VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

A Mini Project Report on

Voter ID Management System B+Tree Indexing Technique

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

AMIT KUMAR SAH
1RN17IS010

K.HEMANTH RAJU
1RN17IS046

Faculty Incharge
Mr. Manoranjan S R
Assistant Professor

Coordinator
Mr. Santhosh Kumar
Assistant Professor

Department of Information Science and Engineering
RNS Institute of Technology
Channasandra, Dr. Vishnuvardhan Road, R R Nagar Post
Bengaluru – 560 098

2020-2021

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 **VOTER ID MANAGEMENT SYSTEM USING B+TREE INDEXING TECHNIQUE** has been successfully completed by **AMIT KUMAR SAH** bearing USN **1RN17IS010** and **K.HEMANTH RAJU** bearing USN **1RN17IS046**, 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 **2020 – 2021**.

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.

Mr. Manoranjan S R	**Mr. Santhosh Kumar**	**Dr. M V Sudhamani**
Faculty Incharge	Coordinator	Professor and HoD
Assistant Professor	Assistant Professor	

External Viva

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

ABSTRACT

The VOTER ID System provides a user-friendly, interactive Menu Driven Interface (MDI) based on local file systems. All data is stored in files on disk. The system uses file handles to access the files. This System is used by the person to view his/her VOTER ID details. The system is initially used to add person records containing the V_ID, name, dob, address, gender. The system can then be used to search, delete, modify or display existing records of all the citizens .In the VOTER ID System, the V_ID field is a character array that can hold a numeric value of some maximum size. The size of the array is larger than the longest string it can hold. We preserve the identity of fields by separating them with delimiters. We have chosen the vertical bar character, as the delimiter here.

In this System, we 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. We have 5 contiguous fields. We use a B+ TREE of indexes to keep byte offsets for each record in the original file. The byte offsets allow 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 B+TREE, and then seek to the record in the data file. Our choice of a record delimiter for the data files is the end-of-line (new-line) character (\n).

A B+TREE of simple indexes on the primary key is used to provide direct access to data records. Each node in the B+TREE consists of a primary key and reference pair of fields. The primary key field is the V_ID field while the reference field is the starting byte offset of the matching record in the data file.

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 mini project work.

We would like to thank our beloved Director **Dr. H N Shivashankar** for his confidence feeling words and support for providing facilities throughout the course.

We would like to express my 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 my words of gratitude to **Dr. M V Sudhamani,** Professor and Head of the Department, Information Science and Engineering, for being the enzyme, master mind behind my mini project work for guiding me and helping me out with the report.

We would like to express our profound and cordial gratitude to our Faculty Incharge **Mr. Manoranjan S R, Assistant Professor**, Department of Information Science and Engineering for his valuable guidance, constructive comments and continuous encouragement throughout the mini project work.

We would like to express our sincere gratitude to our Coordinator **Mr. Santhosh Kumar,** Assistant Professor, Department of Information Science and Engineering, for his constant support and guidance.

AMIT KUMAR SAH

1RN17IS010

K.HEMANTH

RAJU

1RN17IS046

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ASCII	American Standard Code for Information Interchange
AVL	Adelson-Velskii and Landis
MDI	Menu Driven Interface
OS	Operating System
RAM	Random Access Memory
VID	Voter Identity Number
UI	User Interface
Addr	Address

LIST OF FIGURES

Chapter 1

INTRODUCTION

The Voter ID Management System provides a user-friendly, interactive Menu Driven Interface (MDI). All data is stored in files for persistence. The mini project uses 2 files: An Index file, to store the primary index, a Data file, to store voter records like voter ID, name, address, dob etc. and a secondary index file.

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 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.

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 key changes. In 1963, researches developed and elegant self adjusting binary tree structure, called AVL tree, for data in memory, 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 took 10 years until a solution emerged in the form of a B-Tree.

Whereas AVL trees grow from the top down as records were added, B-Trees grew from the bottom up. B-Trees provided excellent access performance, but there was a cost: no longer could a file be accessed sequentially with efficiency. The problem was solved by adding a linked list structure at the bottom level of the B-Tree. The combination of a B-Tree and a sequential linked list is called a B+ tree.

B-Trees and B+ trees provide access times that grow in proportion to $\log_k N$, where N is the number of entries in the file and k is the number of entries indexed in a single block of the B+ Tree structure. This means that B+ Trees can guarantee that you can find 1 file entry among millions with only 3 or 4 trips to the disk. Further, B+Trees guarantee that as you add and delete entries, performance stays about the same.

Hashing is a good way to get what we want with a single request, with files that do not change size greatly over time. Hashed indexes were used to provide fast access to files. But until recently, hashing did not work well with volatile, dynamic files. Extendible dynamic hashing can retrieve information with 1 or at most 2 disk accesses, no matter how big the file became.

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 a system 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. A file is a operation of computer which stores data, information, message, settings, or commands used with the computer program. It is created using system program on the computer.

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.

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 as shown in Fig. 1.1 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

We can count to the end of a field by storing 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. We refer to these fields as length-based.

Method 3: Separate the Fields with Delimiters

We can preserve the identity of fields by separating them with delimiters. All we need to do is 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

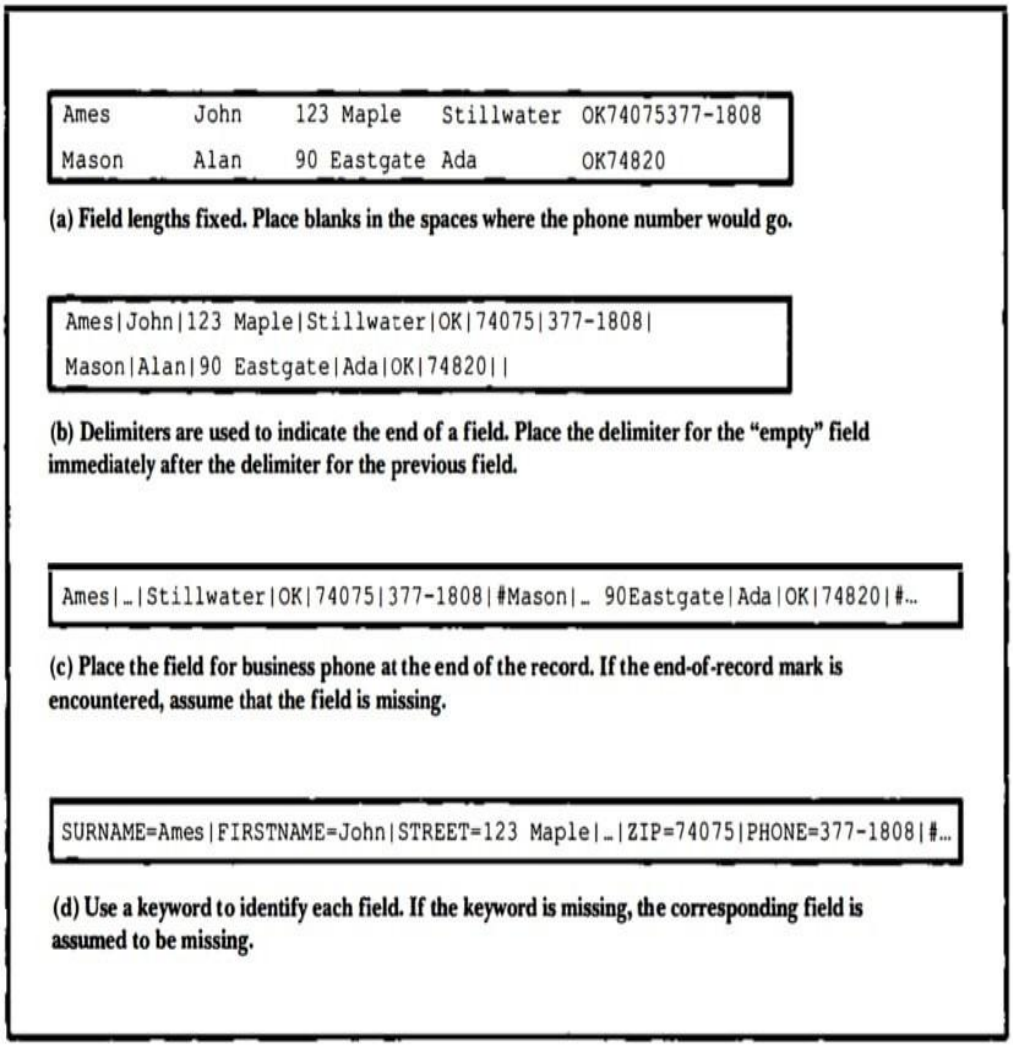

Figure 1.1 Four methods for field structures

Even if we don't know ahead of time which fields the file is supposed to contain. It is also a good format for dealing with missing fields. If a field is missing, this format makes it obvious, because the keyword is simply not there. It is helpful to use this in combination with delimiters, to show division between each value and the keyword for the following field. But this also wastes a lot of space: 50% or more of the file's space could be taken up by the keywords.

A record can be defined as a set of fields that belong together when the file is viewed in terms of a higher level of organization.

Record Structures

The five most often used methods for organizing records of a file are

- Require the records to be predictable number of bytes in length.
- Require the records to be predictable number of fields in length.
- Begin each record with a length indicator consisting of a count of the number of bytes that the record contains.
- 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.

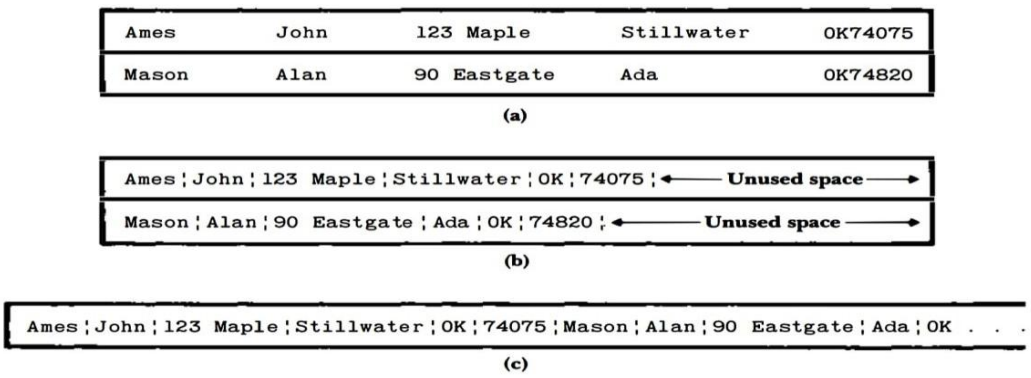

Figure 1.2 Making Records Predictable number of Bytes and Fields

Method 3: Begin Each Record with a Length Indicator

We can communicate 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

We can 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'.

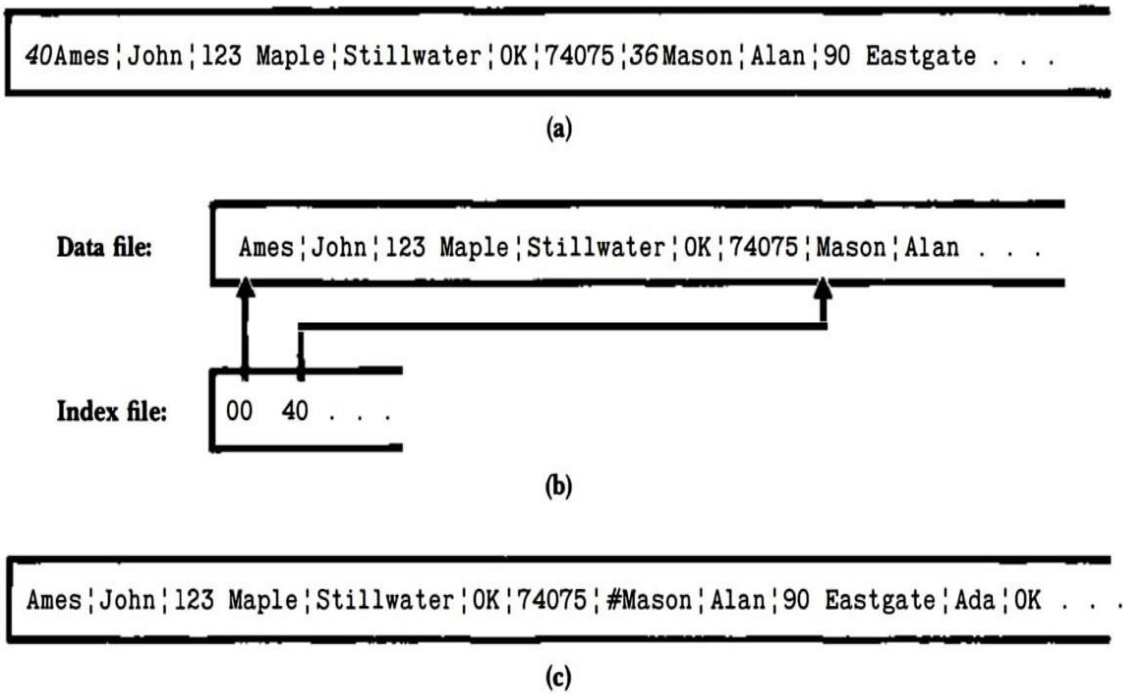

Figure 1.3 Using Length Indicator, Index and Record Delimiters

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.

Chapter 2

SYSTEM ANALYSIS

Analysis of the Project

Voter ID Management System is used in any Election Commission of India to store voter details. The system is initially used to add the details with all its specifications entered correctly into a file corresponding to its Voter ID, which is different for each and every voter. The system can be used to search, delete, modify or display existing records of all the voters.

Structure used to Store the Fields and Records

a) Storing Fields

Fixing the Length of Fields:

In the Voter ID Management System, the Voter ID field is a character array that can hold an integer value of fixed size. Here other fields are character arrays, here the Voter ID is the primary key. By using primary key the entire records can be easily obtained. We preserve the identity of fields by separating them with delimiters. We have chosen the vertical bar character (|), as the delimiter here.

b) Storing Records

Making Records a Predictable Number of Fields:

In this system, we have a variable number of fields, each with a maximum length, that combine to make a data record. Varying 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.

Using an Index to Keep Track of Addresses:

We use secondary indexes to keep byte offsets for each record in the original file. Secondary indexes can built on any field of data file or on combination of fields. Secondary indexes will typically have multiple locations for a single key. Our choice of a record delimiter for the data files is the end-of-line (new-line) character (\n).

Operations Performed on a File

a) INSERTION: The system is initially used to add voter records containing the voter attributes like Voter ID, name of the voter, address, contact number and sex. Records with duplicate Voter ID fields are not allowed to be inserted. If we try insert a duplicate Voter ID it displays a message Voter ID is already present.

b) DISPLAY: The system can then be used to display existing records of all voters. The records are displayed based on which we have entered the voter details which is not in sorted order.

c) SEARCH: The system can then be used to search for existing records of all the voters. The user is prompted for a Voter ID, which is used as the key in searching for records in the simple index. If we select secondary indexing it asks for the voter name which is the secondary key in searching for the records in the secondary index files. The secondary key is searched to obtain the desired primary key of the voter, which is then used to seek to the desired data record in any of the voter 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.

d) DELETE: The system can then be used to delete existing records from all voter details. The reference to a deleted record is removed from index file and $ is placed in data file for a particular record which is deleted. The requested record, if found, is marked for deletion, a "record deleted" message is displayed. If absent, a "record not found" message is displayed to the user.

e) MODIFY: If selected for modify option, the System will ask to enter the particular Voter ID which is to be modified and thereby after entering the Voter ID, the corresponding details of the voter are also displayed and the option to modify any of the field excluding the primary key which is Voter ID here is asked to enter. The user can give all related information and then press enter. And the updated or modified values will reflect back into the file.

Indexing Used

B+TREE

A B+TREE of simple indexes on the primary key is used to provide direct access to data records. Each node in the B+TREE consists of a primary key with the reference to record.

The primary key field is the V_ID field while the reference field is the starting byte offset of the matching record in the data file. Each B+TREE node can have max of 2 child node.

The V_ID is stored in the B +TREE, and hence written to an index file. On retrieval the matched V_ID is used, before it is used to seek to a data record, as in the case of requests for a search, delete, modify operation. As records are added, nodes of the B+ TREE undergo splitting (on detection of overflow), merging (on detection of underflow) or redistribution (to improve storage utilization), in order to maintain a balanced tree structure. The data files are entry-sequenced, that is, records occur in the order they are entered into the file. The contents of the index files are loaded into their respective B - TREEs, prior to use of the system, each time. Each B+TREE is updated as requests for the available operations are performed. Finally, the B +TREEs are written back to their index files, after every insert, delete and modify operation.

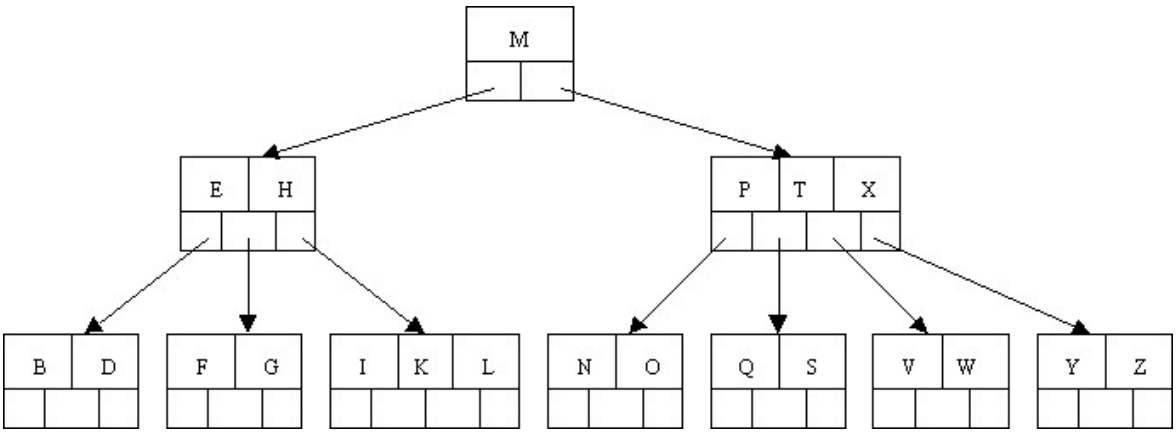

Figure 2.1 A B+Tree Implementation

Chapter 3

SYSTEM DESIGN

Design of the Fields and Records

The V_ID is declared as a character array that can hold a maximum of 10 characters. All the fields name, address, contact number, sex are declared as characters, each having a predetermined range, only within which it is accepted during input. Hence, a typical data file record can have up to 77 bytes of information to be stored, hence occupying a maximum of 90 bytes, in the data file. This includes the 4 bytes taken up by the field delimiters (|) and the 1 byte taken up by the record delimiter (\n). The class declaration of a typical product file record and member functions is as shown in Figure 3.1:

```
class voter
{
        public:
                char V_ID[10];
                char name[30];
                char addr[40];
                char dob[8];
                char gender[6];
                void Clear();
                int Unpack(fstram&);
                int Pack(fstream&);
                void Input(int);
                void Display();
                void Append();
                void Remove(int);
                ~pan(){}
                void Assign(voter&);
 4:1
```

Figure 3.1 Design Of Fields

User Interface

The User Interface or UI refers to the interface between the system 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.

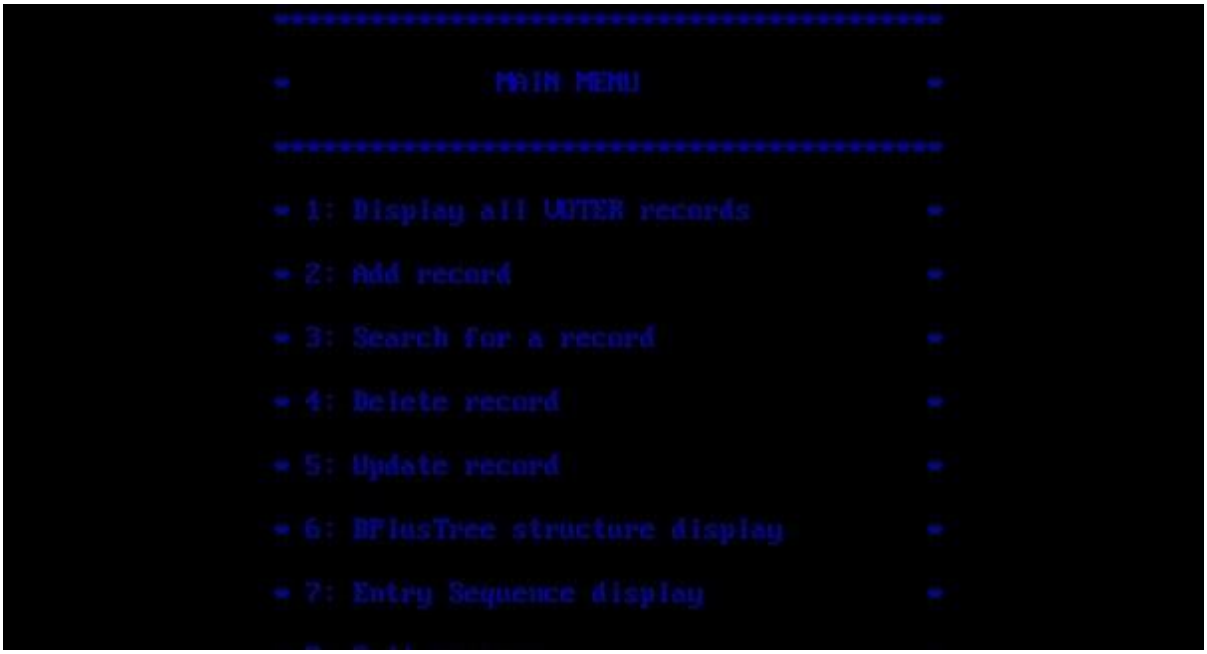

Figure 3.2 User Menu Screen

Insertion of a Record

If the operation desired is Addition of voter details, the user is required to enter 2 as his/her choice, from the menu displayed, after which a new screen is displayed. Next, the user is prompted to enter the Voter ID, name of the voter, address of the voter, date of birth of the voter, sex. The user is prompted for each input until the value entered meets the required criterion for each value. After all the values are accepted, a "record inserted" message is displayed and the user is prompted to press any key to return back to the menu screen. After insertion of details of the voter, every voter is given with the unique Voter ID which cannot be given to any other voters.

Display of a Record

If the operation desired is Display of Records, the user is required to enter 1 as his/her choice from the menu displayed after which new screen is displayed. If there are no records in the file, "no records found" message is displayed. For voter files with atleast one record, each Voter ID, followed by the details of each record within the file, with suitable headings, is displayed. In each case, the user is then prompted to press any key to return back to the menu screen.

Deletion of a Record

If the operation desired is Deletion, the user is required to enter 4 as his/her choice, from the menu displayed, after which a new screen is displayed. Next, the user is prompted to enter the voter name, whose file from which a record is to be deleted. If there are no records in the file, a "no records to delete" message is displayed, and the user is prompted to press any key to return back to the menu screen. If there is at least 1 record in the file, the user is prompted for the voter, whose matching record is to be deleted. The voter name entered is used as a key to search for a matching record. If none is found, a "record not found" message is displayed. If one is found, details of the particular record is displayed followed by confirmation of deletion. Then "record deleted" message is displayed. In each case, the user is then prompted to press any key to return back to the menu screen.

Search for a Record

If the operation desired is Search, the user is required to enter 3 as his/her choice, from the menu displayed, after which a new screen is displayed. Next, the user is prompted to enter the V_ID, whose file the record is to be searched for. The V_ID entered is used as a key to search for a matching record. If none is found, a **"**record not found**"** message is displayed. If one is found, the details of the record, with suitable headings, are displayed. In each case, the user is then prompted to press any key to return back to the menu screen.

Modifying or updating a Record

If the operation desired is Update, the user is required to enter 5 as his/her choice, from the menu displayed, after which a new screen is displayed. Next, the user is prompted to enter the patient id, whose file from which a record is to be modified. If there are no records in the file, a "no records to delete" message is displayed, and the user is prompted to press any key to return back to the menu screen. If there is at least 1 record in the file, the user is prompted for the V_ID, whose matching record is to be updated. Instead of deleting the old record and inserting new one, the system makes changes to the old product itself for the matching V_ID record.

The V_ID entered is used as a key to search for a matching record. If none is found, a "record not found" message is displayed and the user is then prompted to press any key to return back to the menu screen. If one is found, a "confirm permanent modification" message is displayed, after pressing Y, the user is prompted to enter the name of the voter, address of the voter, date of birth of the voter, sex to be inserted into the file. The user is prompted for each input until the value entered meets the required criterion for each value. After all the values are accepted, a "record updated" message is displayed and the user is prompted to press any key to return back to the menu screen.

Design of Index

The B-TREE is declared as a class, an object of which represents a node in the B - TREE. Each node contains a count of the number of entries in the node and a reference to each descendant. The maximum number of descendants is 4 while the minimum is 2. Each node has an array of objects, each an instance of the class type index as shown in Fig. 3.3. Each object of type **"**index**"** contains a page id field and an address field. The contents of the B -TREE are written to the index file on disk after each insert, delete and modify operation. To ensure efficient space utilization, and, to handle the conditions of underflow and overflow, nodes may be merged with their siblings, or split into 2 descendants, thereby creating a new root node for the new nodes created, or, entries within nodes maybe shifted to save space.

The above operations are used to maintain a balanced tree-structure after each insertion and deletion. The links of a node, beginning at the root, are traversed recursively, while displaying, and, writing to the index file, in order to maintain an ascending order among index entries. The same links are traversed when there is a request to search for a record, if present.

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 a system based service or component into the requirements of end users.

About C++

Classes and Objects

A record file is declared as a Voter ID Management System object with the V_ID, voter name, address as its data members. An object of this class type is used to store the values entered by the user, for the fields represented by the above data members, to be written to the data file. The B Tree is declared as the class b+tree, an object of which represents a node in the B Tree. Each node contains a count of the number of entries in the node and a reference to each descendant. The maximum number of descendants is 4 while the minimum is 2. Each node also has an array of objects, each an instance of the class type index. Each object of type index contains a VNO field and an address field, which stores the ASCII representation of the starting byte address at which the corresponding data record is stored in the data file. Class objects are created and allocated memory only at runtime.

Dynamic Memory Allocation and Pointers

Memory is allocated for nodes of the B Tree dynamically, using the method malloc(), which returns a pointer (or reference), to the allocated block. Pointers are also data members of objects of type b tree node, and, an object's pointers are used to point to the descendants of the node represented by it. File handles used to store and retrieve data from files, act as pointers. The free() function is used to release memory, that had once been allocated dynamically.

File Handling

Files form the core of this project and are used to provide for system storage for user entered information on disk. The open() and close() methods, as the names suggest, are defined in the C++ Stream header file fstream.h, to provide mechanisms to open and close files. The physical file handles used to refer to the logical filenames, are also used

to ensure files exist before use and that existing files aren't overwritten unintentionally. The 2 types of files used are data files and index files. open() and close() are invoked on the file handle of the file to be opened/closed. open() takes 2 parameters- the filename and the mode of access. close() takes no parameters.

Character Arrays and Character functions

Character arrays are used to store the PNO fields to be written to data files and stored in a B Tree index object. They are also used to store the ASCII representations of the integer and floating-point fields, that are written into the data file, and the starting byte offsets of data records, to be written to the index file. Character functions are defined in the header file ctype.h. Some of the functions used include:

- toupper() – used to convert lowercase characters to uppercase characters.
- itoa() – to store ASCII representations of integers in character arrays.
- isdigit() – to check if a character is a decimal digit (returns non-zero value) or not (returns zero value).
- atoi() – to convert an ASCII value into an integer.
- atof() – converts an ASCII value into a floating-point number.

Pseudocode

Insertion Module

The insertion operation is implemented by append() function which adds index objects to the B-Tree if the V_ID entered is not a duplicate. It makes calls to other recursive and non-recursive functions.

```
void append()
{
  student std;
  int flag=1, pos;
  fstream file("student.txt",ios::app);
  std.Input(0);
  file.seekp(0,ios::end);
```

```
  pos=file.tellp();
  flag=s.Insert(std.URN,pos);
  if(flag && std.Pack(file)) cout << "\n\t Done...\n";
  else cout << "\n\t Failure.";
  file.close();
}

int SSET :: Insert(char *val,int disp) // function to insert key in block
{
int i=0;
BK x=FindNode(val);
for(i=0;i<x->cnt;i++)
if(strcmpi(x->keys[i],val)==0)
        {
        cout<< " \nkey "<<val<<" is repeated \n"<<endl;
        return 0;
        }
if(x->cnt < 4)
{
for(i=0;i<x->cnt;i++)
if(strcmpi(x->keys[i],val)>0) break;
if(strcmpi(x->keys[i],val)==0){
cout<< "\n key "<<val<<" is repeated \n"<<endl;
return 0;
strcpy(x->keys[d],val);
x->disp[d]=disp;
x->cnt++;
if(x->cnt < 4)
{
for(i=0;i<x->cnt;i++)
if(strcmpi(x->keys[i],val)>=0) break;
if(strcmpi(x->keys[i],val)==0)
//cout<< " key "<<val<<" is repeated "<<endl;
cout<<"";
```

```
else {
int d=i;
i=x->cnt-1;
while(i>=d)
{
strcpy(x->keys[i+1],x->keys[i]);
x->disp[i+1]=x->disp[i];
i--;
}
strcpy(x->keys[d],val);
x->disp[d]=disp;
x->cnt++;
} }
bt.create();
}
return 1;
}
```

Display Module

The display() function displays the all the records in the order we have entered by accessing each index object stored at each node. The byte offsets in the objects are used to retrieve and display the corresponding data record fields.

```
void display(node * p)
{
  int i,j;
  student std;
  if(p->isLeaf())
  {
          fstream file("student.txt",ios::in);
         if(file.fail())
{
         gotoxy(24,10);
         cout<<"!!! ...The File Is Empty... !!!";
```

```
            getch();
            return;
}

  cout<<"RECORDS : "<<p->cnt;
  cout<<"\n************************************************************";
  for(i=0;i<p->cnt;i++)
    {
            block * t=p->ptr[i];
            for(j=0;j<t->cnt;j++)
            {
             file.seekg(t->disp[j],ios::beg);
             if(std.Unpack(file))
             {
            std.Display();
            cout<< "\n\t\t\t\t Press any key ...\n";
  cout<<"\n************************************************************";
            getch();
             }
             else break;
            }
    }
    file.clear();
    file.close();
  }
  for(i=0;i<p->cnt;i++)
    if(p->dptrs[i]!=0) display(p->dptrs[i]);
}
void SSET :: display()          // frunction to display nodes
     {
                    int j=0;
       BK t;
       t=head;
       getch();
```

```
    cout<<"************************************************";
    cout<<"\n Block Structure \n";
    cout<<"************************************************";
        while(t != 0)
    {
        cout<<"\n Node :"<<j;
        for(int i=0;i<t->cnt;i++)
        {
         cout<<"\n keys["<<i<<"] : " <<t->keys[i];
         // <<"\t disp["<<i<<"] : "<<t->disp[i]
//         cout<<"\n";
        }
        t=t->link;
        j++;
    }
   }
```

Deletion Module

The remove() function deletes values from the index file. It will ask for the voter name, if that product is present it will display details of that voter and asks to enter the Voter ID and delete that voter. If that voter is not present then record not found message will be displayed.

```
void remove(char *key)
{
  int r=0,found=0,s;
  char del='N';
  student  stds[100],std;
  fstream file("student.txt",ios::in);
  file.seekg(0,ios::beg);

  while(!file.fail())
    if(std.Unpack(file))
          if(strcmpi(std.URN,key)==0)
```

```
        {
        found=1;
        cout<<" \n Record :";
        std.Display();
        cout<<"\n\n Confirm permanent deletion:[Y/N]";
        cin>>del;
        if(!(del=='Y' || del=='y'))
        {
          stds[r].Clear();
          stds[r++].Assign(std);
        }
        else
        cout<<"\n\n\t Deleted : ";
         }
         else
         {
        stds[r].Clear();
        stds[r++].Assign(std);
         }
  file.clear();
  if(found==0) cout<<"\n\n\t Record not found.";
  else {
    file.close();
    file.open("student.txt",ios::out);
    file.seekp(0,ios::beg);
    for(s=0;s<r;s++)
          if(!(stds[s].Pack(file))) continue;
  }
  file.close();
}
```

Search Module Pseudocode

The search() function searches the secondary file, based on the secondary key which we have entered, if that voter is present it will display details of that voter and ask to enter

the Voter ID and displays the details. If that voter is not present then record not found message will be displayed.

```
void search(char *key)
{
  student std;
  int found=0,i;
  block *dp;
  fstream file("student.txt",ios::in);
  file.seekg(ios::beg);
  dp=bt.search(key,found);
  if(found==0)
          cout<<"\n\n\t Record not found...!\n";
  else
  {
           found=0;
           for(i=0;i<dp->cnt;i++)
            if(strcmpi(key,dp->keys[i])==0)
            {
                    found = 1;
                    file.seekg(dp->disp[i],ios::beg);
           std.Unpack(file);
           cout<<"\n\n\t Record found : ";
           std.Display();
          }
   if(found==0) cout<<"\n\n\t Record not found ";
  }
  file.clear();
  file.close();
}
```

Update Record

A record can be updated using the primary key that is, the primary key. The primary key is entered and all the fields of the record are modified.

```
void update(char *key)
{
 student stds[100],std;
 int f=0,found=0,g;
 char upd='n';
 fstream file("student.txt",ios::in);
 file.seekg(0,ios::beg);
 while(!file.fail())
  if(std.Unpack(file))
          if(strcmpi(std.URN,key)==0) {
    found=1;
           cout<<"\n\tRecord:";
    std.Display();
    cout<<"\n\n Confirm permanent updation:[Y/N] ";
    cin>>upd;
    if(!(upd=='Y' || upd=='y')) {
          stds[f++].Assign(std);
    }
    else {
          cout << "\n\t Enter the new record :\n";
          stds[f].Clear();
          stds[f++].Assign(std);
    }
          }
   else {
           stds[f].Clear();
    stds[f++].Assign(std);
   }
   file.clear();
   if(found==0)
    cout<<"\n\n\t Record not found...!\n";
   else
   {
    file.close();
```

```
file.open("student.txt",ios::out);
file.seekp(0,ios::beg);
for(g=0;g<f;g++)
      if(!(stds[g].Pack(file))) continue;
file.clear();
}
file.close();
}
```

Testing

Software Testing is the process used to help identify the correctness, completeness, security and quality of the developed computer software. Testing is the process of technical investigation and includes the process of executing a program or application with the intent of finding errors.

Unit Testing

It is a level of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc.

In object-oriented programming, the smallest unit is the method, which may belong to base or super class, abstract class or derived or child class. Unit testing frameworks, drivers, stubs, and mock/fake objects are used to assist in unit testing. It is performed by using white box testing method.

Unit testing is the first level of software testing and is performed prior to integration testing. It is normally performed by software developers themselves or their peers. In rare cases, it may be performed by independent software testers.

Table 4.1 Unit test cases for input

Case_id	Description	Input data	Expected o/p	Actual o/p	Status
1	Opening a file to insert the data	Insert option is	File should be opened in append mode	File opened in append	Pass

		selected	without any error messages	mode	
2	Input V_ID	ABC13243 45	Accept the V_ID and prompt for Name will be displayed	"Enter the Name:"	Pass
3	Input V_ID	ABC13243 5	Invalid V_ID and displays the message "V_ID already exist"	"V_ID already exist"	Pass
4	Enter the V_ID: Enter the name: Enterthe address: Enter the dob: Enter the sex:	ABC12367 RAJ Banglore 06/04/1999 Male	It should accept all the fields and it will return to the main menu	"Press any key to return to main menu"	Pass
5	File Updation	-	It should pack all the fields and will be appended to the data file	Data will be updated both in index and data file	Pass
6	Closing a file	-	File should be closed without any error	File is closed	Pass

Integration Testing

Integration testing is also taken as integration and testing this is the major testing process where the units are combined and tested. Its main objective is to verify whether the major

parts of the program is working fine or not. This testing can be done by choosing the options in the program and by giving suitable inputs it is tested.

Table 4.2 Integration testing for all modules

Case_Id	Description	Input	Expected o/p	Actual o/p	Status
1	To display the entered records of the data file	Enter the option 1 in the menu	Should display the record one after the other	Display the record one after the other	Pass
2	To add the new records into the data file	Enter the option 2 in menu	Should display the record entry form	Display the record entry form	Pass
3	To search for particular record in the file	Enter the option 3 in menu and should enter the V_ID:ABC1432437	Should display record not found	Record not found	Pass
4	To search for a particular record in the file	Enter the option 3 in the menu and should enter the V_ID:ABC1234567	Should display the record is found and contents of searched record	The record is found. Contents will be displayed	Pass
5	To delete a particular record in the file	Enter the option 4 in menu and should enter V_ID:2	Should display record not found	Record not found	Pass
6	To delete a particular record in the file	Enter the option 4 in menu and should enter V_ID:ABC1234567	Deletes the record	The record is deleted	Pass

7	To update a particular record in file	Enter the option 5 in menu and should enter V_ID:ABCXYZ1234	Should display record not found	Record not found	Pass
8	To update a particular record in the file	Enter the option 5 in menu and should enter V_ID:ABC1234567	Record is found and allows the user to re-enter the details in the record for updation	Record is found and allows the user to re-enter the details in the record for updation	Pass
9	To display all the entered records in the data file	Enter the option 3 in menu	Displays all the saved records	Display all the saved records	Pass
10	To quit the program	Enter the option 5 in the menu	Exit the program	Exit the program	Pass

System Testing

System testing is defined as testing of a complete and fully integrated software product. This testing falls in black –box testing where in knowledge of the inner design of the code is not a pre-requisite and is done by the testing team. System testing is done after integration testing is complete. System testing should test functional and non-functional requirements of the software. It is the third level of software testing performed after integration tesing and before acceptance testing. It is the process of testing an integrated systems to verify that it meets specified requirements. Usually, black box testing method is used.

Table 4.3 System Testing for voter record details

Case id	Description	Input data	Expected o/p	Actual o/p	Status
1	To display	Display records for	Record is	Record is	Pass

Case id	Description	Input data	Expected o/p	Actual o/p	Status
	the records	valid V_ID='ABC1234567'(present)and invalid V_ID ='ABC1223345'(not present)	displayed for valid and record not found for invalid V_ID	displayed for valid V_ID and record not found for invalid V_ID	
2	To search the records	Display records for valid V_ID='ABC1234567'(present)and invalid V_ID ='XYZ2225557'(not present)	Record is displayed for valid and record not found for invalid	Record is displayed for valid and record not found for invalid	Pass
3	To delete the records	Delete records for valid V_ID = 'ABC1234567'(present) and invalid V_ID='CSK1122334' (not present)	Record is deleted for valid and record not found for invalid	Record is deleted for valid and record not found for invalid	Pass
4	To update the records	Update records for valid V_ID = 'ABC1234567'(present) and invalid V_ID='PSK4455667' (not present)	Record is Updated for valid and record not found for invalid	Record is Updated for valid and record not found for invalid	Pass

Acceptance Testing

Acceptance Testing is a level of software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery. It is performed by:

- Internal Acceptance Testing (Also known as Alpha Testing) is performed by members of the organization that developed the software but who are not directly involved in the project (Development or Testing). Usually, it is the members of Product Management, Sales and/or Customer Support.

- External Acceptance Testing is performed by people who are not employees of the organization that developed the software.

Table 4.4 Acceptance testing for voter record details

Case ID	Description	Input Data	Expected o/p	Actual o/p	Status
1.	Insertion Module	Inserting valid data	Record added successfully	Record added successfully	Pass
2.	Deletion Module	Enter valid VOTER id.	Record deleted successfully	Record deleted successfully	Pass
3.	Modification Module	Enter valid VOTER id.	Record updated successfully	Record updated successfully	Pass

Discussion of Results

All the menu options provided in the application and its operations have been presented in as screen shots from Fig 4.1 to 4.5

Menu Options: This is the home screen through which user interacts with the system. It gives the user various options to access the records using operations such as insertion, deletion, modification, searching, and display of voter's details. The user can implement insertion, deletion and traversal in the file using b-tree to index the user records. To exit the system at any time, user can press option 7 to return back.

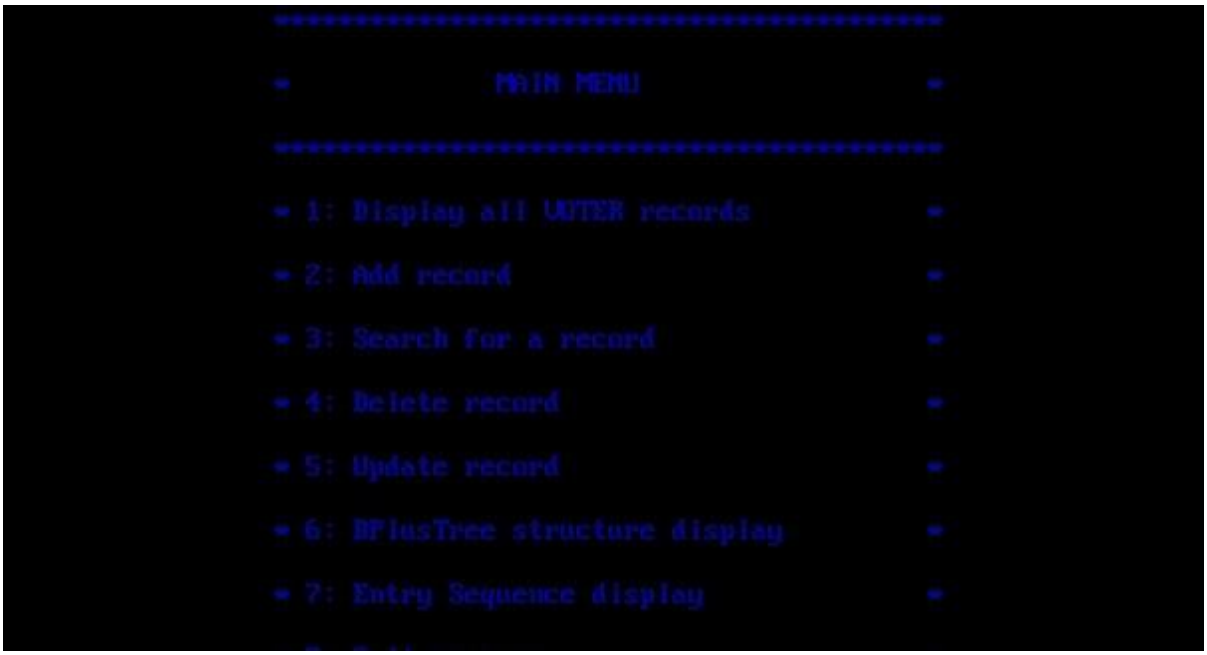

Figure 4.1 Main menu

Insertion: Inserting a new voter record. Each voter record should have unique Voter ID as shown in figure 4.2.

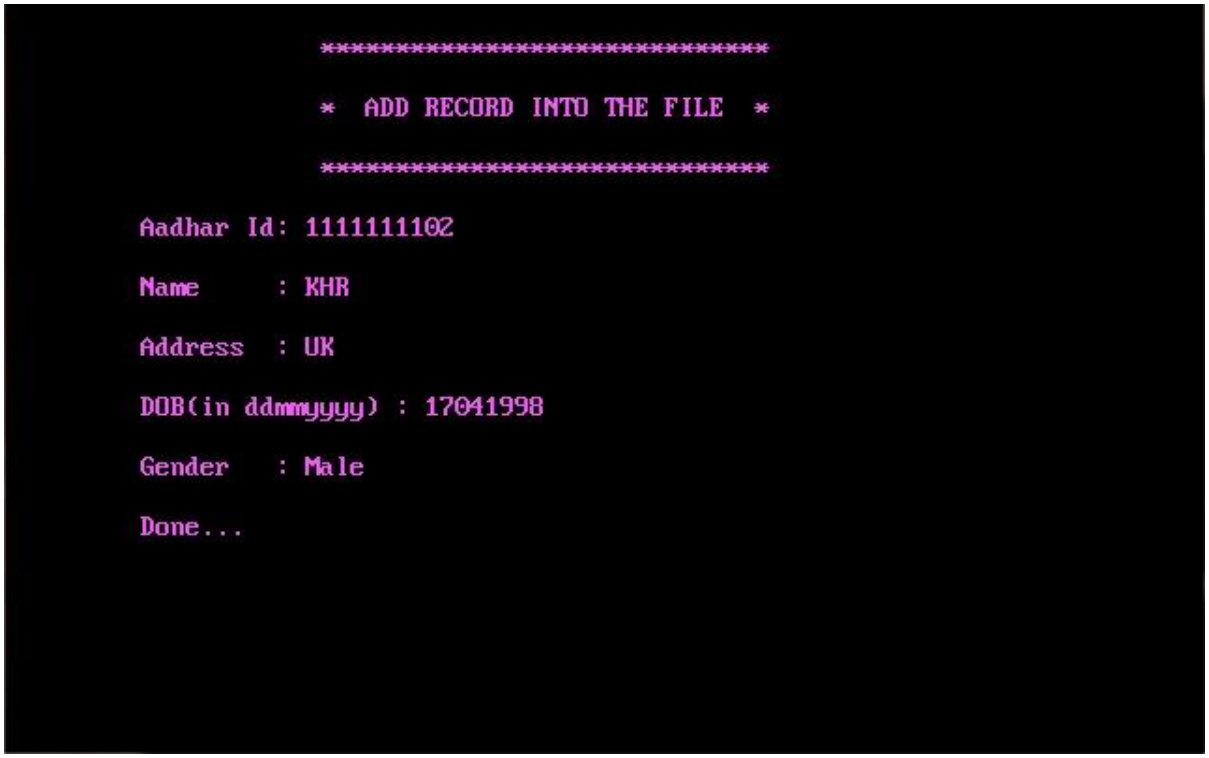

Figure 4.2 Insertion of record

Deletion of a product: Deleting the voter record by giving voter name as input as shown in figure 4.3.

Figure 4.3 Deletion of Record

Searching of a Product: Searching the voter record by giving voter's name as searching key as shown in figure 4.4.

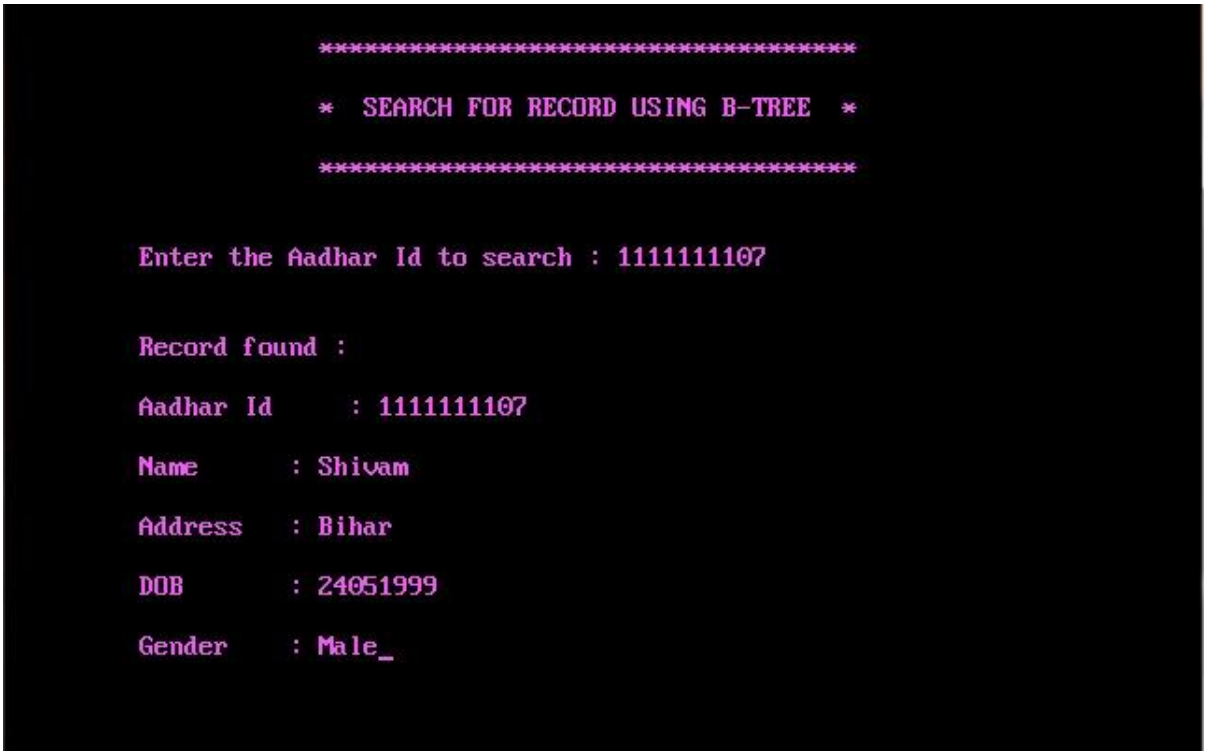

Figure 4.4 Searching Record

Updating of a Details: Updating the voters details using V_ID as a primary key as shown in figure 4.5.

Figure 4.5 Updating of a Record

Data file contents: Stored details of voter record in datafile. `$` indicates deleted and modified voter record as shown in figure 4.6.

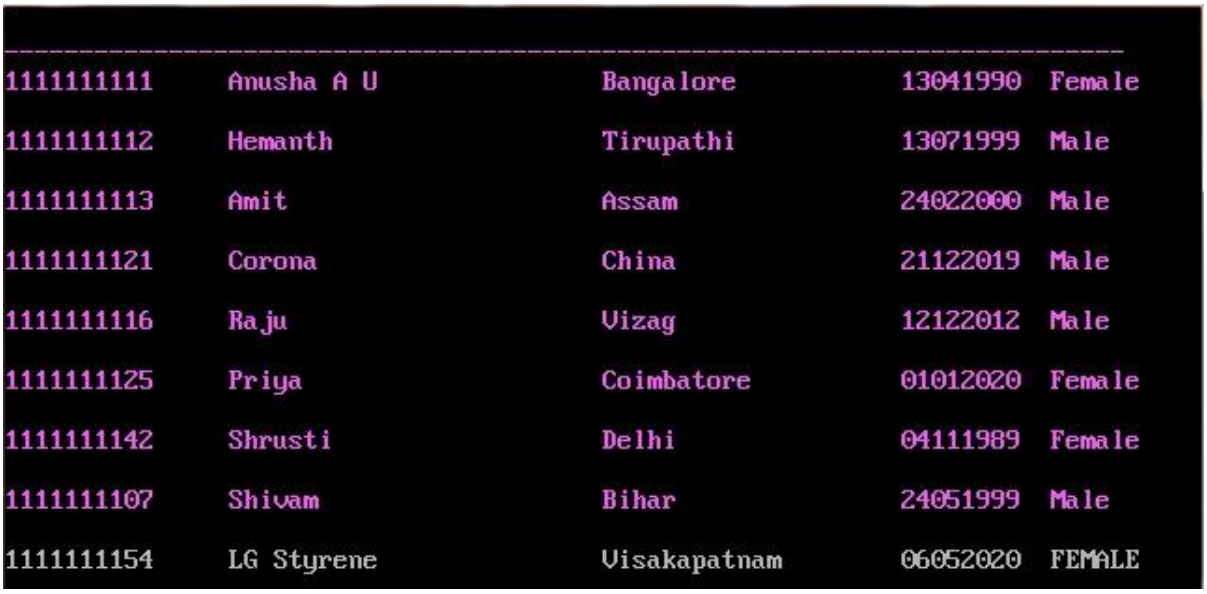

Figure 4.6 Data file details

Index File Contents: Index file contains secondary key and primary reference in it as shown in figure 4.7.

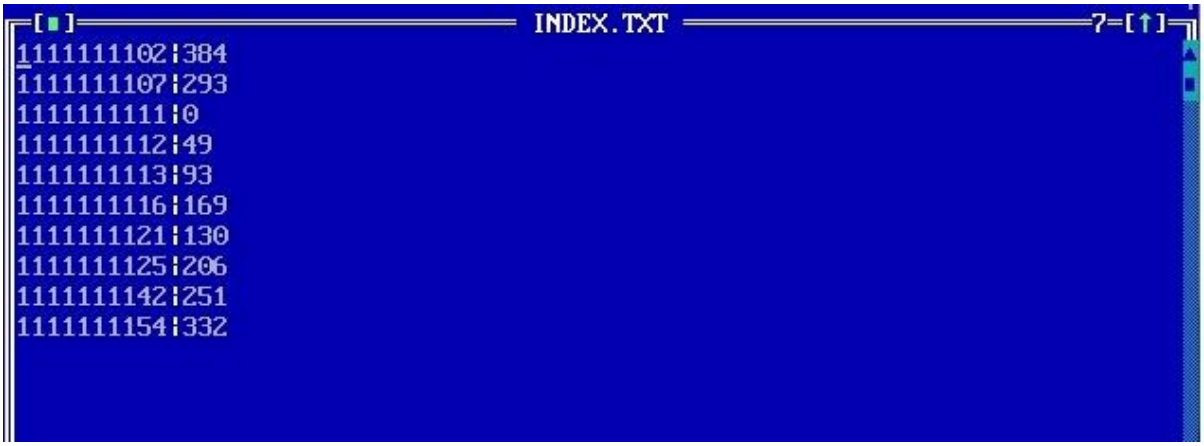

Figure 4.7 Index file details

Chapter 5

CONCLUSION AND FUTURE ENHANCEMENTS

This system helps to store the voter details in the Voter ID electronically. The stored details of the voters are secured and can be retrieved at any point of time. This system reduces the amount of manual data entry and gives greater efficiency. The User Interface of it is very friendly and can be easily used by anyone. It also decreases the amount of time taken to write voter details and other modules. Hence this system is performing all the tasks accurately and is doing the work for which it is made.

We can enhance this system by including more facilities like adding specific user details such as his other government ID proof details (PAN Card, Aadhar Card etc) so as to link all his credentials in one place. Providing such features enables the user to include more usefulness into the system. User Interface (GUI) can be improved using better graphics.

REFERENCES

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

[2] www.stackoverflow.com

[3] www.tutorialpoint.com

[4] www.wikepedia.com

[5] www.softwaretestingfundamentals.com