VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

A Mini Project Report on

"MUSIC RECORD MANAGEMENT SYSTEM"

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 Engineering*** *Visvesvaraya Technological University, Belagavi*

Submitted by

PHALGUNI S PRASAD	PRAGNA G PRASAD	PRAJNA S G
1RN20IS105	1RN20IS106	1RN20IS107

Guide
Ms.Kavitha B
Assistant Professor
Dept.of ISE,RNSIT

Faculty In-Charge
Ms.Vinutha G K
Assistant Professor
Dept. of ISE,RNSIT

ESTD : 2001
An Institute with a Difference

Department of Information Science and Engineering

RNS Institute of Technology

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

2022 – 2023

RNS Institute of Technology

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

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

ESTD : 2001

An Institute with a Difference

CERTIFICATE

This is to certify that the mini project report entitled **MUSIC RECORD MANAGEMENT SYSTEM** has been successfully completed by **PHALGUNI S PRASAD** bearing USN **1RN20IS105, PRAGNA G PRASAD** bearing USN **1RN20IS106**, and **PRAJNA S G** bearing USN **1RN20IS107** 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 the academic year 2022 – 2023. 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 structure Laboratory for the said degree.

Ms.Kavitha B	**Ms. Vinutha G K**	**Dr. Suresh L**
Project Guide	Faculty In-charge	Professor and HoD
Assistant Professor	Assistant Professor	Dept. of ISE, RNSIT
Dept. of ISE, RNSIT	Dept. of ISE, RNSIT	

External Viva

Name of the Examiners	**Signature with Date**
1. ____________	**1.** ____________
2. ____________	**2.** ____________

ABSTRACT

Music Record Management System is an essential tool for anyone looking to manage a large collection of music. These systems allow users to store, organize, and retrieve music data quickly and efficiently, making it easy to find and play their favorite songs. One approach to managing music data is the use of B Plus-trees, a data structure that allows for fast and effective data retrieval and insertion. In this report, we will explore the use of B Plus-trees in music record management systems, and examine the benefits and limitations of this approach. We will also discuss future enhancements that could be made to improve the functionality and usability of these systems. Overall, our findings suggest that the use of B Plus-trees in music record management systems offers a number of benefits, including improved system performance, faster data retrieval times, and increased accuracy and efficiency. As such, they are an ideal choice for anyone looking to manage a large music collection, whether for personal or professional purposes.

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 thank our beloved Director **Dr. M K Venkatesha** for his confidence feeling words and support for providing facilities throughout the course.

We would like to express our thanks to our Principal **Dr. Ramesh Babu H S** 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, master mind behind my Project work for guiding me and helping me out with the report.

We would like to express our profound and cordial gratitude to our Project Guide **Ms. Kavitha B,** 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 express our sincere gratitude to **Mrs. Vinutha G K,** Assistant Professor, Department of Information Science and Engineering, for her constant support and guidance.

We would like to thank all other teaching and non-teaching staff of Information Science & Engineering who have directly or indirectly helped me to carry out the project work. And lastly, we would hereby acknowledge and thank our parents who have been a source of inspiration and also instrumental in carrying out this Project work.

PHALGUNI S PRASAD
1RN20IS105
PRAGNA G PRASAD
1RN20IS106
PRAJNA S G
1RN20IS107

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER-1

INTRODUCTION

1.1 Introduction to File Structures:

File Structures is the Organization of Data in Secondary Storage Device in
Such a way that minimizes the access time and the storage space.

1.1.1 File Structure:

It is a combination of,

- Representation for data in files and
- The operations for accessing the data

It allows applications to read write and modify data search .An improvement in file Structure design make a application hundred times faster .To access the data faster From the storage disk we use file structure .Disks are slow which are used to pack Thousands of megabytes of data.

1.1.2 Records and its types:

Are collection of fields, possibly of different Data types, typically in fixed number of sequence. The fields of are also be called members. For example, a date could be sorted as a record containing a numeric year field, a Month field represented as a string, and a numeric day of month field. A record are distinguished from arrays by the fact that the number of fields is typically fixed, each field has a name, and that each field may have a different types.

A record type is a data type that describes such values and variables. The Definition includes specifying the data type of each field and an identifiers by which it can be accessed.

1.1.3 Why we need File Structure??

As we know without proper structure of organizing is ordinary system, it will generate some types of problems. So avoiding this kind of problems we go for File Structure.

The File structure means it's tell about how the system will stored and access the record from memory and also its tell how the disk are performed its tasks and also how to speed up the execution of transferring data.

- **Data processing from a computer science prospective:**

-storage of data

-organization of data

-access to data

This will be built on your on your knowledge of Data Structures. Some of the important concepts used

Field Structures:

(1) Fixed Length Fields: The method to organize fields is by limiting the maximum size of each field. The advantage in this method is that since the size of each field is fixed, the entire field can be read at once.

(2) Length Indicator Fields: The length of each field is specified as a prefix to actual data.

(3) Delimited Fields: Any special character which is not a part of actual data can be used as separator.

Record Structures:

(1) Fixed Length Record Structure: Each record is stored in fixed size. The size can be determined by adding the maximum space occupied by each field and some space reserved for the header data.

(2) Fixed Field Count: The number of the fields in each record is fixed.

(3) Index Structure for Records: An index is a collection of key field and reference field.

(4) Key Field: Key Field is a member of record which can uniquely identify the record.

(5) Reference Field: Reference Field contains the value that points to the address of the corresponding record in the file.

1.2 Primary Indexing:

A primary index consists of all prime-key attributes of a table and a pointer to physical memory address of the record of data file. To retrieve a record on the basis of all primary key attributes, primary index is used for fast searching. Binary search is done on index table and then directly retrieve that record from physical memory. It may be sparse.

- **Advantages of Primary index**
  - (i) Search operation is very fast.
  - (ii) Index table record is usually smaller.
  - (iii) A Primary index is guaranteed not to duplicate.

- **Disadvantages of Primary index**
  - (i) There is only one primary index of a table. To search a record on less than all prime-key attributes, linear search is performed on index table.
  - (ii) To create a primary index of an existing table, record should be in some sequential order otherwise database is required to be adjusted.

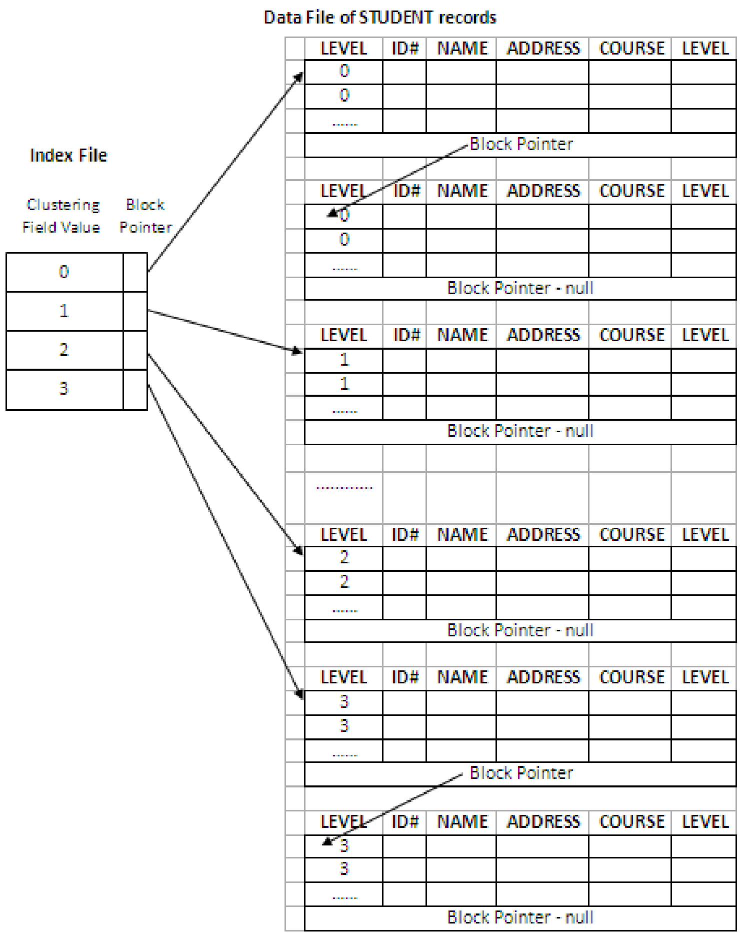


Fig – 1.1: An example of primary indexing.

CHAPTER - 2

INTRODUCTION TO PROJECT AND ALGORITHM

2.1 Indexing:

Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. It is a data structure technique which is used to quickly locate and access the data in a database.

Indexes are created using a few database columns.

The first column is the **Search key** that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly.

Note: The data may or may not be stored in sorted order.

The second column is the **Data Reference** or **Pointer** which contains a set of pointers holding the address of the disk block where that particular key value can be found.

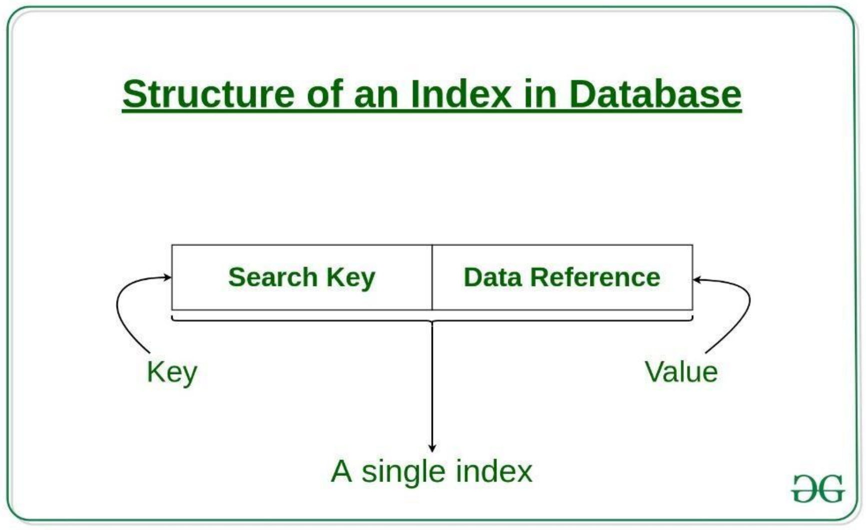


Fig-2.1: Structure of an Index in Database.

- **The indexing has various attributes:**

- **Access Types**: This refers to the type of access such as value based search, range access, etc.
- **Access Time**: It refers to the time needed to find particular data element or set of elements.

- **Insertion Time**: It refers to the time taken to find the appropriate space and insert a new data.
- **Deletion Time**: Time taken to find an item and delete it as well as update the index structure.
- **Space Overhead**: It refers to the additional space required by the index.

In general, there are two types of file organization mechanism which are followed by the indexing methods to store the data:

1. Sequential File Organization or Ordered Index File: In this, the indices are based on a sorted ordering of the values. These are generally fast and a more traditional type of storing mechanism. These Ordered or Sequential file organization might store the data in a dense or sparse format:

(i) Dense Index:

- For every search key value in the data file, there is an index record.
- This record contains the search key and also a reference to the first data record with that search key value.

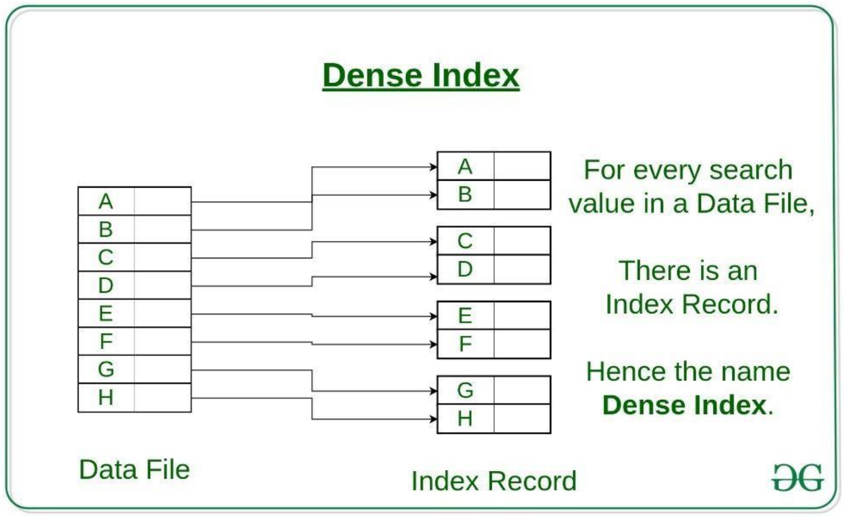

Dense Index
A
B
C
D
E
F
G
H
For every search value in a Data File,
There is an Index Record.
Hence the name Dense Index.
Data File
Index Record

Fig – 2.2: Dense Index

(ii) Sparse Index:

- The index record appears only for a few items in the data file. Each item points to a block as shown.

- To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
- We start at that record pointed to by the index record, and proceed along with the pointers in the file (that is, sequentially) until we find the desired record.

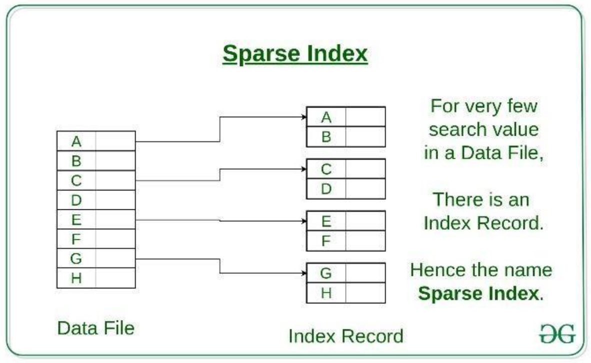

Sparse Index
A
B
C
D
E
F
G
H
For very few search value in a Data File,
There is an Index Record.
Hence the name Sparse Index.
Data File
Index Record

Fig – 2.3 : Sparse Index

2. Hash File organization:

Indices are based on the values being distributed uniformly across a range of buckets. The buckets to which a value is assigned is determined by a function called a hash function.

There are primarily three methods of indexing:

- Clustered Indexing
- Non-Clustered or Secondary Indexing
- Multilevel Indexing

1. Clustered Indexing:

when more than two records are stored in the same file these types of storing known as cluster indexing. By using the cluster indexing we can reduce the cost of searching reason being multiple records related to the same thing are stored at one place and it also gives the frequent joining of more than two tables (records).

Clustering index is defined on an ordered data file. The data file is ordered on a non-key field. In some cases, the index is created on non-primary key columns which may not be unique for each record. In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create index out of them. This method is known as the clustering index. Basically, records with similar characteristics are grouped together and indexes are created for these groups.

For example, students studying in each semester are grouped together. i.e. 1st Semester students, 2nd semester students, 3rd semester students etc. are grouped.

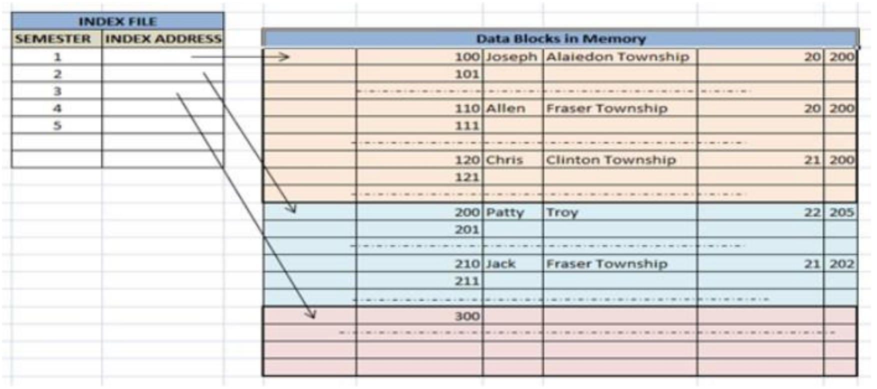

INDEX FILE
SEMESTER
INDEX ADDRESS
1
2
3
4
5
Data Blocks in Memory
100 Joseph Alaiedon Township 20 200
101
110 Allen Fraser Township 20 200
111
120 Chris Clinton Township 21 200
121
200 Patty Troy 22 205
201
210 Jack Fraser Township 21 202
211
300

Fig – 2.4: Example of Clustered Indexing.

- **Clustered index sorted according to first name (Search key)**
- ***Primary Indexing:***

  This is a type of Clustered Indexing wherein the data is sorted according to the search key and the primary key of the database table is used to create the index. It is a default format of indexing where it induces sequential file organization. As primary keys are unique and are stored in a sorted manner, the performance of the searching operation is quite efficient.

2. Non-clustered or Secondary Indexing:

A non-clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored. Data is not physically stored in the order of the index. Instead, data is present in leaf nodes. For e.g. the contents page of a book. Each entry gives us the page number or location of the information stored. The actual data here (information on each page of the book) is not organized but we have an ordered reference (contents page) to where the data points actually lie. We can have only dense ordering in the non-clustered index as sparse ordering is not possible because data is not physically organized accordingly.

It requires more time as compared to the clustered index because some amount of extra work is done in order to extract the data by further following the pointer. In the case of a clustered index, data is directly present in front of the index.

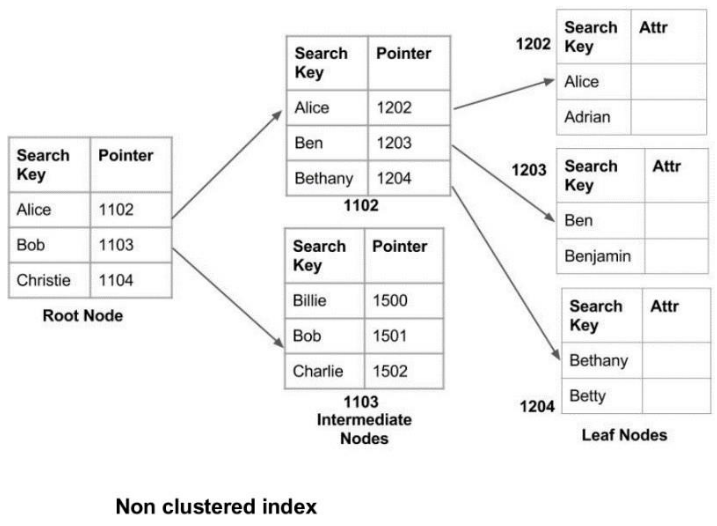


Fig – 2.5: Example for Non-clustered index

3. Multilevel Indexing:

With the growth of the size of the database, indices also grow. As the index is stored in the main memory, a single-level index might become too large a size to store with multiple disk accesses. The multilevel indexing segregates the main block into various smaller blocks so that the same can stored in a single block. The outer blocks are divided into inner blocks which in turn are pointed to the data blocks. This can be easily stored in the main memory with fewer overheads.

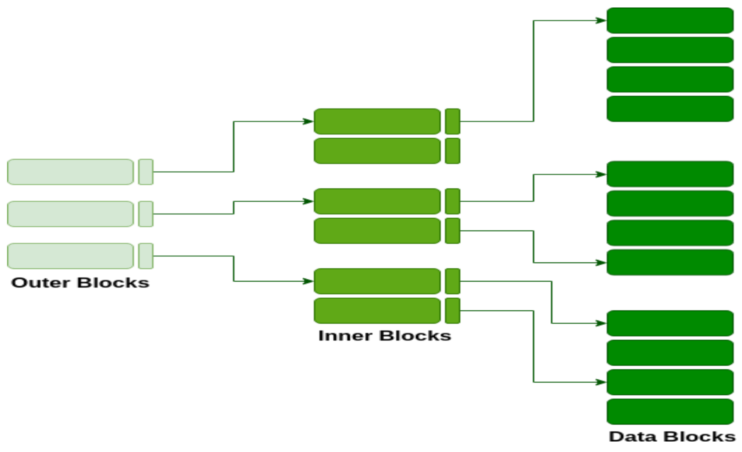


Fig – 2.6: Multilevel Indexing Diagram.

- **ALGORITHM:**

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

From the data structure point of view, following are some important categories of algorithms −

- **Search** − Algorithm to search an item in a data structure.
- **Sort** − Algorithm to sort items in a certain order.
- **Insert** − Algorithm to insert item in a data structure.
- **Update** − Algorithm to update an existing item in a data structure.
- **Delete** − Algorithm to delete an existing item from a data structure.

Characteristics of an Algorithm

Not all procedures can be called an algorithm. An algorithm should have the following characteristics −

- **Unambiguous** − Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- **Input** − An algorithm should have 0 or more well-defined inputs.
- **Output** − An algorithm should have 1 or more well-defined outputs, and should match the desired output.
- **Finiteness** − Algorithms must terminate after a finite number of steps.
- **Feasibility** − Should be feasible with the available resources.
- **Independent** − An algorithm should have step-by-step directions, which should be independent of any programming code.

How to Write an Algorithm?

There are no well-defined standards for writing algorithms. Rather, it is problem and resource dependent. Algorithms are never written to support a particular programming code.

As we know that all programming languages share basic code constructs like loops (do, for, while), flow-control (if-else), etc. These common constructs can be used to write an algorithm. We write algorithms in a step-by-step manner, but it is not always the case. Algorithm writing is a process and is executed after the problem domain is well-defined. That is, we should know the problem domain, for which we are designing a solution.

- **Create index function.**

```
1. Open the txt file in input mode using fstream.
2. While not the end of file.
1. Get the position.
2. Erase the buffer content.
3. Extract the characters from file and append it to buffer.
4. If the buffer first position is not equal to *
1. Check for buffer empty if so then break.
2. Call extracted function and assign to id.
3. Assign id to id list.
4. Assign pos to addlist.
3. End while.
4. Call sort function.
5. Erase the Buffer.
6. End of function.
```

- **Insert function.**

```
1. Read the details with id.
2. Append the details with id to buffer.
3. Open the txt file in input mode.
4. Append the buffer elements to file and close the file.
5. Assign id to id list.
6. Assign pos to addlist.
7. Call sort index function.
8. End of function.
```

- **Extract Id function.**

```
1.Get buffer as parameter.
2.Until the buffer has first delimiter symbol
    1.Assign buffer element to id.
3. Return the id.
```

- **SortIndex function.**

```
1.Declare variables.
2.For i in range of 0 to count.
    1.For j in range of i+1 to count.
        1.If id in Idlist at ith position is greater than id in Idlist at jth position, then sort the id in idlist
        2.End if.
    2. End for.
3. End for.
4. End.
```

- **Search function.**

```
1.Get the key as parameter.
2.Declare Variables.
3.Erase the buffer element.
4.Call search Index function with parameter key  to get the position
5.If position equals to -1 then print record not found.
6.Else if position is greater than or equal to 0.
    1.Open the txt file.
    2.Get the position from addlist.
    3.Seek the file
    4.Extract the characters from file and append it to buffer.
    5.Print the message and details. Close the file.
7. End.
```

- **Search Index function.**

1.Assign the low to zero and high to count.

2.While low is less than mid.

1.Mid element is equal to sum of low and height and divide the sum by 2.

2.If mid element of Idlist is equal to key. Then return mid.

3.Else if mid element of Idlist is less than key, Then low equal to mid+1.return mid

4.Else if mid element of Idlist is greater than key, Then high equals to mid-1. Return mid

5.Else return -1;

3.End.

- **Remove function.**

1.Declare variable.

2.Call search Index function with key as parameter and assign to position.

3.If position is greater than 0.then.

1.Open txt file.

2.Get the position from addlist.

3.Seek the file

4.Put * at the beginning of deleted file.

5.Move the next file id to deleted file location id in id list and same in addlist.

6.Decrement the count.

4.Else

1.Print the record not found.

5.Display updated file.

CHAPTER - 3

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.

3.1 Problem statement

This project is mainly concerned with developing a system where a user can add song details, search the details of the song using primary indexing, and remove the details. The music record management system is developed using Primary Indexing and B-Tree.

3.2 Objectives of the project

• The project Music Record Management system has been made to automate system.

• To allow only authorized user to access various functions and process available in the system.

• This system will help receptionist to add , remove and view the details on each record.

3.3 About the Programming Language C++

In the program "Time Complexity of Search and Traverse of a Key in primary indexing", we have used C++ as a medium to compile and run the program because of its versatile uses in the industry.

It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights. C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications, including desktop applications, servers (e.g. e-commerce, web search or SQL servers), and performance-critical applications. C++ is a compiled language, with implementations of it available on many platforms. Many vendors provide C++ compilers, including the Free Software Foundation, Microsoft, Intel, and IBM.

In the index of the application we provided 5 navigation choices

1. Display

2. Add

3. Search

4. Delete

5 .Update

6. Structure Display

7. Music File Display

8. Exit

##1 Display:
Here the user can view all the records details that were entered.

##2. Add:
Here the user can add a new record by entering the values for song id, name of the song, Date of release, and name of the artist.

##3 Search:
Here the user can search for the desired record using the primary key, that is, music Id.

##4. Delete:
Here the user can delete a record by entering the unique Id. The user will be prompted to confirm the action.

##5. Update:
Here the user can update the fields of any record. First, the existence of the record is searched and then the user is prompted to enter the values for each field.

##6. Structure Display:
In this option, the program displays the position of each node in the B-tree.

##7.Music File display:
Here all the records as stored in the output file is displayed on the screen.

##8. Exit:
By giving this choice user will exit from the application

3.4 Pseudo code:

Class:

```
class node// class for btree node
 {
  public:
 char keys[4][11];
 node *dptrs[4];
 node  *uptr;
 block *ptr[4];
int cnt;
 node();
 ~node(){}
int isLeaf();
 void split(node *,node *,node *);
 };
/**************************************************/
class btree     // class for btree
 {
  public:
 btree();
 int insert(char*,block *);
 node* findLeaf(char*,int&);
 block *search(char*,int&);
 void create();
 void dis(node*);
 ~btree(){}
 node *root;
 };
```

Insert:

```
int btree::insert(char* val, block *d)
{
  int i,flg;
  node *x=findLeaf(val,flg),*a1,*a2;
  if(flg==1)
  {
    //cout << "\n\t Repeated! " << val << endl;
    return 0;
  }
  if((x->cnt)==4)//there is no space in the record so insert
  {
    x->split(x,a1,a2);//split the leaf and upper nodes if needed
    x=findLeaf(val,flg);
  }
  for(i=2;i>=0;i--)
    if(strcmp((x->keys[i]),'\0')!=0)
      if(strcmp(val,(x->keys[i]))<0)
      {
strcpy(x->keys[i+1],x->keys[i]);
x->ptr[i+1]=x->ptr[i];
//forget about dptrs!
      }
      else
      {
strcpy(x->keys[i+1],val);
x->ptr[i+1]=d;
x->cnt++;
break;
      }
  if(i<0)
  {
    strcpy(x->keys[0],val);
    x->ptr[0]=d;
    x->cnt++;
  }
  else if(i==(x->cnt)-2)
    while((x->uptr)!=0)
    {
      x=x->uptr;
      strcpy(x->keys[(x->cnt)-1],val);
 x->ptr[(x->cnt)-1]=d;
    }
    return 1;
}
```

Search:

```
block *btree::search(char *key,int &fnd)
{
  int i,flg=0;
  fnd=0;
  node *x=findLeaf(key,flg),*a1,*a2;
  for(i=0;i<x->cnt;i++)
  if(strcmp(key,x->keys[i])<=0)
   {
    fnd=1;
    return(x->ptr[i]);
   }
  return head;
```

Split:

```
void block :: split()// function to split the block
      {
       block *newnode;
       newnode=new block;
       int j=0;
       for(int i=2;i<this->cnt;i++)
       {
strcpy(newnode->keys[j],this->keys[i]);
strcpy(this->keys[i],'\0');
newnode->disp[j]=this->disp[i];
this->disp[i]=-1;
newnode->cnt++;
j++;
       }
       newnode->link = this->link;
       this->link=newnode;
       this->cnt-=2;
       bt.create();
}
```

Merge:

```
void block :: merge()// function to merge two blocks
      {
       block *t1,*t2;
       t1=head;
       if(t1 == this)        // merge first block
       {
if(t1->link->cnt < 4 )  // check wether next block can be merged
{
 for(int p=t1->link->cnt;p>=0;p--)
 {
  strcpy(t1->link->keys[p],t1->link->keys[p-1]);
  t1->link->disp[p] = t1->link->disp[p-1];
 }
 strcpy(t1->link->keys[0],t1->keys[0]);
 t1->link->disp[0]=t1->disp[0];
 t1->link->cnt++;
 head=t1->link;
 t1->link=0;
 delete t1;
}
else if(t1->link->cnt==4)     // redistribution of keys
{
 strcpy(t1->keys[1],t1->link->keys[0]);
 t1->disp[1]=t1->link->disp[0];
 t1->cnt++;
 for(int c=0;c<t1->link->cnt;c++)
 {
  strcpy(t1->link->keys[c],t1->link->keys[c+1]);
  t1->link->disp[c]=t1->link->disp[c+1];
 }
 t1->link->cnt--;
}
  }
      else// find which block to be merged
      {
while(t1->link != this)
 t1=t1->link;
if(t1->cnt < 4)// merge with left node
{
 strcpy(t1->keys[t1->cnt],this->keys[0]);
 t1->disp[t1->cnt] = this->disp[0];
 t1->link = this->link;
 t1->cnt++;
 this->link=0;
 delete this;
}
else
{
 if(this->link !=0)  // check wether node to be merged is last
 {                    // if not
  t2=this->link;
  if(t2->cnt < 4)      // merge with right node
  {
   for(int i=t2->cnt;i>=0;i--)
   {
    strcpy(t2->keys[i],t2->keys[i-1]);
    t2->disp[i]=t2->disp[i-1];
   }
strcpy(t2->keys[0],this->keys[0]);
   t2->disp[0]=this->disp[0];
   t2->cnt++;
   t1->link = this->link;
   this->link=0;
   delete this;
  }
  else
  {//Redistribution
   t2=this->link;
   strcpy(this->keys[this->cnt],t2->keys[0]);
   this->disp[this->cnt]=t2->disp[0];
   this->cnt++;
   for(int i=0;i<t2->cnt;i++)
   {
    strcpy(t2->keys[i],t2->keys[i+1]);
    t2->disp[i]=t2->disp[i+1];
   }
   t2->cnt--;
  }
 }
 else// if it is last block
 {
  strcpy(this->keys[1],this->keys[0]);
  this->disp[1]=this->disp[0];
  strcpy(this->keys[0],t1->keys[t1->cnt-1]);
  this->disp[0]=t1->disp[t1->cnt-1];
  this->cnt++;
  t1->cnt--;
 }
}
 }
}
```

3.5 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 4.1.1: Unit test cases for input

Case_id	Description	Input data	Expected o/p	Actual o/p	Status
1	On selecting the choice the particular roles is allowed to access	choice	Particular role access will be allowed	Access allowed	Pass
2	Input username and password	Admin	Admin section is displayed	Admin section is displayed	Pass
3	Input option to perform a add,delete,display the music record and index list	Choice 1	Perform add, delete, display album details	Perform add,delete,display record details	Pass
4	Enter the song ID: Enter the song name: Enter the date of publication: Enter the artist name :	203 Lalisa 5-12-2020 Lisa	It should accept all the fields and display music records added successfully	Music record added successfully	Pass
5	Display music album records	-	It should unpack and display music album record	Displays music record	Pass

6	Remove the music records	Enter the Id	Record will be deleted updated file will be displayed	Record will be deleted updated file will be displayed	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 4.1.2 Integration testing for all modules

Case_Id	**Description**	**Input**	**Expected o/p**	**Actual o/p**	**Status**
1	To display the entered records of music name	Enter the option 1 In the menu. Enter the Id of the Music	Should display the record of music based on Id	Display the record of music based on ID	Pass
2	To display all the music record	Enter the option 2 in menu	Should display the record entry form	Display the record entry form	Pass
3	To display the entered records of	Enter the option 1 in the menu Enter the Id of the	Should display the record of	Display the record of music	Pass

	music to user	music	music name based on Id	name based on ID	
4	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
5	To quit the program	Enter the option 0 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 testing 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 4.1.3: System Testing for Music record details

Case id	Description	Input data	Expected o/p	Actual o/p	Status
1	To display the records	Display records for valid id “(present)	Record is displayed for Valid and record not found for invalid ID	Record is displayed for valid ID and record not found for invalid ID	Pass
2	To search the records	Display records for valid ID “(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 ID = (present)	Record is deleted for valid and record not found for invalid	Record is deleted 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 systems 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.

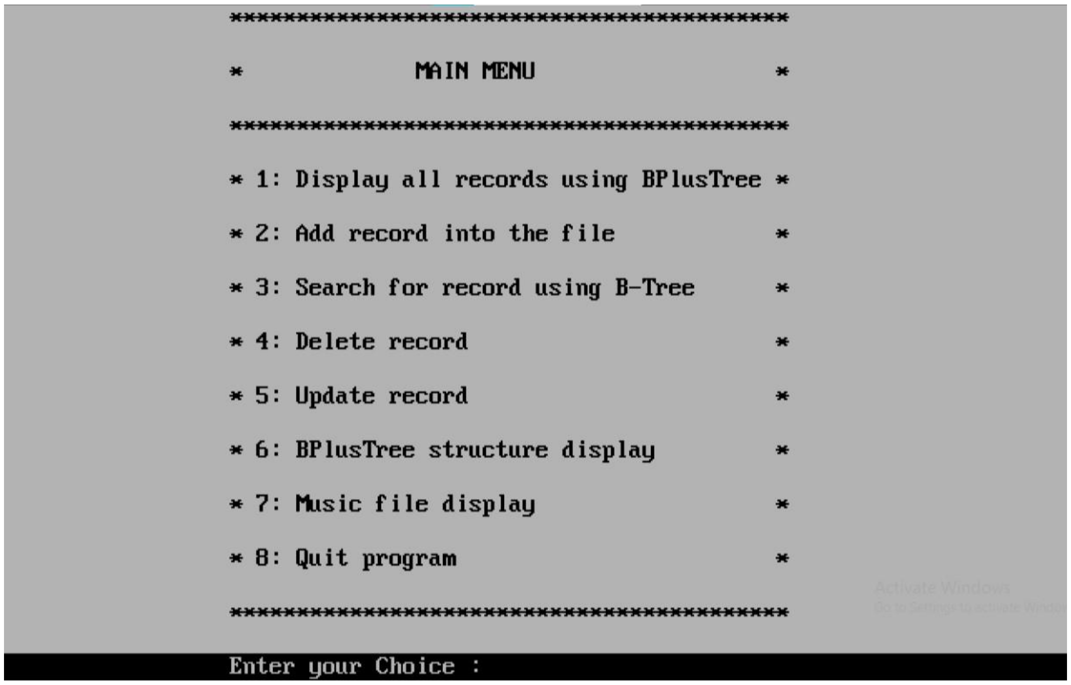
Table - 4.1.4: Acceptance testing for Hospital record details

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 id.	Record deleted successfully	Record deleted successfully	Pass
3.	Display Module	Enter valid id.	Record displayed successfully	Record displayed successfully	Pass

CHAPTER - 4

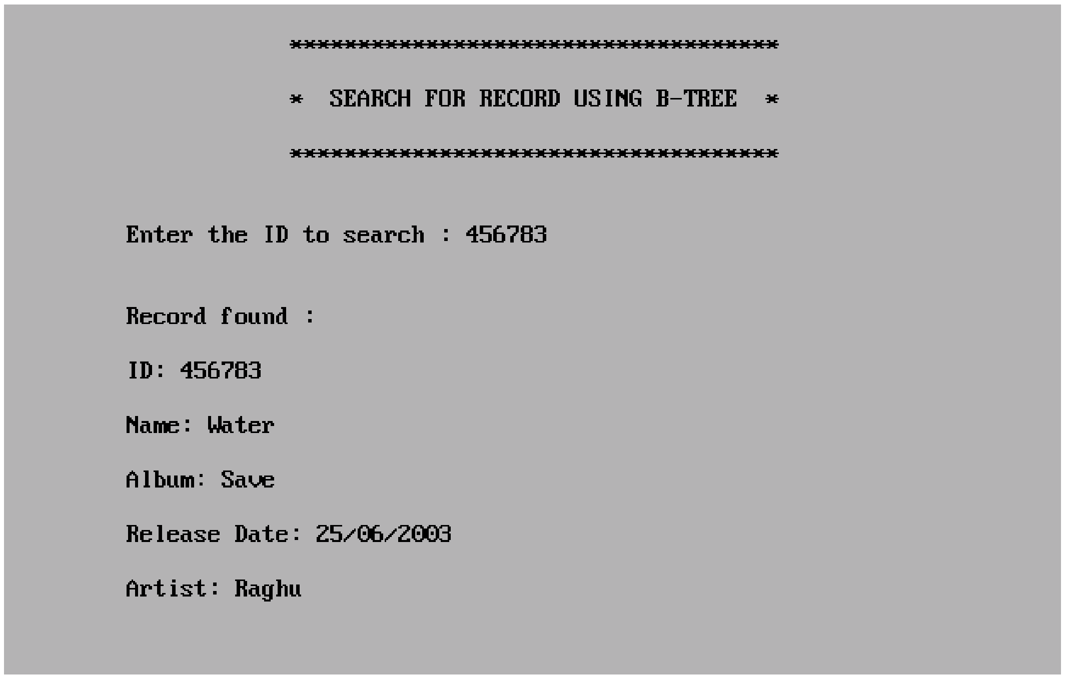
SNAPSHOTS

Fig- 4.1: Main Menu



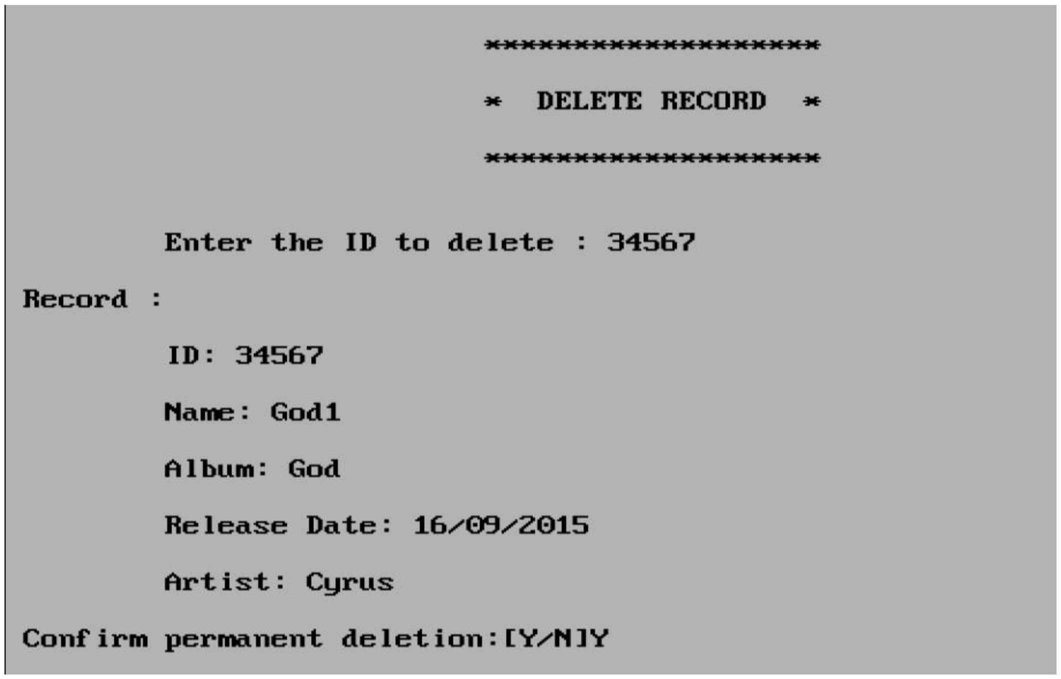
```
*******************************************
*              MAIN MENU                  *
*******************************************
* 1: Display all records using BPlusTree *
* 2: Add record into the file            *
* 3: Search for record using B-Tree      *
* 4: Delete record                       *
* 5: Update record                       *
* 6: BPlusTree structure display         *
* 7: Music file display                  *
* 8: Quit program                        *
*******************************************
Enter your Choice :
```

Fig - 4.2: Search for a record using B Tree



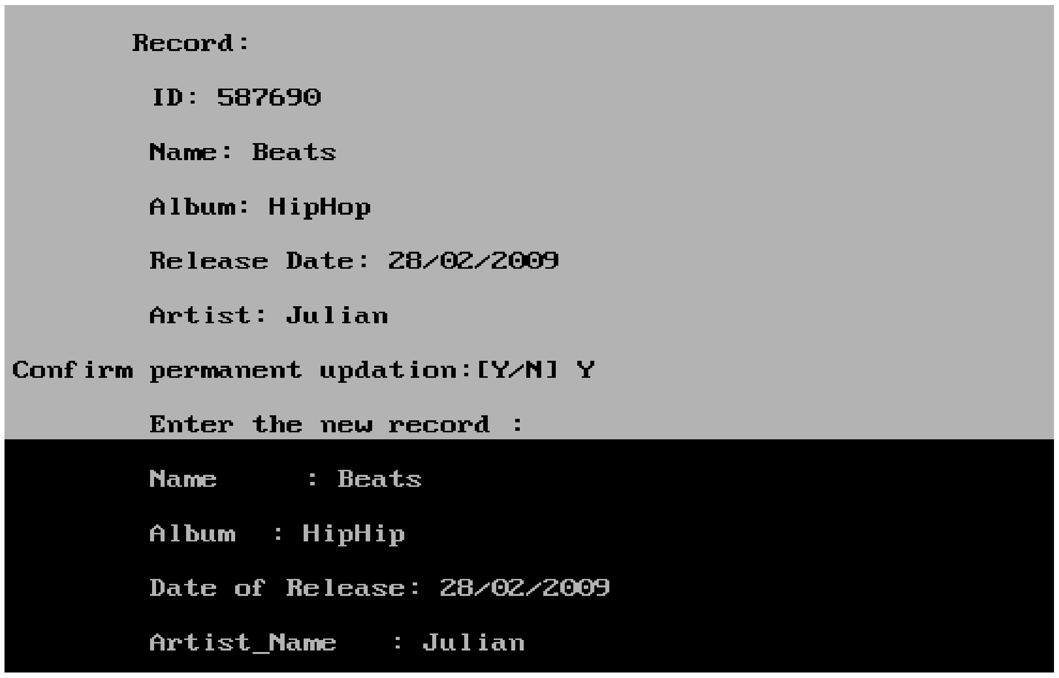
```
        **************************************
        *  SEARCH FOR RECORD USING B-TREE  *
        **************************************

Enter the ID to search : 456783

Record found :
ID: 456783
Name: Water
Album: Save
Release Date: 25/06/2003
Artist: Raghu
```

Fig - 4.3: Deleting a record



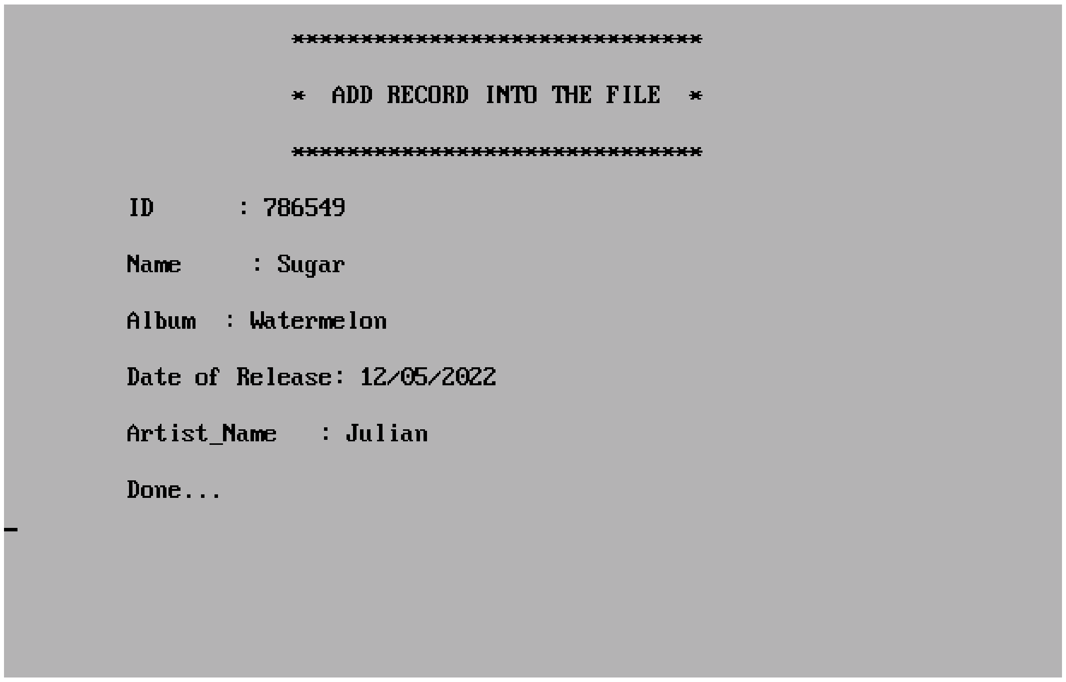
```
                              ********************
                              *  DELETE RECORD   *
                              ********************

          Enter the ID to delete : 34567
Record :
          ID: 34567
          Name: God1
          Album: God
          Release Date: 16/09/2015
          Artist: Cyrus
Confirm permanent deletion:[Y/N]Y
```

Fig – 4.4: Updating a record



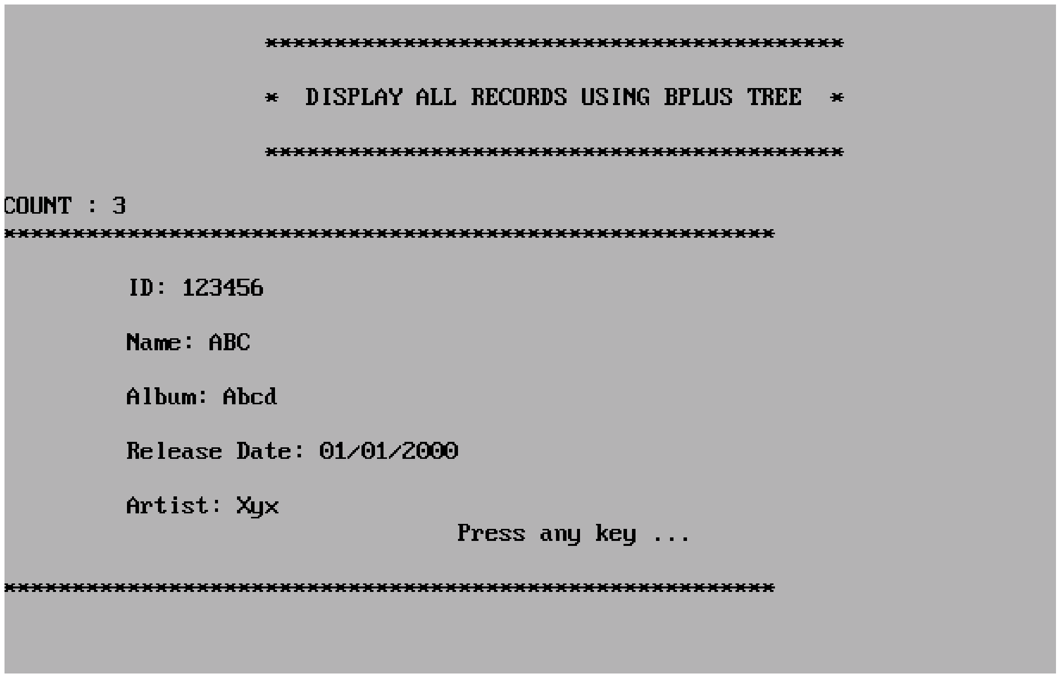
```
        Record:
         ID: 587690
         Name: Beats
         Album: HipHop
         Release Date: 28/02/2009
         Artist: Julian
Confirm permanent updation:[Y/N] Y
         Enter the new record :
         Name      : Beats
         Album   : HipHip
         Date of Release: 28/02/2009
         Artist_Name    : Julian
```

Fig - 4.5: Entering the record details



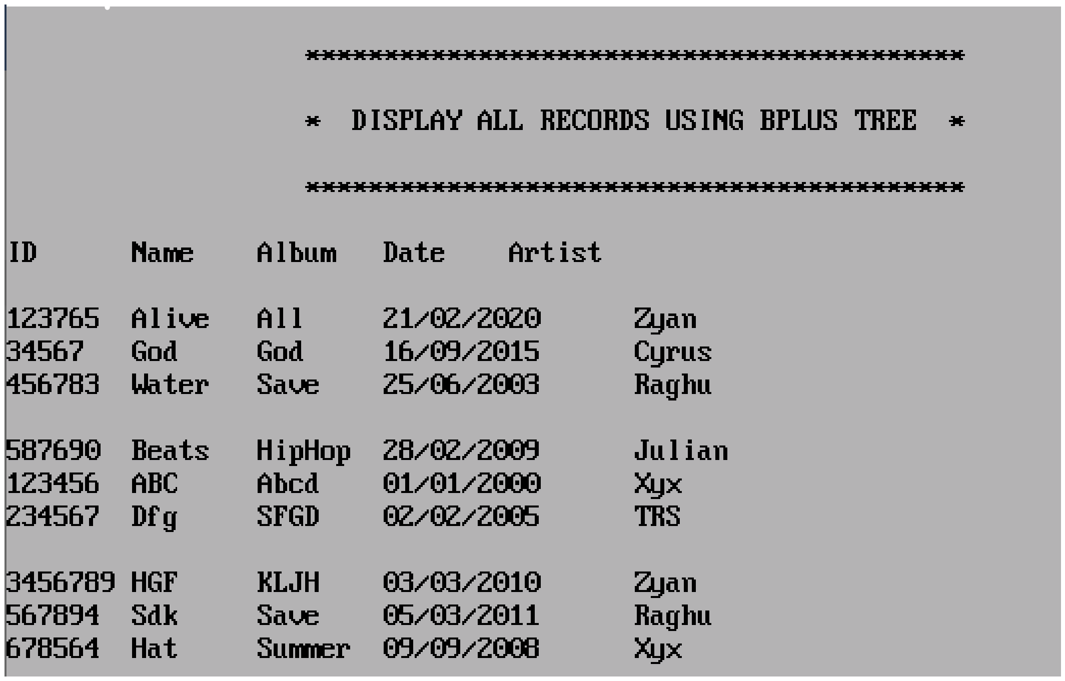
```
                    ******************************
                    *  ADD RECORD INTO THE FILE  *
                    ******************************

         ID      : 786549

         Name     : Sugar

         Album  : Watermelon

         Date of Release: 12/05/2022

         Artist_Name   : Julian

         Done...
_
```

Fig - 4.6. : Displaying all the records using BPlus Tree



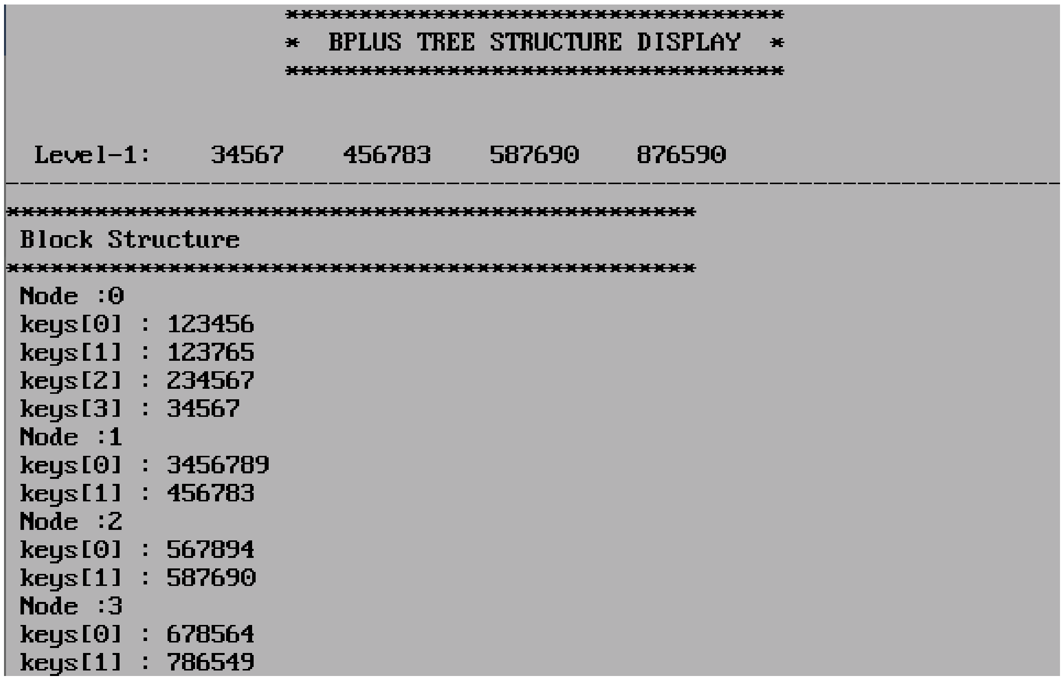
```
                   ******************************************
                   *  DISPLAY ALL RECORDS USING BPLUS TREE  *
                   ******************************************

COUNT : 3
*********************************************************

         ID: 123456

         Name: ABC

         Album: Abcd

         Release Date: 01/01/2000

         Artist: Xyx
                                 Press any key ...

*********************************************************
```

Fig – 4.7 : Displaying all the records using BPlus Tree



```
                    *******************************************

                    *  DISPLAY ALL RECORDS USING BPLUS TREE  *

                    *******************************************

ID      Name    Album   Date    Artist

123765  Alive   All     21/02/2020      Zyan
34567   God     God     16/09/2015      Cyrus
456783  Water   Save    25/06/2003      Raghu

587690  Beats   HipHop  28/02/2009      Julian
123456  ABC     Abcd    01/01/2000      Xyx
234567  Dfg     SFGD    02/02/2005      TRS

3456789 HGF     KLJH    03/03/2010      Zyan
567894  Sdk     Save    05/03/2011      Raghu
678564  Hat     Summer  09/09/2008      Xyx
```

Fig - 4.8: BPlus Tree structure display



```
                  **********************************
                  *  BPLUS TREE STRUCTURE DISPLAY  *
                  **********************************


 Level-1:    34567    456783    587690    876590
------------------------------------------------------------------------
***********************************************
 Block Structure
***********************************************
 Node :0
 keys[0] : 123456
 keys[1] : 123765
 keys[2] : 234567
 keys[3] : 34567
 Node :1
 keys[0] : 3456789
 keys[1] : 456783
 Node :2
 keys[0] : 567894
 keys[1] : 587690
 Node :3
 keys[0] : 678564
 keys[1] : 786549
```

CHAPTER - 5

CONCLUSION AND FUTURE ENHANCEMENTS

In conclusion, the use of B Plus-trees in music record management systems has proven to be an effective approach. The system is able to handle large amounts of data efficiently and effectively, and is able to provide fast and accurate data retrieval and insertion. B-trees have been shown to be particularly useful in managing large music collections, and have been used successfully in a variety of settings, including music stores, libraries, and personal collections. Overall, the use of BPlus-trees in music record management systems offers a number of benefits, including improved system performance, reduced data retrieval times, and increased accuracy and efficiency.

1. Integration with streaming services: The system could be enhanced to allow users to stream music directly from the system, providing a more seamless listening experience.

2. Improved search functionality: The search functionality could be improved to allow users to search for music based on a wider range of criteria, such as album art, genre, and release date.

3. Social media integration: The system could be enhanced to allow users to share their music collections and playlists on social media platforms, such as Facebook and Twitter.

4. Mobile app: A mobile app could be developed to allow users to access their music collections on the go, and to provide additional functionality, such as the ability to download music for offline listening.

5. User interface improvements: The system's user interface could be improved to make it more intuitive and user-friendly, with features such as drag-and-drop functionality and customizable playlists

REFERENCES

Books:

1.File Structures with C++ by Michael J Folk.

2.File Structures using C++ by K R Venugopal, K G Srinivasa, P M Krishnaraj, Tata McGraw-Hill, 2009, Referred Page No-4, 39-40.

3.https://www.geeksforgeeks.org/indexing-in-databases-set-1/

4.https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm#:~:text=Algorithm%20is%20a%20step%2Dby,more%20than%20one%20program