IS159

Faculty Incharge

Ms. Sudha V
Assistant Professor
Dept. of ISE, RNSIT

Department of Information Science and Engineering

RNS Institute of Technology

Channasandra, Dr. Vishnuvardhan Road, R R Nagar Post
Bengaluru – 560 098

2021 – 2022

RNS Institute of Technology

Channasandra, Dr.Vishnuvardhan Road, RR Nagar Post,

Bengaluru – 560 098

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the mini project report entitled **Human Resource Record using Indexing** has been successfully completed by **SURAJ CHAUDHARY** bearing USN **1RN19IS162** and **SUMIT KUSHWAHA** bearing USN **1RN19IS159**, presently VI semester students of **RNS Institute of Technology** in partial fulfillment of the requirements as a part of the **FILE STRUCTURES** Laboratory for the award of the degree of ***Bachelor of Engineering in Information Science and Engineering*** under **Visvesvaraya Technological University, Belagavi** during academic year **2021 – 22**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements as a part of File structures Laboratory for the said degree.

Ms. Sudha V	**Dr. Suresh L**
Faculty Incharge	Professor and Head of Department
Assistant Professor	
Dept. of ISE, RNSIT	Dept. of ISE, RNSIT

External Viva

	Name of the Examiners	**Signature with date**
1.		
2.		

ABSTRACT

This project titled “Human-Resources Records using Indexing” hasbeen done using Eclipse IDE with the platform Windows and language Java. The database used for the project is ‘Human Resources’ records. The project mainly focuses on building the index for the records which is fed in CSV format file, then various operations with a menu choice is displayed to the end user, such as Insert, Search, Deleteand Modify. For the purpose of searching efficiently, binary search algorithm is being used. The inserted record will be initially packed and then will be put in the record file. The user can also Unpack all the records in the file which will be displayed.

The index files generated comprises of a key value and its respective position in the record file. This position will be used for various operations.

So, Indexing is a ‘way to optimize the performance of file access by minimizing the number of disk accesses required to process the required data’.

ACKNOWLEDGMENT

The fulfillment and rapture that go with the fruitful finishing of any assignment would be inadequate without the specifying the people who made it conceivable, whose steady direction and support delegated the endeavors with success.

We would like to profoundly thank **Management** of **RNS Institute of Technology** for providing such a healthy environment to carry out this Project work.

We would like to express our thanks to our Principal **Dr. M K Venkatesha** for his support and inspired me towards the attainment of knowledge.

We wish to place on record our words of gratitude to **Dr. Suresh L,** Professor and Head of the Department, Information Science and Engineering, for being the enzyme and master mind behind our Project work.

We would like to express our profound and cordial gratitude to our Faculty incharge, **Ms. Sudha V,** Assistant Professor, Department of Information Science and Engineering for her valuable guidance, constructive comments and continuous encouragement throughout the Project work.

We would like to thank all other teaching and non-teaching staff of Information Science & Engineering who have directly or indirectly helped us to carry out the project work.

And last, we would hereby acknowledge and thank our parents who have been a source of inspiration and also instrumental in carrying out this Project work.

SURAJ CHAUDHARY

USN: 1RN19IS162

SUMIT KUSHWAHA

USN: 1RN19IS159

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

Human Resource Record using indexing technique is used to store the employee resource details such as the employee id, name of the employee and other personal details of employees. The administrator can create, delete, search and modify employee details.

1.1 Introduction to File Structure

A file structure is a combination of representations for data in files and of operations for accessing the data. A file structure allows applications to read, write, and modify data. It might also support finding the data that matches some search criteria or reading through the data in some particular order. An improvement in file structure design may make an application hundreds of times faster. The details of the representation of the data and the implementation of the operations determine the efficiency of the file structure for particular applications.

1.1.1 History

Early work with files presumed that files were on tape, since most files were. Access was sequential, and the cost of access grew in direct proportion, to the size of the file. As files grew intolerably large for unaided sequential access and as storage devices such as hard disks became available, indexes were added to files. The indexes made it possible to keep a list of keys and pointers in a smaller file that could be searched more quickly. With key and pointer, the user had direct access to the large, primary file. But simple indexes had some of the same sequential flaws as the data file, and as the indexes grew, they too became difficult to manage especially for dynamic files in which the set of keys changes.

In the early 1960's, the idea of applying tree structures emerged. But trees can grow very unevenly as records are added and deleted, resulting in long searches requiring many disk accesses to find a record. In 1963, researchers developed an elegant, self-adjusting binary tree structure, called AVL tree, for data in memory. The problem was that, even with a balanced binary tree, dozens of accesses were required to find a record in even moderate-sized files. A method was needed to keep a tree balanced when each node of thee tree was not a single record, as in a binary tree, but a file block containing dozens, perhaps even hundreds, of records .

1.1.2 About the file

When we talk about a file on disk or tape, we refer to a particular collection of bytes stored there. A file, when the word is used in this sense, physically exists. A disk drive may contain hundreds, even thousands of these physical files. From the standpoint of an application program, a file is somewhat like a telephone line connection to a telephone network. The program can receive bytes through this phone line or send bytes down it, but it knows nothing about where these bytes come from or where they go. The program knows only about its end of the line. Even though there may be thousands of physical files on a disk, a single program is usually limited to the use of only about 20 files.

The application program relies on the OS to take care of the details of the telephone switching system. It could be that bytes coming down the line into the program originate from a physical file they come from the keyboard or some other input device. Similarly, bytes the program sends down the line might end up in a file, or they could appear on the terminal screen or some other output device. Although the program doesn't know where the bytes are coming from or where they are going, it does know which line it is using. This line is usually referred to as the logical file, to distinguish it from the physical files on the disk or tape.

1.1.3 Various kinds of storage of fields and records

A field is the smallest, logically meaningful, unit of information in a file.

Field Structures

The four most common methods of adding structure to files to maintain the identity of fields are:

- Force the fields into a predictable length.
- Begin each field with a length indicator.
- Place a delimiter at the end of each field to separate it from the next field
- Use a “keyword=value” expression to identify each field and its contents.

Method 1: Fix the Length of Fields

In the above example, each field is a character array that can hold a string value of some maximum size. The size of the array is 1 larger than the longest string it can hold. Simple arithmetic is sufficient to recover data from the original fields. The disadvantage of this approach is adding all the padding required to bring the fields up to a fixed length, makes the file much larger.

We encounter problems when data is too long to fit into the allocated amount of space. We can solve this by fixing all the fields at lengths that are large enough to cover all cases, but this makes the problem of wasted space in files even worse. Hence, this approach isn't used with data with large amount of variability in length of fields, but where every field is fixed in length if there is very little variation in field lengths.

Method 2: Begin Each Field with a Length Indicator

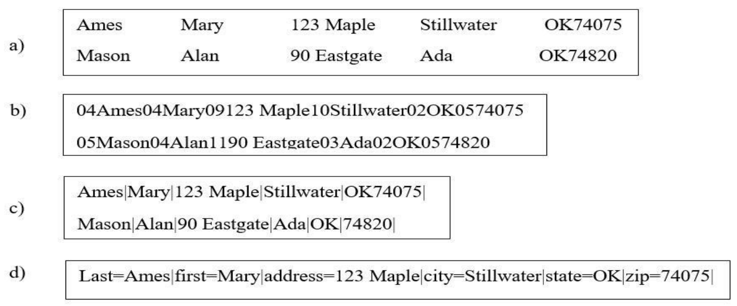
The end of the field can be stored by counting the field length just ahead of the field. If the fields are not too long (less than 256 bytes), it is possible to store the length in a single byte at the start of each field. The fields are referred as length-based.

Method 3: Separate the Fields with Delimiters

The identity of fields can be preserved by separating them with delimiters. It is needed to choose some special character or sequence of characters that will not appear within a field and then insert that delimiter into the file after writing each field. White-space characters (blank, new line, tab) or the vertical bar character, can be used as delimiters.

Method 4: Use a "Keyword=Value"

Expression to Identify Fields This has an advantage the others don't. It is the first structure in which a field provides information about itself. Such self-describing structures can be very useful tools for organizing files in many applications. It is easy to tell which fields are contained in a file.



```
a)  Ames    Mary    123 Maple     Stillwater    OK74075
    Mason   Alan    90 Eastgate   Ada           OK74820

b)  04Ames04Mary09123 Maple10Stillwater02OK0574075
    05Mason04Alan1190 Eastgate03Ada02OK0574820

c)  Ames|Mary|123 Maple|Stillwater|OK74075|
    Mason|Alan|90 Eastgate|Ada|OK|74820|

d)  Last=Ames|first=Mary|address=123 Maple|city=Stillwater|state=OK|zip=74075|
```

Figure 1.1 Four methods for field structures

Record Structures

The five most often used methods for organizing records are:

- Require the records to be predictable number of bytes in length.
- Require the records to be predictable number of fields in length.
- Use a second file to keep track of the beginning byte address for each record.
- Place a delimiter at the end of each record to separate it from the next record.

Method 1: Make the Records a Predictable Number of Bytes (Fixed-Length Record)

A fixed-length record file is one in which each record contains the same number of bytes. In the field and record structure shown, we have a fixed number of fields, each with a predetermined length, that combine to make a fixed-length record. Fixing the number of bytes in a record does not imply that the size or number of fields in the record must be fixed. Fixed-length records are often used as containers to hold variable numbers of variable-length fields. It is also possible to mix fixed and variable-length fields within a record.

Method 2: Make Records a Predictable Number of Fields

Rather than specify that each record in a file contains some fixed number of bytes, we can specify that it will contain a fixed number of fields. In the figure below, we have 6 contiguous fields and we can recognize fields simply by counting the fields modulo 6.

Method 3: Begin Each Record with a Length Indicator

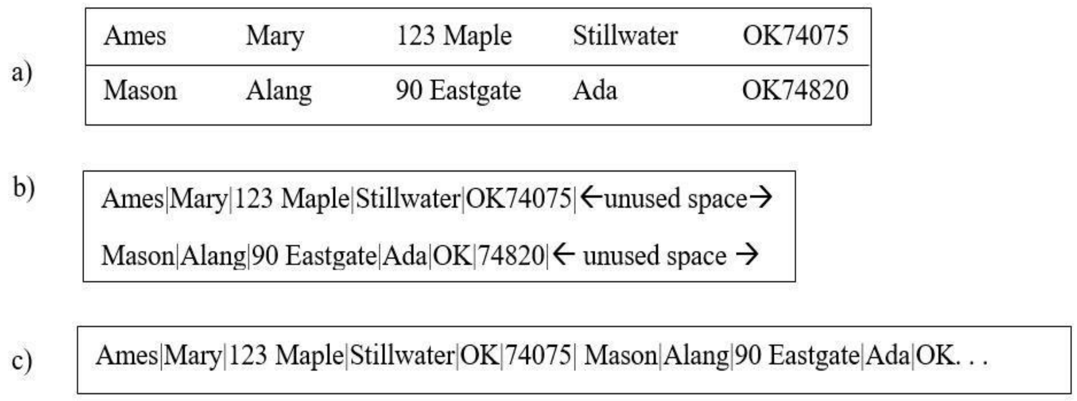
Communication of the length of records by beginning each record with a filed containing an integer that indicates how many bytes there are in the rest of the record. This is commonly used to handle variable-length records.

Method 4: Use an Index to Keep Track of Addresses

Use an index to keep a byte offset for each record in the original file. The byte offset allows us to find the beginning of each successive record and compute the length of each record. We look up the position of a record in the index, then seek to the record in the data file.

Method 5: Place a Delimiter at the End of Each Record

It is analogous to keeping the fields distinct. As with fields, the delimiter character must not get in the way of processing. A common choice of a record delimiter for files that contain readable text is the end-of-line character (carriage return/ new-line pair or, on Unix systems, just a new-line character: \n). Here, use a # character as the record delimiter.


a)

Ames	Mary	123 Maple	Stillwater	OK74075
Mason	Alang	90 Eastgate	Ada	OK74820

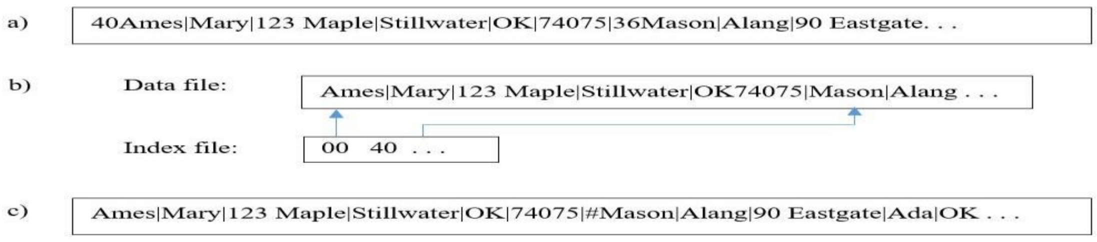
b)

```
Ames|Mary|123 Maple|Stillwater|OK74075|←unused space→
Mason|Alang|90 Eastgate|Ada|OK|74820|← unused space →
```

c)

```
Ames|Mary|123 Maple|Stillwater|OK|74075| Mason|Alang|90 Eastgate|Ada|OK. . .
```

Figure 1.2 Making Records Predictable number of Bytes and Fields


a)

```
40Ames|Mary|123 Maple|Stillwater|OK|74075|36Mason|Alang|90 Eastgate. . .
```

b)

```
Data file:   Ames|Mary|123 Maple|Stillwater|OK74075|Mason|Alang . . .
Index file:  00  40  . . .
```

c)

```
Ames|Mary|123 Maple|Stillwater|OK|74075|#Mason|Alang|90 Eastgate|Ada|OK . . .
```

Figure 1.3 Using Length Indicator, Index and Record Delimiters

1.1.4 Application of File Structure

Relative to other parts of a computer, disks are slow. 1 can pack thousands of megabytes on a disk that fits into a notebook computer. The time it takes to get information from even relatively slow electronic random access memory (RAM) is about 120 nanoseconds. Getting the same information from a typical disk takes 30 milliseconds. So, the disk access is a quarter of a million times longer than a memory access. Hence, disks are very slow compared to memory.

On the other hand, disks provide enormous capacity at much less cost than memory. They also keep the information stored on them when they are turned off. Tension between a disk's relatively slow access time and its enormous, non-volatile capacity, is the driving force behind file structure design. Good file structure design will give us access to all the capacity without making this applications spend a lot of time waiting for the disk.

Chapter 2

SYSTEM ANALYSIS

2.1 Analysis of Application

Human Resource Record can be used by the administrator who can create, delete, search and modify the employee details.

2.2 Structure used to Store the Fields and Records Storing Fields

Fixing the Length of Fields:

In the Human Resource Record, the emp_id field is a character array that can hold a string value of some maximum size. The size of the array is larger than the longest string it can hold. Simple arithmetic is sufficient to recover data from the original fields.

Separating the Fields with Delimiters:

We preserve the identity of fields by separating them with delimiters. We have chosen the vertical bar character, as the delimiter here.

Storing Records

Making Records a Predictable Number of Fields:

In this system, have a fixed number of fields, each with a maximum length, that combine to make a data record. Fixing the number of fields in a record does not imply that the size of fields in the record is fixed. The records are used as containers to hold a mix of fixed and variable-length fields within a record.

2.3 Operations Performed on a File

▪ **Insertion**

The system is used to add employee record containing emp_Id, name, phone number and other personal details. Records with duplicate emp_id fields are not allowed to be inserted. The length of the emp_id is checked to see whether it contains only 6 characters.

▪**Retrieve**

The system can then be used to search for existing records of people. Retrieves the set of record for the given search emp_id. The primary index file is searched to obtain the desired starting byte address, which is then used to seek to the desired data record in any of the files. The details of the requested record, if found, are displayed, with suitable headings on the user's screen. If absent, a "record not found" message is displayed to the user.

- **Delete**

The system can then be used to delete existing records from record file. The reference to a deleted record is removed from index while the deleted record persists in the data file. The requested record, if found, is marked for deletion, a "record deleted" message is displayed, and the reference to it is removed from the primary index file. If absent, a "record not found" message is displayed to the user. The deleted record space is filled with " * " symbol.

- **Modify**

The system can be used to insert the modified employee record in the file. Modify the set of records based on single emp_id. The requested record, if found, is marked for modification, and stores the correct modified record in the file and display the message as "record updated successfully", and the reference to it is removed from the primary indexing file. If absent, a "recordnot found" message is displayed to the user.

2.4 Indexing Used

Simple Indexing

We know that data is stored in the form of records. Every record has a key field, which helps it to be recognized uniquely. Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing not only helps in querying, but can also be used in any algorithm being built to reduce the time taken by that algorithm.

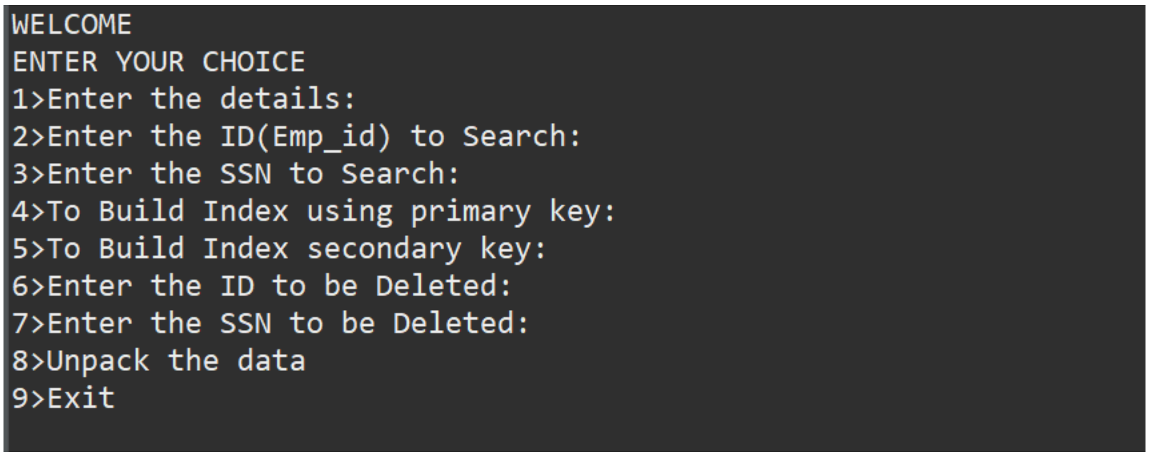
Simple indexes use simple arrays. An index lets us impose order on a file impose order on a file without rearranging the file. Indexes provide multiple access paths multiple access paths to a file— multiple indexes multiple indexes(like library catalog providing search for author, book and title) An index can provide keyed access to variable-length record files.

Chapter 3

SYSTEM DESIGN

3.1 User Interface

The User Interface or UI refers to the interface between the application and the user. Here, the UI is menu-driven, that is, a list of options (menu) is displayed to the user and the user is prompted to enter an integer corresponding to the choice that represents the desired operation to be performed.



```
WELCOME
ENTER YOUR CHOICE
1>Enter the details:
2>Enter the ID(Emp_id) to Search:
3>Enter the SSN to Search:
4>To Build Index using primary key:
5>To Build Index secondary key:
6>Enter the ID to be Deleted:
7>Enter the SSN to be Deleted:
8>Unpack the data
9>Exit
```

Figure 3.2 Menu Screen

3.1.1 Insertion of a Record

The user is given with an option to insert the records into the recordfile. Upon execution of the insertion operation, the new record will be inserted into the record file, primary-index file and also the secondary- index file. This record will be inserted at the end of the file, i.e. it will be appended to these files. After insertion the index files are rebuilt once again.

This insertion operation is carried out by getData() method which reads all the required fields, and then add() method packs all these data and then writes it to corresponding files.

3.1.2 Display all Record

If the operation desired is Display of Records, the user is required to enter 4 as his/her choice, from the menu displayed, after which a new screen is displayed. If there are no records in any file, it display a message as "no records found". For the employee file all the details of each record within the file, with suitable headings, is displayed.

3.1.3 Deleting a record

In delete operation the user has the privilege to delete the records. Deletion of records can again be done based on primary-index or secondary-index.
The record to be deleted will be prefixed with a asterisk (*) symbol, which indicates that the record is deleted and this record will not be considered while unpacking the records.

3.1.4 Searching and modifying the Records

The user can search a record from the record file based on primary- index or secondary-index.
Searching using primary-index file prompts the user to enter the primary key, this key will be used to locate the record in the record file. This search operation makes use of the binary search algorithm in order to efficiently search the primary key and seek its position.
Searching using secondary-index prompts the user to enter the secondary key, this key will be used to locate all the records which contains this secondary key.

The user can also modify the records present in the record file. This modify() method permits the user to modify any field of the recordwhich he has searched.

Chapter 4

IMPLEMENTATION

Implementation is the process of: defining how the system should be built, ensuring that it is operational and meets quality standards. It is a systematic and structured approach for effectively integrating the software based service or component into the requirements of end users.

4.1 About Java

Java is a widely used object-oriented programming language and software platform that runs on billions of devices, including notebook computers, mobile devices, gaming consoles, medical devices and many others. The rules and syntax of Java are based on the C and C++ languages.

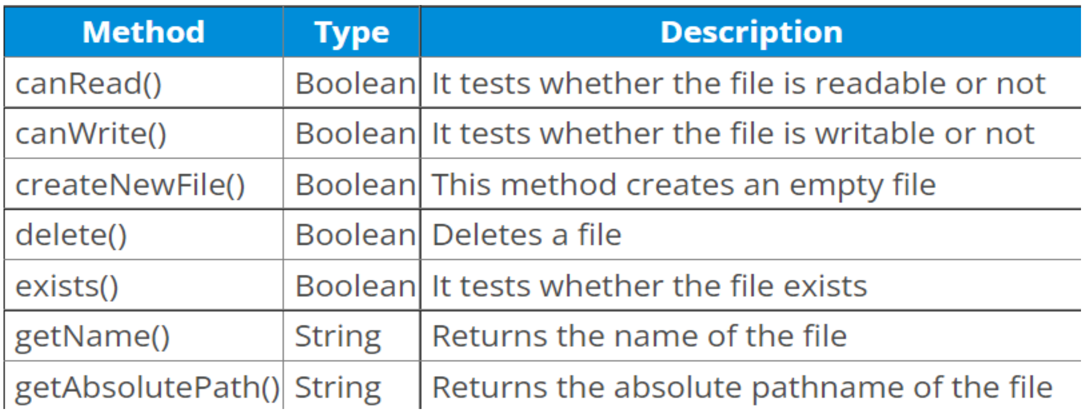
4.1.1 Classes and Objects

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake. A Class is like an object constructor, or a "blueprint" for creating objects.

4.1.2 File Handling

File handling in Java implies reading from and writing data to a file. The File class from the java.io package, allows us to work with different formats of files. In order to use the File class, you need to create an object of the class and specify the filename or directory name.

Below table depicts few methods that are used for performing operations on Java files.



Method	Type	Description
canRead()	Boolean	It tests whether the file is readable or not
canWrite()	Boolean	It tests whether the file is writable or not
createNewFile()	Boolean	This method creates an empty file
delete()	Boolean	Deletes a file
exists()	Boolean	It tests whether the file exists
getName()	String	Returns the name of the file
getAbsolutePath()	String	Returns the absolute pathname of the file

4.2 Pseudo code

4.2.1 Insertion Module Pseudo code

The getData() method is used to add new employee details to the file. Values are inserted into record and index files.

```
public void getData()
{
	@SuppressWarnings("resource")
	Scanner scanner = new Scanner(System.in);
	System.out.println("Enter the Emp_ID: ");
	Emp_ID = scanner.next();
	System.out.println("Enter the First_Name: ");
	First_Name = scanner.next();
	System.out.println("Enter the Last_Name: ");
	Last_Name = scanner.next();
	System.out.println("Enter the Gender: ");
	Gender = scanner.next();
	System.out.println("Enter the E_Mail: ");
	E_Mail = scanner.next();
	System.out.println("Enter the Fathers_Name: ");
	Fathers_Name = scanner.next();
	System.out.println("Enter the Mothers_Name: ");
	Mothers_Name = scanner.next();
	System.out.println("Enter the Date_of_Birth: ");
	Date_of_Birth = scanner.next();
	System.out.println("Enter the Age: ");
	Age = scanner.next();
	System.out.println("Enter the Date_of_Joining: ");
	Date_of_Joining = scanner.next();
	System.out.println("Enter the Salary:  ");
	Salary = scanner.next();
	System.out.println("Enter the SSN:  ");
	SSN = scanner.next();
	System.out.println("Enter the Phone no:  ");
	Phone_No = scanner.next();
}

public void add()
{
	String data = Emp_ID +","+  First_Name +","+ Last_Name +","+
	Gender +","+ E_Mail +","+ Fathers_Name +","+ Mothers_Name +","+
	Date_of_Birth +","+ Age +"," + Date_of_Joining + "," +  Salary +
	"," + SSN + "," + Phone_No ;
	try{
		RandomAccessFile recordfile = new RandomAccessFile
		("C:\\Users\\82522\\eclipse-
		workspace\\FS\\src\\records\\humanresources.txt","rw");
		recordfile.seek(recordfile.length());
```

```
            long pos = recordfile.getFilePointer();
            recordfile.writeBytes(data+"\n");
            recordfile.close();

            RandomAccessFile indexfile = new RandomAccessFile
            ("C:\\Users\\82522\\eclipse-
            workspace\\FS\\src\\index.txt","rw");
            indexfile.seek(indexfile.length());
            indexfile.writeBytes(Emp_ID+","+pos+"\n");
            indexfile.close();

            RandomAccessFile indexfile1 = new RandomAccessFile
            ("C:\\Users\\82522\\eclipse-
            workspace\\FS\\src\\index1.txt","rw");
            indexfile1.seek(indexfile1.length());
            indexfile1.writeBytes(SSN+","+pos+"\n");
            indexfile1.close();
        }
        catch(IOException e){
            System.out.println(e);
        }

 }
```

4.2.2 Deletion Module Pseudo code

The delete() method deletes the record from the file, the user is prompted for the emp_id, whosematching record is to be deleted. The deleted record space is filled with "*" symbol.

Function to remove a record with emp_id as primary key

```
 public void delete() throws IOException {
        System.out.println("Enter the primary key to delete record ");
        @SuppressWarnings("resource")
        Scanner scanner = new Scanner(System.in);
        String prikey = scanner.next();
        int pos = binarySearch(sI, 0, reccount-1, prikey);
        System.out.println("WAIT FOR FEW SECONDS....: ");
        if (pos == -1) {
            System.out.println("Record not found in the record file");
            return ;
        }
        RandomAccessFile recordfile = new RandomAccessFile
        ("C:\\Users\\82522\\eclipse-
        workspace\\FS\\src\\records\\humanresources.txt","rw");
        try {
            recordfile.seek(Long.parseLong(sI[pos].getRecPos()));
             recordfile.writeBytes("*");
             recordfile.close();
         }
        catch (NumberFormatException e) {
              e.printStackTrace();
        }
        catch (IOException e) {
              e.printStackTrace();
        }

  }
```

4.2.3 Search and Modify Module Pseudocode

The search() function searches the file for the record of the emp_id entered by the user and displays the corresponding record details.

Function to search a record by passing address and key

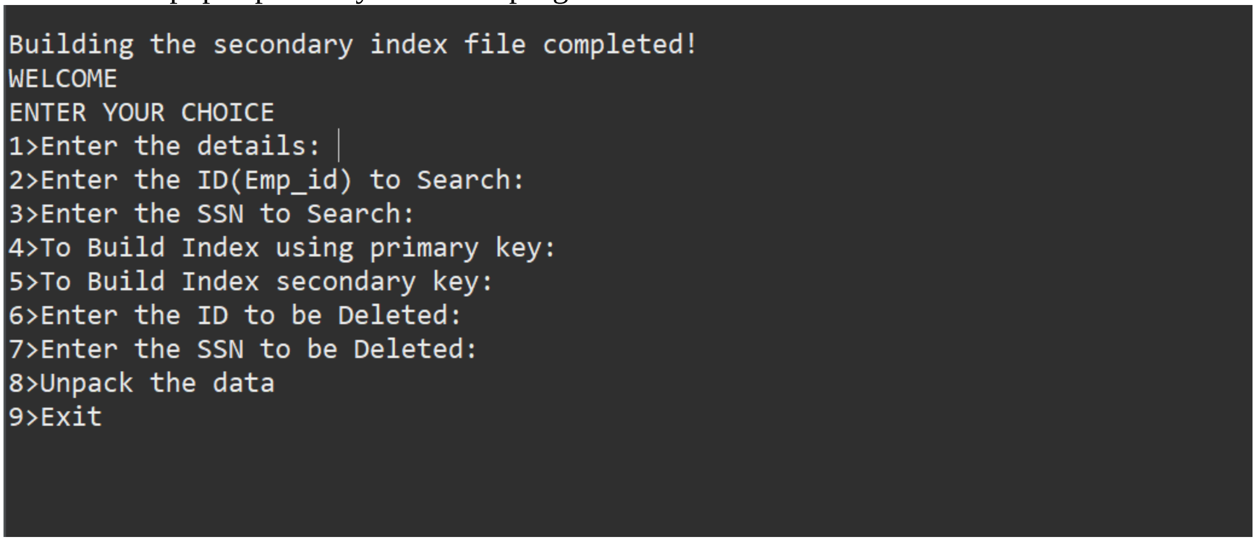
```
  int binarySearch(demo2 s[], int l, int r, String ssn) {
        int mid;
        while (l<=r) {
              mid = (l+r)/2;
              if(s[mid].getseckey().compareTo(ssn)==0) {return mid;}
              if(s[mid].getseckey().compareTo(ssn)<0) l = mid + 1;
              if(s[mid].getseckey().compareTo(ssn)>0) r = mid - 1;
        }
        return -1;

  }


  public void search(){
        System.out.println("Enter the SSN to search: ");
        @SuppressWarnings("resource")
        Scanner scanner = new Scanner(System.in);
        String ssn = scanner.next();
        int oripos = binarySearch(sI, 0, reccount-1, ssn);
        if (oripos == -1) {
              System.out.println("Record not found in the record file");
             return ;
        }
        int pos = oripos;
        do {
              readFile(pos);
              pos--;
              if (pos < 0)
                    break;
        }while(sI[pos].getseckey().compareTo(ssn)==0);
        pos = oripos + 1 ;
        while(sI[pos].getseckey().compareTo(ssn)==0 && pos > reccount-1)
              readFile(pos);
              pos++;
        }
        System.out.println("Do you want to modify????? Y/N");
        String modi = scanner.next();
        if ( modi.equalsIgnoreCase("y"))
        {
                System.out.println("What do you want to change");
                System.out.println("Enter your choice");
                System.out.println("1.Emp_ID \n 2. First_Name \n
                3.Last_Name");

                int choice = scanner.nextInt();
                switch(choice)
                {
                        case 1 :System.out.println("Enter the Emp_ID ")
                                break;

                        case 2:System.out.println("Enter the First_Name");
                                break;

                        case 3: System.out.println("Enter the Last_Name");
                                break;
                }
                long pointer = recordfile.getFilePointer();
                String pack = tmp_Emp_ID +","+  tmp_First_Name +","+
                tmp_Last_Name +","+ tmp_Gender +","+ tmp_E_Mail +","+
                tmp_Fathers_Name +","+ tmp_Mothers_Name +","+
                tmp_Date_of_Birth +","+ tmp_Age +"," +
                tmp_Date_of_Joining + "," +  tmp_Salary + "," + tmp_SSN +
                "," + tmp_Phone_No ;
                if( pack.length()>length)
                {
                        if(reccount==1) {
                                pointer = 0;
                        }
                         else {
                                pointer = pointer-(length+1);
                        }
                        recordfile.seek(pointer);
                        recordfile.writeBytes("*");//deleting a record//marking
                        it as deleted
                        recordfile.seek(recordfile.length());
                        recordfile.writeBytes(pack+"\n");
                        recordfile.close();
                }
                else {
                        if(reccount==1) {
                                pointer = 0;
                        }
                         else {
                                pointer = pointer-(length+1);
                         }
                        recordfile.seek(pointer);
                        recordfile.writeBytes(pack);
                        recordfile.close();
                }
        }
        else{
                System.out.println("ok done");
        }
}
```

4.3 Discussion of Results

All the menu options provided in the application and its operations have been presented in as screen shots

4.3.1 Home menu

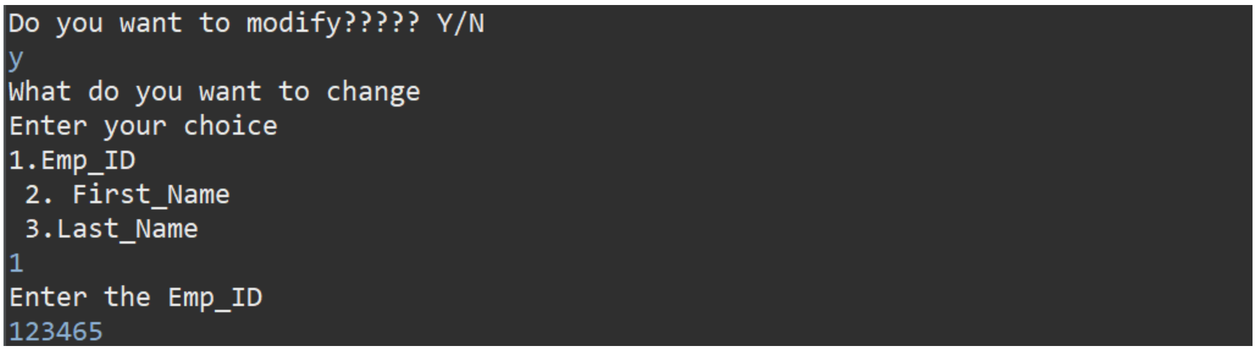
This menu pops up when you run the program.



```
Building the secondary index file completed!
WELCOME
ENTER YOUR CHOICE
1>Enter the details:
2>Enter the ID(Emp_id) to Search:
3>Enter the SSN to Search:
4>To Build Index using primary key:
5>To Build Index secondary key:
6>Enter the ID to be Deleted:
7>Enter the SSN to be Deleted:
8>Unpack the data
9>Exit
```

Figure 4.1: Home Menu

4.3.2 Modification of record

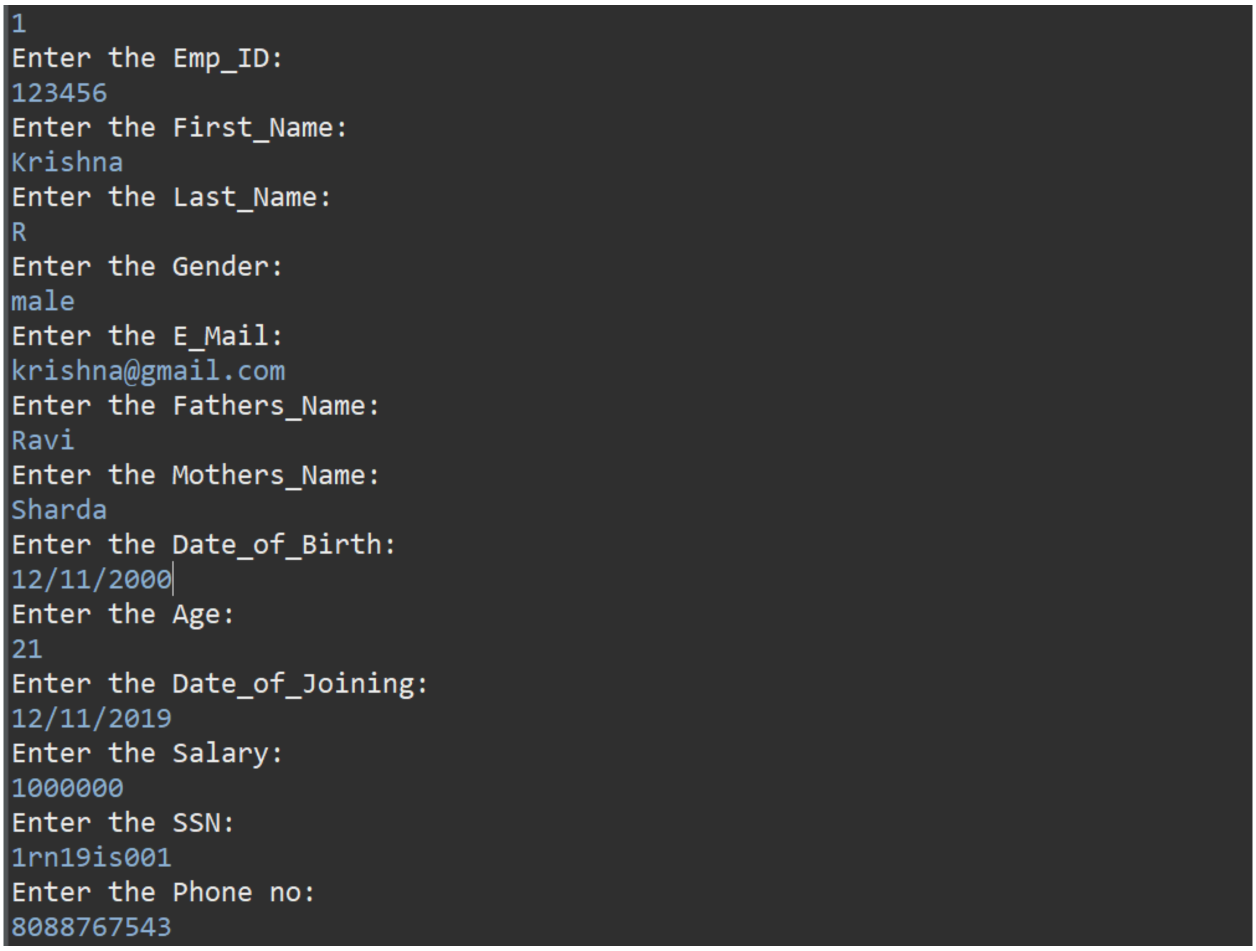


```
Do you want to modify????? Y/N
y
What do you want to change
Enter your choice
1.Emp_ID
 2. First_Name
 3.Last_Name
1
Enter the Emp_ID
123465
```

Figure. 4.8 Modification of record

4.3.3 Adding a record

This is how one adds a record to the file.

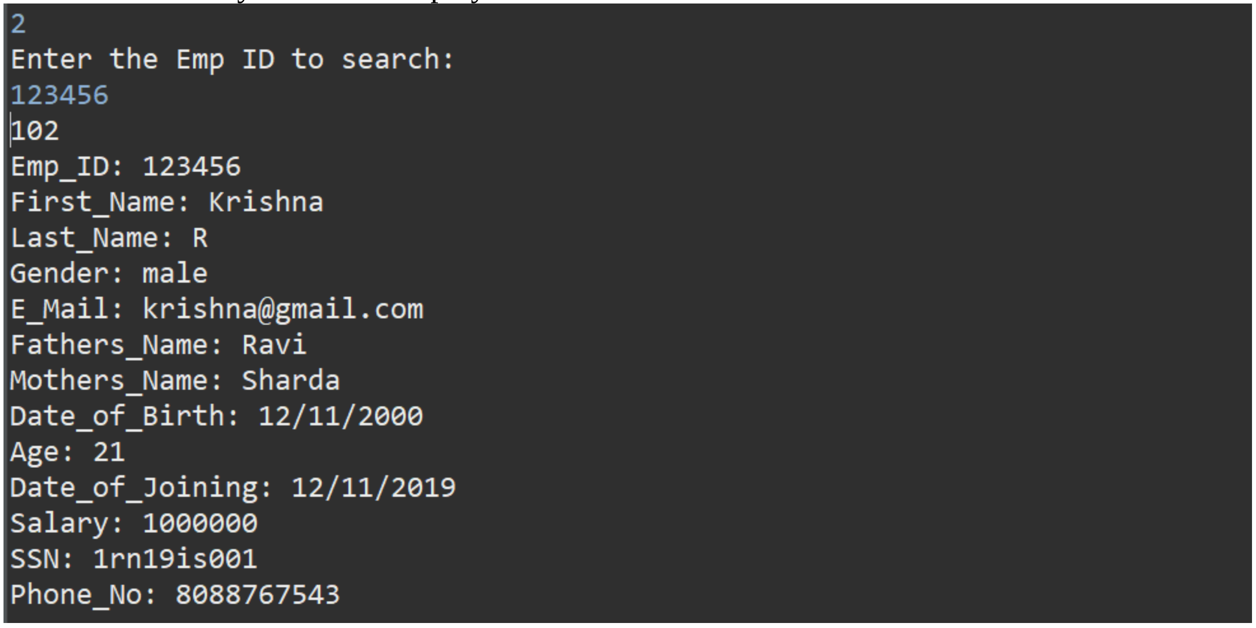


```
1
Enter the Emp_ID:
123456
Enter the First_Name:
Krishna
Enter the Last_Name:
R
Enter the Gender:
male
Enter the E_Mail:
krishna@gmail.com
Enter the Fathers_Name:
Ravi
Enter the Mothers_Name:
Sharda
Enter the Date_of_Birth:
12/11/2000
Enter the Age:
21
Enter the Date_of_Joining:
12/11/2019
Enter the Salary:
1000000
Enter the SSN:
1rn19is001
Enter the Phone no:
8088767543
```

Figure 4.2 Insertion of record

4.3.4 Searching of record

The record with key 123456 is displayed as shown.

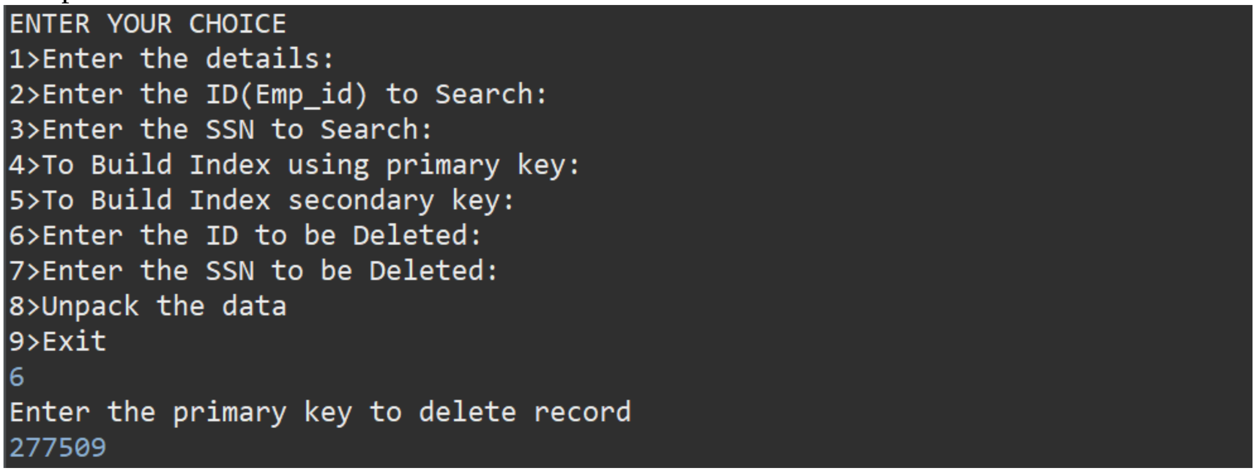


```
2
Enter the Emp ID to search:
123456
102
Emp_ID: 123456
First_Name: Krishna
Last_Name: R
Gender: male
E_Mail: krishna@gmail.com
Fathers_Name: Ravi
Mothers_Name: Sharda
Date_of_Birth: 12/11/2000
Age: 21
Date_of_Joining: 12/11/2019
Salary: 1000000
SSN: 1rn19is001
Phone_No: 8088767543
```

Figure: 4.4 Searching for record

4.3.5 Deletion of record

The process of deletion of a record from the file.

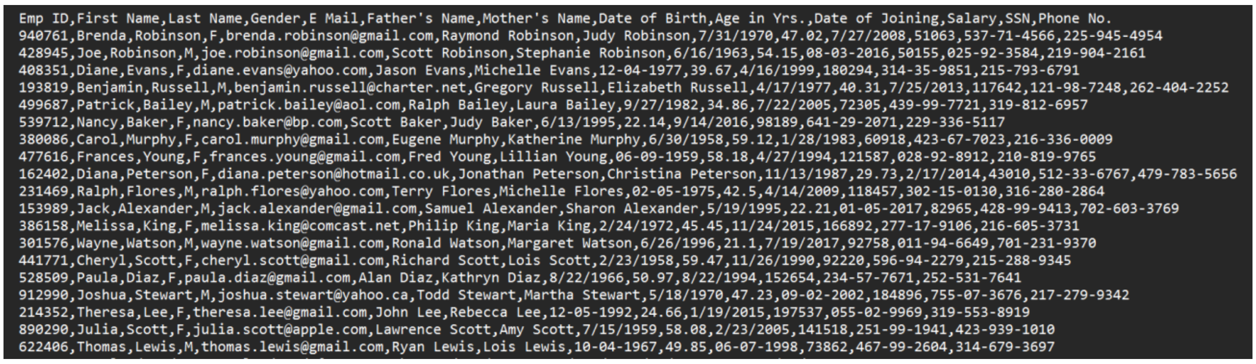


```
ENTER YOUR CHOICE
1>Enter the details:
2>Enter the ID(Emp_id) to Search:
3>Enter the SSN to Search:
4>To Build Index using primary key:
5>To Build Index secondary key:
6>Enter the ID to be Deleted:
7>Enter the SSN to be Deleted:
8>Unpack the data
9>Exit
6
Enter the primary key to delete record
277509
```

Figure. 4.5 Deletion of record

4.3.6 File Contents

This is how the data is stored in the record file.

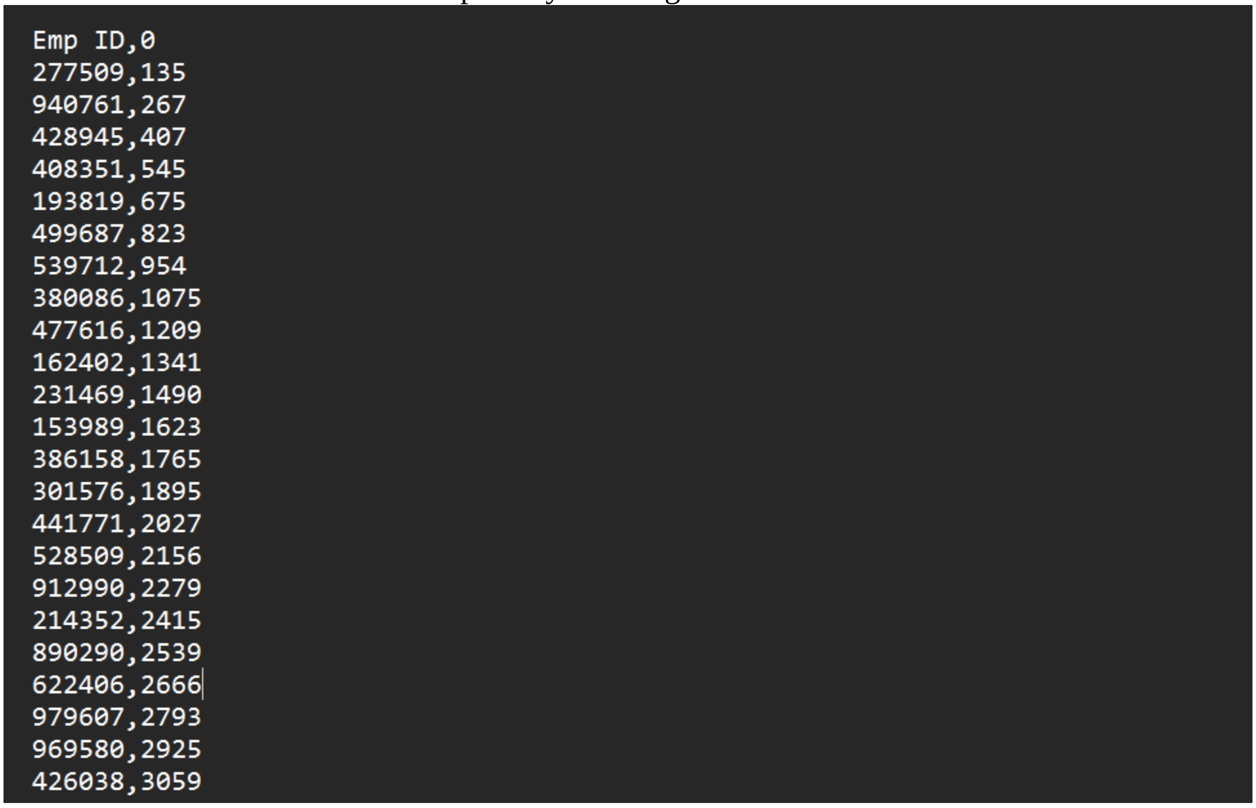


```
Emp ID,First Name,Last Name,Gender,E Mail,Father's Name,Mother's Name,Date of Birth,Age in Yrs.,Date of Joining,Salary,SSN,Phone No.
940761,Brenda,Robinson,F,brenda.robinson@gmail.com,Raymond Robinson,Judy Robinson,7/31/1970,47.02,7/27/2008,51063,537-71-4566,225-945-4954
428945,Joe,Robinson,M,joe.robinson@gmail.com,Scott Robinson,Stephanie Robinson,6/16/1963,54.15,08-03-2016,50155,025-92-3584,219-904-2161
408351,Diane,Evans,F,diane.evans@yahoo.com,Jason Evans,Michelle Evans,12-04-1977,39.67,4/16/1999,180294,314-35-9851,215-793-6791
193819,Benjamin,Russell,M,benjamin.russell@charter.net,Gregory Russell,Elizabeth Russell,4/17/1977,40.31,7/25/2013,117642,121-98-7248,262-404-2252
499687,Patrick,Bailey,M,patrick.bailey@aol.com,Ralph Bailey,Laura Bailey,9/27/1982,34.86,7/22/2005,72305,439-99-7721,319-812-6957
539712,Nancy,Baker,F,nancy.baker@bp.com,Scott Baker,Judy Baker,6/13/1995,22.14,9/14/2016,98189,641-29-2071,229-336-5117
380086,Carol,Murphy,F,carol.murphy@gmail.com,Eugene Murphy,Katherine Murphy,6/30/1958,59.12,1/28/1983,60918,423-67-7023,216-336-0009
477616,Frances,Young,F,frances.young@gmail.com,Fred Young,Lillian Young,06-09-1959,58.18,4/27/1994,121587,028-92-8912,210-819-9765
162402,Diana,Peterson,F,diana.peterson@hotmail.co.uk,Jonathan Peterson,Christina Peterson,11/13/1987,29.73,2/17/2014,43010,512-33-6767,479-783-5656
231469,Ralph,Flores,M,ralph.flores@yahoo.com,Terry Flores,Michelle Flores,02-05-1975,42.5,4/14/2009,118457,302-15-0130,316-280-2864
153989,Jack,Alexander,M,jack.alexander@gmail.com,Samuel Alexander,Sharon Alexander,5/19/1995,22.21,01-05-2017,82965,428-99-9413,702-603-3769
386158,Melissa,King,F,melissa.king@comcast.net,Philip King,Maria King,2/24/1972,45.45,11/24/2015,166892,277-17-9106,216-605-3731
301576,Wayne,Watson,M,wayne.watson@gmail.com,Ronald Watson,Margaret Watson,6/26/1996,21.1,7/19/2017,92758,011-94-6649,701-231-9370
441771,Cheryl,Scott,F,cheryl.scott@gmail.com,Richard Scott,Lois Scott,2/23/1958,59.47,11/26/1990,92220,596-94-2279,215-288-9345
528509,Paula,Diaz,F,paula.diaz@gmail.com,Alan Diaz,Kathryn Diaz,8/22/1966,50.97,8/22/1994,152654,234-57-7671,252-531-7641
912990,Joshua,Stewart,M,joshua.stewart@yahoo.ca,Todd Stewart,Martha Stewart,5/18/1970,47.23,09-02-2002,184896,755-07-3676,217-279-9342
214352,Theresa,Lee,F,theresa.lee@gmail.com,John Lee,Rebecca Lee,12-05-1992,24.66,1/19/2015,197537,055-02-9969,319-553-8919
890290,Julia,Scott,F,julia.scott@apple.com,Lawrence Scott,Amy Scott,7/15/1959,58.08,2/23/2005,141518,251-99-1941,423-939-1010
622406,Thomas,Lewis,M,thomas.lewis@gmail.com,Ryan Lewis,Lois Lewis,10-04-1967,49.85,06-07-1998,73862,467-99-2604,314-679-3697
```

Figure. 4.10 Record File

4.3.7 Index File Contents

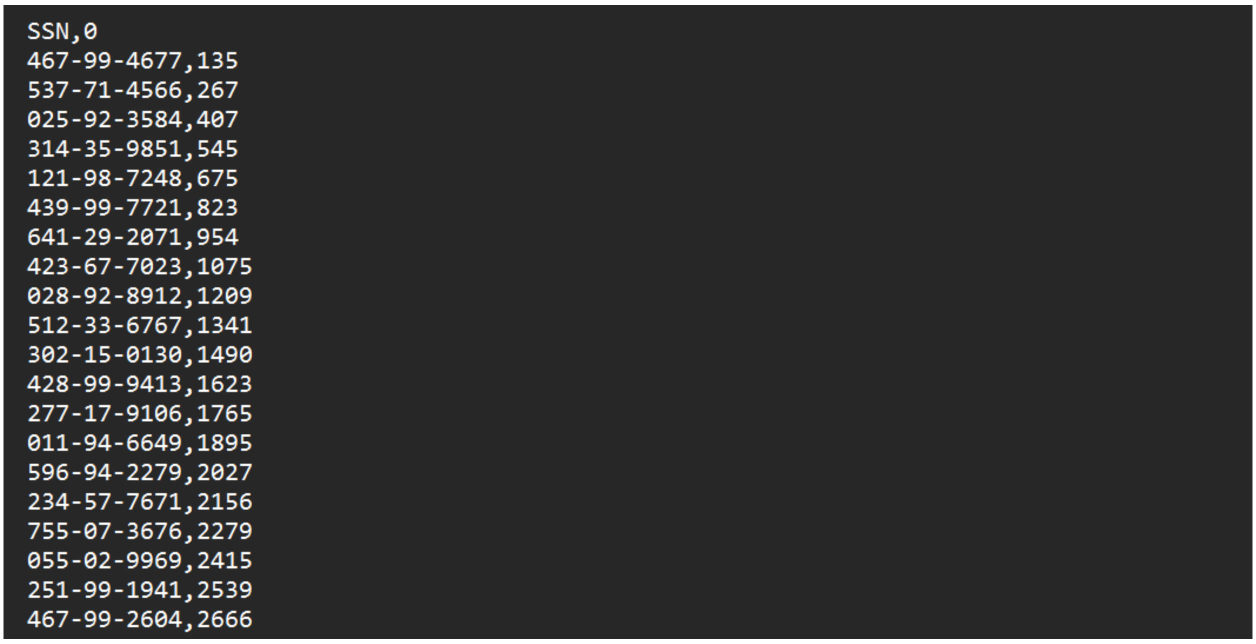
This is how the data is stored in the primary indexing file.



```
Emp ID,0
277509,135
940761,267
428945,407
408351,545
193819,675
499687,823
539712,954
380086,1075
477616,1209
162402,1341
231469,1490
153989,1623
386158,1765
301576,1895
441771,2027
528509,2156
912990,2279
214352,2415
890290,2539
622406,2666
979607,2793
969580,2925
426038,3059
```

Figure. 4.8 Primary Index file

This is how the data is stored in the secondary indexing file.



```
SSN,0
467-99-4677,135
537-71-4566,267
025-92-3584,407
314-35-9851,545
121-98-7248,675
439-99-7721,823
641-29-2071,954
423-67-7023,1075
028-92-8912,1209
512-33-6767,1341
302-15-0130,1490
428-99-9413,1623
277-17-9106,1765
011-94-6649,1895
596-94-2279,2027
234-57-7671,2156
755-07-3676,2279
055-02-9969,2415
251-99-1941,2539
467-99-2604,2666
```

Figure. 4.8 Secondary Index file

Chapter 5

CONCLUSION AND FUTURE ENHANCEMENTS

The main objective of Indexing is to optimize the performance of file access by minimizing the number of disk accesses required to process the required data' has been achieved.

Indexes are used to quickly locate data without having to search every row in a database, every time a database record is accessed.

In this project, building the index files takes up a lot of time depending on the size of the record file. The larger the size of the record file, the more time it takes to build the index files. In case when the size of the index file is too large and exceeds the size of the main memory, then handling the index file itself will not be possible.

Therefore, we can rely on other data structures such as B- trees, Hashing, Extensible Hashing etc. to build index files for large record files efficiently and within short period of time.

REFERENCES

[1] Michael J.Folk, Bill Zoellick, Greg Riccardi: File Structures-An Object-Oriented Approach in C++, PEARSON, 3rd Edition, Pearson Education, 1998.

[2] K.R.Venugoapal, , K.G.Srinivas, P.M.Krishnaraj:File structures Using C++,TATA McGraw Hill 2008.

[3] www.wikipedia.org

[4] www.w3cSchool.com/php?

[5] www.tutorialspoint.com/sdlc/

[6] www.geeksforgeeks.org